TMA521/MMA510
Optimization, project course

Introduction: simple/difficult problems, matroid problems

Michael Patriksson and Ann-Brith Strömberg

2009–09–02

TMA521/MMA510 Optimization, project course

- $\approx$ 3 meetings/lectures per week during three–four weeks. Schedule on the course homepage: `www.math.chalmers.se/Math/Grundutb/CTH/tma521/0910/`
- Two projects:
  - Lagrangian relaxation for a VLSI design problem (Matlab)
  - Large-scale set covering problems: heuristics and optimizing methods (competition!)
- Literature: Optimization theory for large systems (Lasdon, 2002, Cremona), lecture notes, hand-outs from books and articles.
- Examination: Written reports on the two projects. Oral presentations and opposition!
- For higher grades than pass (4, 5, VG): oral exam.

Topics: Turning difficult problems into a sequence of simpler problems (decomposition–coordination)

- Lagrangian relaxation (IP, NLP)
- Dantzig–Wolfe decomposition (LP)
- Benders decomposition (IP, NLP)
- Column generation (LP, IP, NLP)
- Heuristics (IP)
- Branch & Bound (IP, non-convex NLP)
- Greedy algorithms (IP, NLP)
- Subgradient optimization (convex NLP, Lagrangian duals)

Simple problems—Wolsey

- For simple problems, there exist polynomial algorithms (they belong to the complexity class $\mathcal{P}$), preferably with a small largest exponent.

- Network flow problems (shortest paths; maximum flows; minimum cost (single-commodity) network flows; transportation problem; assignment problem; maximum cardinality matching). See Wolsey!

- Linear programming

- Problems over simple matroids (next!)

Matroids and the greedy algorithm (Lawler)

- *Greedy algorithm:* Create a "complete solution" by iteratively choosing the best alternative. *Never regret* a previous choice.

- Which problems can be solved using such a simple method?

– Problems whose feasible sets can be described by *matroids*.

Matroids and independent sets

- Given a finite set $\mathcal{E}$ and a family $\mathcal{F}$ of subsets of $\mathcal{E}$.
If $\mathcal{I} \in \mathcal{F}$ and $\mathcal{I}' \subseteq \mathcal{I}$ imply $\mathcal{I}' \in \mathcal{F}$, then the elements of $\mathcal{F}$ are called *independent*.

- A *matroid* $M = (\mathcal{E}, \mathcal{F})$ is a structure in which $\mathcal{E}$ is a finite set of *elements* and $\mathcal{F}$ is a *family of subsets* of $\mathcal{E}$, such that

  1. $\emptyset \in \mathcal{F}$ and all proper subsets of a set $\mathcal{I}$ in $\mathcal{F}$ are in $\mathcal{F}$.
  2. If $\mathcal{I}_p$ and $\mathcal{I}_{p+1}$ are sets in $\mathcal{F}$ with $|\mathcal{I}_p| = p$ and $|\mathcal{I}_{p+1}| = p+1$, then $\exists$ an element $e \in \mathcal{I}_{p+1} \setminus \mathcal{I}_p$ such that $\mathcal{I}_p \cup \{e\} \in \mathcal{F}$.

- Let $M = (\mathcal{E}, \mathcal{F})$ be a matroid and $\mathcal{A} \subseteq \mathcal{E}$. If $\mathcal{I}$ and $\mathcal{I}'$ are *maximal independent subsets* of $\mathcal{A}$, then $|\mathcal{I}| = |\mathcal{I}'|$.

Matroids—Example I:

$\mathcal{E}$ = a set of column vectors in $\mathbb{R}^n$

$\mathcal{F}$ = the set of linearly independent subsets of vectors in $\mathcal{E}$.

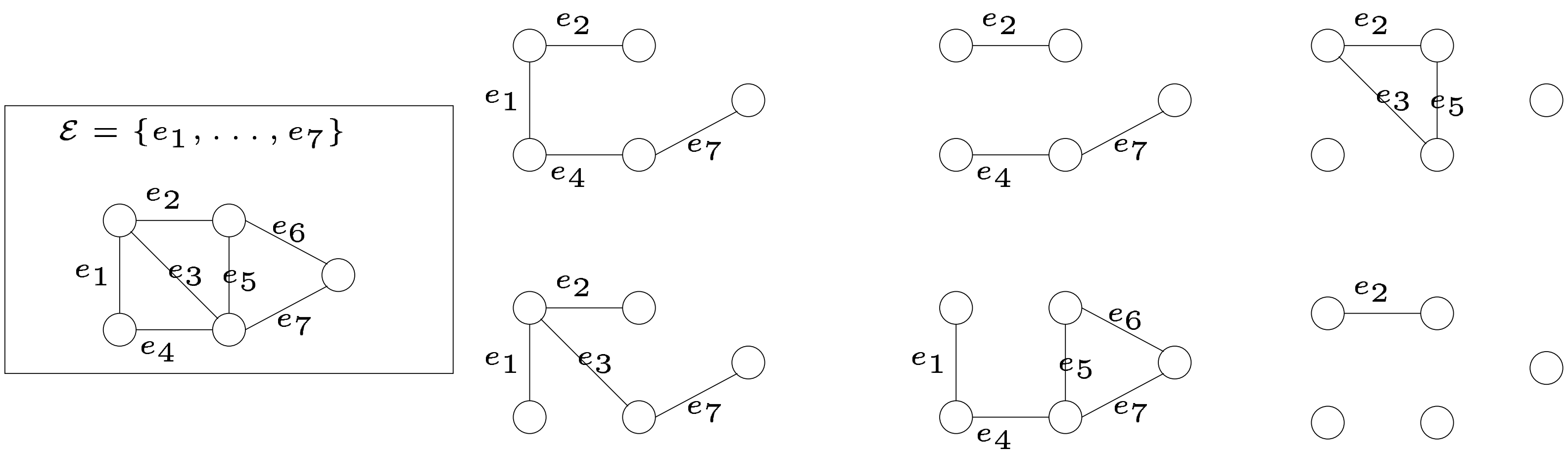
Example $n = 3$ and $\mathcal{E} = [e_1, \ldots, e_5] = \begin{bmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 3 & 2 & 0 & 0 & 1 \end{bmatrix}$

We have $\{e_1, e_2, e_3\} \in \mathcal{F}$ and $\{e_2, e_3\} \in \mathcal{F}$ but $\{e_1, e_2, e_3, e_5\} \notin \mathcal{F}$ and $\{e_1, e_4, e_5\} \notin \mathcal{F}$

Matroids—Example II:

$\mathcal{E}$ = the set of links (edges, arcs) in an undirected graph = $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$\mathcal{F}$ = the set of all cycle-free subsets of links in $\mathcal{E}$

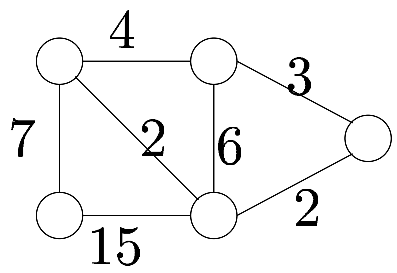


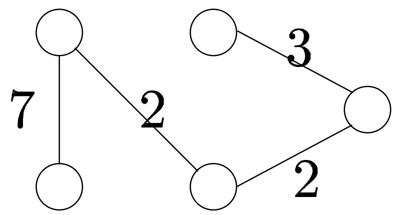
$\{e_1, e_2, e_4, e_7\} \in \mathcal{F}$, $\{e_2, e_4, e_7\} \in \mathcal{F}$, $\{e_2, e_3, e_5\} \notin \mathcal{F}$,
$\{e_1, e_2, e_3, e_7\} \in \mathcal{F}$, $\{e_1, e_4, e_5, e_6, e_7\} \notin \mathcal{F}$, $\{e_2\} \in \mathcal{F}$.

Matroids and the greedy algorithm—Example II:

- Let $w(e)$ be the cost of element $e \in \mathcal{E}$.
  Problem: Find the element $\mathcal{I} \in \mathcal{F}$ of *maximal cardinality* such that the total cost is at minimum/maximum.

- Example II–continued: $w(\mathcal{E}) = (7, 4, 2, 15, 6, 3, 2)$





An element $\mathcal{I} \in \mathcal{F}$ of maximal

cardinality with minimum total cost

The Greedy algorithm for minimization problems

1. $\mathcal{A} = \emptyset$.
2. Sort the elements of $\mathcal{E}$ in increasing order with respect to $w(e)$.
3. Take the first element $e \in \mathcal{E}$ in the list. If $\mathcal{A} \cup \{e\}$ is still independent $\Longrightarrow$ let $\mathcal{A} := \mathcal{A} \cup \{e\}$.
4. Repeat from step 3. with the next element—until either the list is empty, or $\mathcal{A}$ has the maximal cardinality.

What are the corresponding algorithms in Examples I and II?

Examples

- Example I (linearly independent vectors): Let

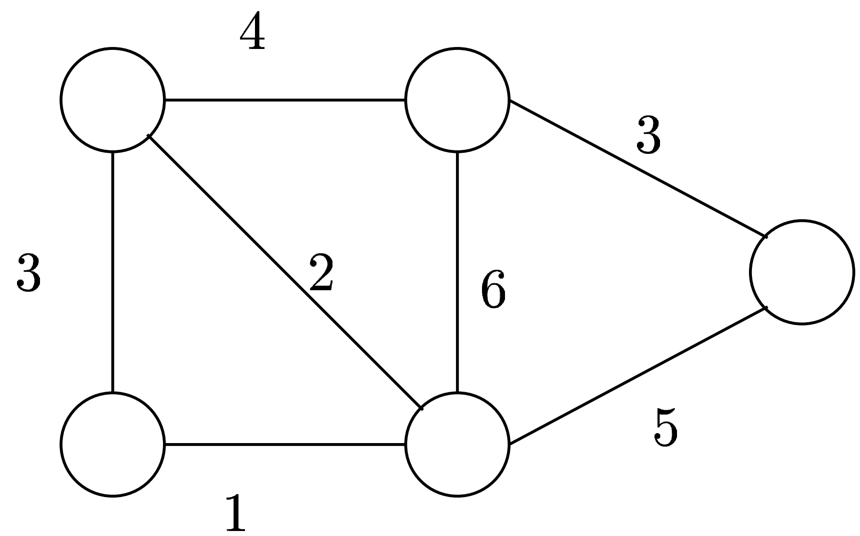
$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 2 & 0 & 1 \\ 0 & -1 & -1 & 1 & 1 \\ 3 & 2 & 8 & 1 & 4 \\ 2 & 1 & 5 & 0 & 2 \end{pmatrix},$$

$$\boldsymbol{w}^{\mathrm{T}} = \begin{pmatrix} 10 & 9 & 8 & 4 & 1 \end{pmatrix}.$$

- Choose the maximal independent set with the maximal weight.

- Can this technique solve linear programming problems?

- Example II (minimum spanning trees): The maximal set of cycle-free links in an undirected graph is a *spanning tree*; in a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, it has $|\mathcal{N}| - 1$ links.
- Classic greedy algorithm (Kruskal's algorithm) has complexity $O(|\mathcal{E}| \cdot \log(|\mathcal{E}|))$. The main cost is in the sorting itself.
- Prim's algorithm builds the spanning tree through graph search techniques, from node to node; complexity $O(|\mathcal{N}|^2)$.



- Example III (in fact not a matroid problem):
  Continuous relaxation of the 0/1-knapsack problem (BKP):

$$\text{maximize } f(\boldsymbol{x}) := \sum_{j=1}^{n} c_j x_j,$$

$$\text{subject to } \sum_{j=1}^{n} a_j x_j \leq b, \qquad (a_j, b \in \mathcal{Z}_+)$$

$$0 \leq x_j \leq 1, \quad j = 1, \ldots, n.$$

- Greedy algorithm: Sort c_j/a_j in descending order; set the variables to 1 until the knapsack is full. One variable may become fractional and the rest zero.

- Linear programming duality shows that the greedy algorithm is correct.

Linear programming dual:

$$\begin{array}{llcrcll}
\text{minimize} & bu & + & \sum_{j=1}^{n} w_j, & & & \\
\text{subject to} & a_j u & + & w_j & \geq & c_j, & j = 1, \ldots, n, \\
 & u & & & \geq & 0, & \\
 & & & w_j & \geq & 0, & j = 1, \ldots, n
\end{array}$$

Hint: Complementarity slackness.

- Rounding down gives a feasible solution to (BKP).
  Is it also optimal in (BKP)?

$$\begin{aligned} \text{maximize } & f(\boldsymbol{x}) := 2x_1 + cx_2, \\ \text{subject to } & x_1 + cx_2 \leq c, \qquad (c \in \mathcal{Z}_+) \\ & x_1, x_2 \in \{0, 1\}, \end{aligned}$$

- If $c \geq 2$ then $\boldsymbol{x}^* = (0, 1)^{\mathrm{T}}$ and $f^* = c$.
- The greedy algorithm, plus rounding, always gives $\bar{\boldsymbol{x}} = (1, 0)^{\mathrm{T}}$, with $f(\bar{\boldsymbol{x}}) = 2$; an arbitrarily bad solution (for c large).

- Example IV: the traveling salesman problem (TSP)
- The greedy algorithm would select the next best city which does not lead to a sub-tour. Optimal?

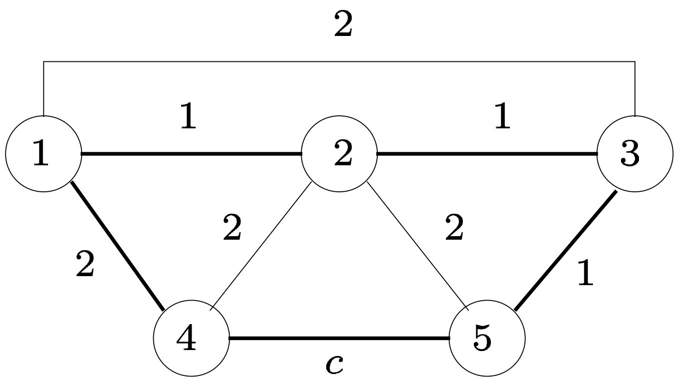
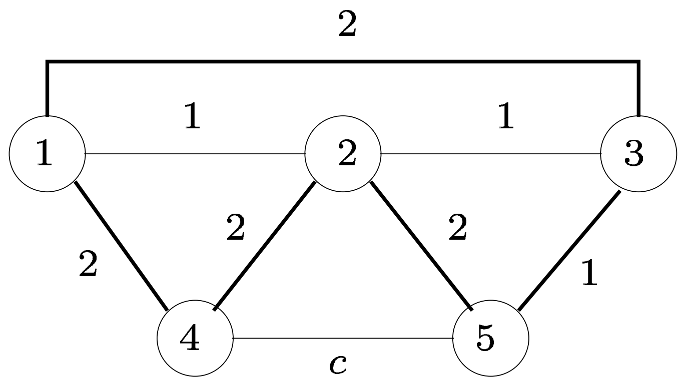


Figure 1: Greedy



Optimal when $c \geq 4$

- Not optimal when $c \gg 0$.

- Example V: the shortest path problem (SPP)
- The greedy algorithm constructs a path that uses, locally, the cheapest link to reach a new node. Optimal?

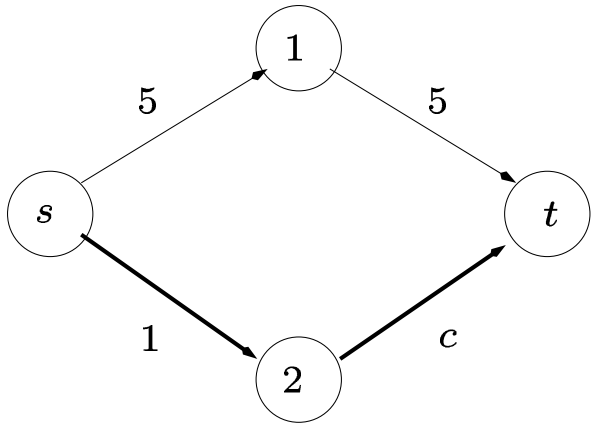
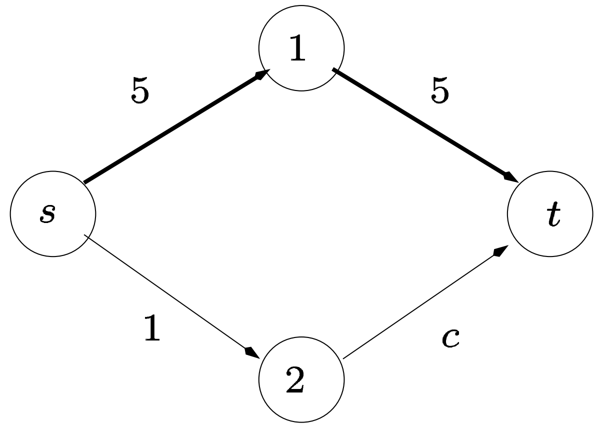


Figure 2: Greedy



Optimal when $c \geq 9$

- Not optimal when $c \gg 0$.

- Example VI: Semi-matching:

$$\text{maximize } f(\boldsymbol{x}) := \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij},$$

$$\text{subject to } \sum_{j=1}^{n} x_{ij} \leq 1, \quad i = 1, \dots, m,$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \dots, m, \; j = 1, \dots, n.$$

- Semi-assignment: replace maximum $\Longrightarrow$ minimum; "$\leq$" $\Longrightarrow$ "$=$"; $m = n$.

- Algorithm:
  For each i: choose the best (lowest) w_{ij}, set $x_{ij} = 1$ for that j, and $x_{ij} = 0$ for every other j.

Matroid types

- *Graph matroid*: $\mathcal{F}$ = the set of forests in a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$.
  Example problem: MST.

- *Partition matroid*: Consider a partition of $\mathcal{E}$ into m sets $\mathcal{B}_1, \ldots, \mathcal{B}_m$ and let d_i $(i = 1, \ldots, m)$ be non-negative integers. Let

$$\mathcal{F} = \{\mathcal{I} \mid \mathcal{I} \subseteq \mathcal{E}; \quad |\mathcal{I} \cap \mathcal{B}_i| \leq d_i,\ i = 1, \ldots, m\}.$$

  Example problems: semi-matching in bipartite graphs.

- *Matrix matroid*: $S = (\mathcal{E}, \mathcal{F})$, where $\mathcal{E}$ is a set of column vectors and $\mathcal{F}$ is the set of subsets of $\mathcal{E}$ with linearly independent vectors.
  Observe: The above matroids can be written as matrix matroids!

Problems over matroid intersections

- Given two matroids $M = (\mathcal{E}, \mathcal{P})$ and $N = (\mathcal{E}, \mathcal{R})$, find the maximum cardinality set in $\mathcal{P} \cap \mathcal{R}$.
- *Example 1*: maximum-cardinality matching in a bipartite graph is the intersection of two partition matroids (with $d_i = 1$).
- The intersection of two matroids can not be solved by using the greedy algorithm.
- There exist polynomial algorithms for them. For example, bipartite matching and assignment problems can be solved as maximum flow problems, which are polynomially solvable.

- *Example 2*: The traveling salesman problem (TSP) is the intersection of three matroids: a graph matroid and two partition matroids (see its formulation using assignment + tree constraints).
- TSP is *not* solvable in polynomial time.
- *Conclusion*:
  - Matroid problems are extremely easy to solve
  - Two-matroid problems are polynomially solvable
  - Three-matroid problems are very difficult!

The traveling salesman problem—three different mathematical formulations

Different formulations of the (undirected) TSP, which give rise to different algorithms when Lagrangian relaxed or otherwise manipulated.

Tree-based formulation
(1)–(2): assignment; (3): cycle-free

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, \qquad i \in \mathcal{N}, \qquad (1)\\
& \sum_{i=1}^{n} x_{ij} = 1, \qquad j \in \mathcal{N}, \qquad (2)\\
& \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}} x_{ij} \le |\mathcal{S}| - 1, \quad \mathcal{S} \subset \mathcal{N}, \qquad (3)\\
& x_{ij} \in \{0,1\}, \qquad i,j \in \mathcal{N}.
\end{aligned}$$

- Relax (3): Assignment.
- Relax (1)–(2): 1-MST, if adding redundant constraints from the original problem.

Node adjacency based formulation. (1): Adjacency condition; (2): Redundant; (3): cycle-free (alternative version)

[Hamilton cycle = spanning tree + one link: every node adjacent to two nodes]

$$
\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
\text{subject to} & \sum_{j=1}^{n} x_{ij} = 2, & i \in \mathcal{N}, & (1) \\
& \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} = n, & & (2) \\
& \displaystyle\sum_{(i,j)\in(\mathcal{S},\mathcal{N}\setminus\mathcal{S})} x_{ij} \geq 1, & \mathcal{S} \subset \mathcal{N}, & (3) \\
& x_{ij} \in \{0,1\}, & i,j \in \mathcal{N}. &
\end{array}
$$

- Relax (1), except for node s: 1-tree relaxation.
- Relax (3): 2-matching.

Tree-based formulation for directed graphs
(1)–(2): assignment; (3): Redundant; (4) Cycle-free

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{E}} c_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{j:(i,j)\in\mathcal{E}} x_{ij} = 1, \qquad i \in \mathcal{N}, \qquad (1)$$

$$\sum_{i:(i,j)\in\mathcal{E}} x_{ij} = 1, \qquad j \in \mathcal{N}, \qquad (2)$$

$$\sum_{(i,j)\in\mathcal{E}} x_{ij} = |\mathcal{N}|, \qquad (3)$$

$$\sum_{(i,j)\in(\mathcal{S},\mathcal{N}\setminus\mathcal{S})^+} x_{ij} + \sum_{(j,i)\in(\mathcal{S},\mathcal{N}\setminus\mathcal{S})^-} x_{ij} \geq 1, \qquad \mathcal{S} \subset \mathcal{N}, \qquad (4)$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in \mathcal{E}.$$

- Relax (1) or (2), plus (4): semi-assignment.
- Relax (3) plus (4): assignment.
- Relax (1), and (2) except for node s: directed 1-tree relaxation.