TMA521/MMA511 Large Scale Optimization Lecture 1 Introduction: simple/difficult problems, matroid problems

Ann-Brith Strömberg
Associate Professor
Applied Mathematics/Optimization

2013–01–21

TMA521/MMA511 Optimization, project course

- **Examiner/lecturer:** Ann-Brith Strömberg (room L2087, anstr@chalmers.se)
- **Lecturers:** Michael Patriksson, Karin Thörnblad
- **Schedule:** 12 lectures, 3 seminars/workshops
  `www.math.chalmers.se/Math/Grundutb/CTH/tma521/1213/`
- **Two projects:**
  - Lagrangian relaxation for a VLSI design problem (Matlab)
  - Column generation applied to a real production scheduling problem (AMPL/Cplex, Matlab)
- **Literature:** *Optimization theory for large systems* (Lasdon, 2002, Cremona), *An introduction to continuous optimization* (Andréasson et al., Cremona), hand-outs from books and articles, lecture notes
- **Examination:** Written reports on the two projects, oral presentations and oppositions
- For **higher grades** than pass (4, 5, VG): oral exam

Topics: Turn difficult problems into sequences of simpler ones using decomposition and coordination

Prerequisites

- Linear Programming (LP), [Mixed] Integer Linear programming ([M]ILP), NonLinear Programming (NLP),

Decomposition methods covered

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

Properties of simple problems

- What we here call *simple problems* can be solved in polynomial time w.r.t. the problem size
- For *simple problems*, there exist *polynomial algorithms* preferably with a small largest exponent
- E.g., sorting an array of n elements can be done in time proportional to at most
  - n^2 operations (bubble sort)
  - $n \log n$ operations (heapsort)

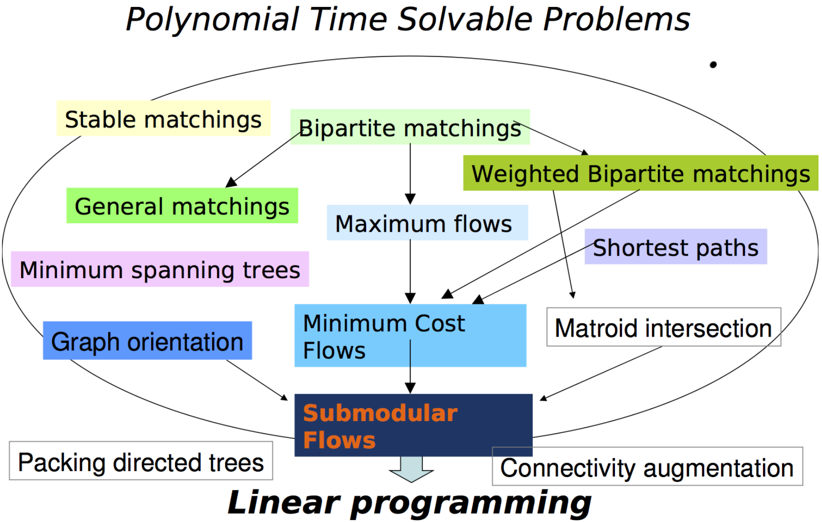
Examples of simple problems

- Network flow problems (see Wolsey):
  - Shortest paths
  - Maximum flows
  - Minimum cost (single-commodity) network flows
  - The transportation problem
  - The assignment problem
  - Maximum cardinality matching

- Problems over simple matroids (see Lawler)

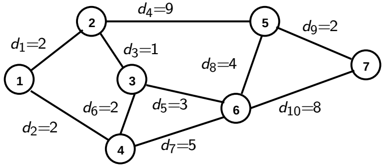
- Linear programming (see Andréasson et al.)

Polynomial time solvable problems



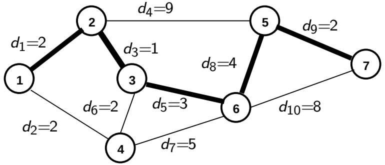
Example: Shortest path

Find the shortest path from node 1 to node 7



d_i = length of edge i

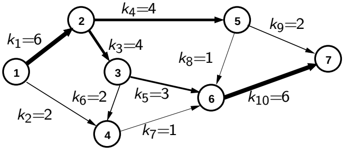
Shortest path from node 1 to node 7



Total length: 12

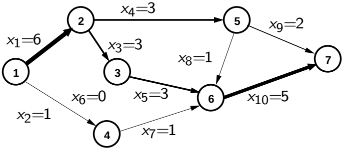
Example: Maximum flow

Find the maximum flow from node 1 to node 7



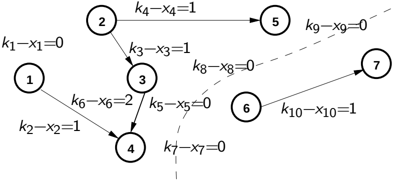
k_i = flow capacity of arc i

Maximum flow from node 1 to node 7



x_i = optimal flow through arc i

Minimum cut separating nodes 1 and 7



$k_i - x_i$ = residual flow capacity on arc i

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 $\mathcal{I}_p \cup \{e\} \in \mathcal{F}$

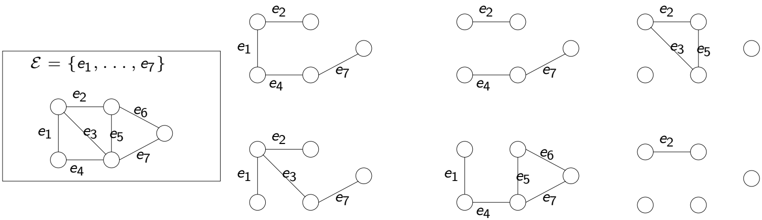
- Let $M = (\mathcal{E}, \mathcal{F})$ be a matroid and $\mathcal{A} \subseteq \mathcal{E}$.
  If $\mathcal{I}$ and $\mathcal{I}'$ are maximal independent subsets of $\mathcal{A}$, then $|\mathcal{I}| = |\mathcal{I}'|$

Example I: Matric matroids

- $\mathcal{E}$ = a set of column vectors in $\mathbb{R}^n$
- $\mathcal{F}$ = the set of linearly independent subsets of vectors in $\mathcal{E}$.
- Let $n = 3$ and $\mathcal{E} = [e_1, \dots, e_5] = \begin{bmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 3 & 2 & 0 & 0 & 1 \end{bmatrix}$
- 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 $\{e_1, e_4, e_5\} \notin \mathcal{F}$

Example II: Graphic matroids

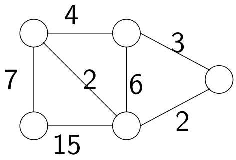
- $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ = the set of edges in an undirected graph
- $\mathcal{F}$ = the set of all cycle-free subsets of edges in $\mathcal{E}$

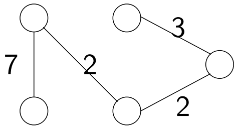


- $\{e_1, e_2, e_4, e_7\} \in \mathcal{F}$, $\{e_2, e_4, e_7\} \in \mathcal{F}$, $\{e_2, e_3, e_5\} \notin \mathcal{F}$, $\{e_1, e_2, e_3, e_7\} \in \mathcal{F}$, $\{e_1, e_4, e_5, e_6, e_7\} \notin \mathcal{F}$, $\{e_2\} \in \mathcal{F}$.

Matroids and the greedy algorithm applied to 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 $\mathcal{A}$ possesses the maximal cardinality.

Which are the special versions of this algorithm for Examples I and II?

Example I: Linearly independent vectors—matric matroids

- Let

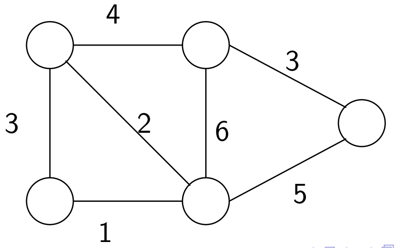
$$
\mathbf{A} = \begin{pmatrix} 1 & 0 & 2 & 0 & 1 \\ 0 & -1 & -1 & 1 & 1 \\ 3 & 2 & 8 & 1 & 4 \\ 2 & 1 & 5 & 0 & 2 \end{pmatrix},
$$

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

- Choose the maximal independent set with the maximum weight
- Can this technique solve linear programming problems?

Example II: minimum spanning trees (MST) —graphic matroids

- The maximal cycle-free set of 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: continuous knapsack problem (in fact not a matroid problem)

- Continuous relaxation of the 0/1-knapsack problem (BKP):

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

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

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

- Greedy algorithm:
  1. Sort c_j/a_j in descending order
  2. Set the variables to 1 until the knapsack is full
  3. One variable may become fractional and the rest zero
- Linear programming duality shows that the greedy algorithm solves the problem correctly

Example III, continued

Linear programming dual:

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

Hint: Complementarity slackness.

Example III, continued: Binary knapsack problem

- Rounding down the fractional variable value yields a feasible solution to (BKP)
- Is it also optimal in (BKP)?

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

- If $c \ge 2$ then $\mathbf{x}^* = (0,1)^{\mathrm{T}}$ and $f^* = c$.
- The greedy algorithm, plus rounding, always yields $\bar{\mathbf{x}} = (1,0)^{\mathrm{T}}$, with $f(\bar{\mathbf{x}}) = 2$
- This solution is arbitrarily bad (when c is large)

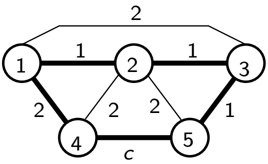
Example IV: The traveling salesperson problem (TSP)

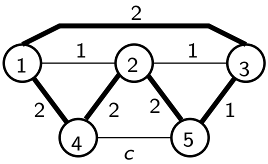
The greedy algorithm for the TSP:

1. Start in node 1
2. Go to the nearest node which is not yet visited
3. Repeat step 2 until no nodes are left
4. Return to node 1; the tour is closed

- Greedy solution

  Not optimal whenever $c > 4$.



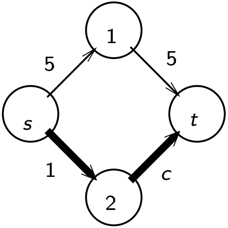


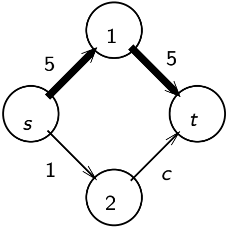
Optimal solution for $c \geq 4$

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?
- Greedy solution

  Not optimal whenever $c > 9$





Optimal solution for $c \geq 9$

Example VI: Semi-matching

$$
\begin{aligned}
\text{maximize } f(\mathbf{x}) &:= \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}, \\
\text{subject to } \sum_{j=1}^{n} x_{ij} &\leq 1, \quad i = 1, \dots, m, \\
x_{ij} &\in \{0, 1\}, \quad i = 1, \dots, m, \ j = 1, \dots, n.
\end{aligned}
$$

- Semi-assignment
  Replace maximum $\Longrightarrow$ minimum; "$\leq$" $\Longrightarrow$ "$=$"; let $m = n$
- Algorithm
  For each i:
  1. choose the best (lowest) w_{ij}
  2. 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 problem: 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 expressed 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 $d_i = 1$). DRAW ILLUSTRATION!
- The intersection of two matroids can *not* be solved by using the *greedy algorithm*
- There exist *polynomial algorithms* for them, though
- *Examples*: bipartite matching and assignment problems can be solved as maximum flow problems, which are polynomially solvable

Problems over matroid intersections, cont.

- Example 2: The traveling salesperson problem (TSP) is the intersection of three matroids:
  - one graph matroid
  - two partition matroids

  (formulation on next page: assignment + tree constraints)
- TSP is *not* solvable in polynomial time.
- Conclusion (not proven here):
  - Matroid problems are extremely easy to solve (greedy works)
  - Two-matroid problems are polynomially solvable
  - Three-matroid problems are very difficult (exponential solution time)
- The TSP—different mathematical formulations give rise to different algorithms when Lagrangean relaxed or otherwise decomposed

TSP: Assignment formulation for directed graphs

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, && i \in \mathcal{N}, && (1) \\
& \sum_{i=1}^{n} x_{ij} = 1, && j \in \mathcal{N}, && (2) \\
& \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} = n, && && (3) \\
& \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, && \mathcal{S} \subset \mathcal{N}, && (4) \\
& x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}.
\end{aligned}
$$

- (1)–(2): assignment; (3): sum of (1) (redundant); (4): cycle-free
- Relax (3)–(4) $\Rightarrow$ Assignment
- Relax (1)–(2) $\Rightarrow$ 1-MST

TSP: Node valence formulation for undirected graphs

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

- (1): valence = 2; (2): sum of (1); (3): cycle-free (alt. version)
- Hamiltonian cycle = spanning tree + one link $\Rightarrow$ every node receives valence = 2
- Relax (1), except for node s $\Rightarrow$ 1-tree relaxation.
- Relax (3) $\Rightarrow$ 2-matching.