Project course: Optimization TM Introduction: simple/difficult problems; matroid problems

Michael Patriksson

15 March 2004

Project course: Optimization TM, 2004

- $\approx$ 3 meetings per week during three–four weeks
- Projects:
  - Lagrangian relaxation for a VLSI design problem (Matlab package)
  - Large-scale set covering problems: heuristics and optimizing methods (competition!)
- Literature: Lecture notes, hand-outs from books.
- Examination: Written reports on the two projects. Oral presentation, with opposition!
- For better 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)

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. In the greedy algorithm, one never regrets a choice made previously.
- Which problems can be solved using such a simple method?
- Problems that can be described by *matroids*.
- Given a finite set $\mathcal{E}$ and a family $\mathcal{F}$ of subsets of $\mathcal{E}$. If $\mathcal{A} \in \mathcal{F}$ and $\mathcal{A}' \subseteq A$ implies that $\mathcal{A}' \in \mathcal{F}$, then the system $S = (\mathcal{E}, \mathcal{F})$ is an *independent system*.

- 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, II:

$\mathcal{E}$ = the set of links (edges, arcs) in an undirected graph,

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

- Let $w(e)$ be the cost of an element in $\mathcal{E}$. Problem: Find the element $\mathcal{A} \in \mathcal{F}$ of maximal cardinality such that the total cost is minimal/maximal.

The Greedy algorithm for minimization problems

- $\mathcal{A} = \emptyset$.
- Sort the elements of $\mathcal{E}$ in increasing order with respect to $w(e)$.
- Take the first element $e \in \mathcal{E}$ in the list. If $\mathcal{A} \cup \{e\}$ is still independent $\Longrightarrow$ let $\mathcal{A} := \mathcal{A} \cup \{e\}$.
- Continue with the next element.
- Continue 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 LP 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):
  LP relaxation of the 0/1 knapsack problem (BKP):

$$\begin{aligned} \text{maximize } f(\boldsymbol{x}) &= \sum_{j=1}^{n} c_j x_j, \\ \text{subject to } \sum_{j=1}^{n} a_j x_j &\leq b, \qquad\qquad (a_j, b \in \mathcal{Z}_+) \\ 0 \leq x_j &\leq 1, \quad j = 1, \ldots, n. \end{aligned}$$

- Greedy algorithm: Sort c_j/a_j in descending order; set the variables to 1 until the knapsack is full. The last variable may become fractional.
- LP duality shows that the greedy algorithm is correct.

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

$$
\begin{aligned}
\text{maximize } & f(\boldsymbol{x}) = 2x_1 + cx_2, \\
\text{subject to } & \sum_{j=1}^{n} x_1 + cx_2 \leq c, \\
& x_1, x_2 \in \{0, 1\},
\end{aligned}
$$

where c is a positive integer.

- 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.

- 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?

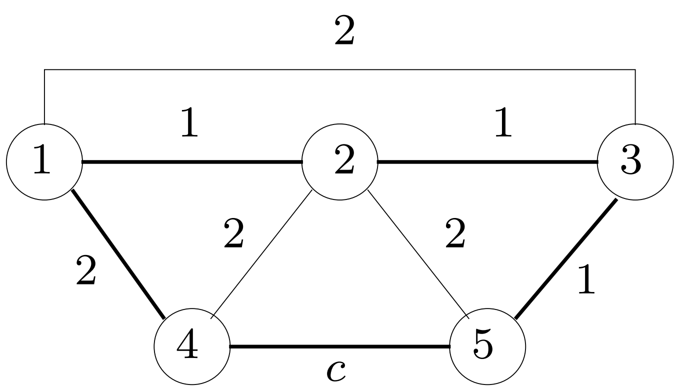
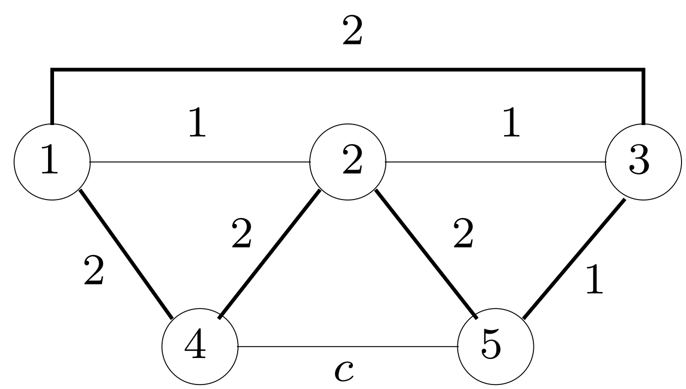


Figure 1: Greedy



Optimal

- 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?

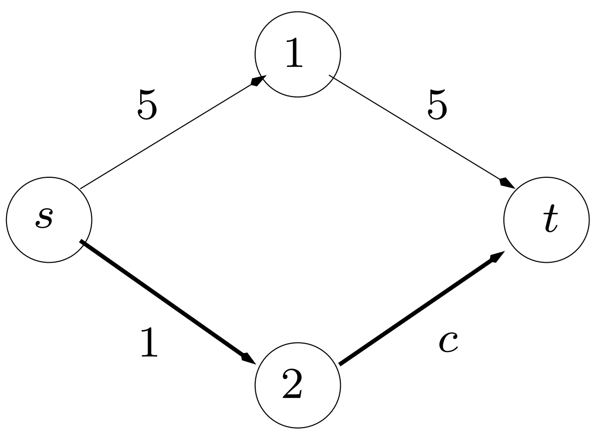
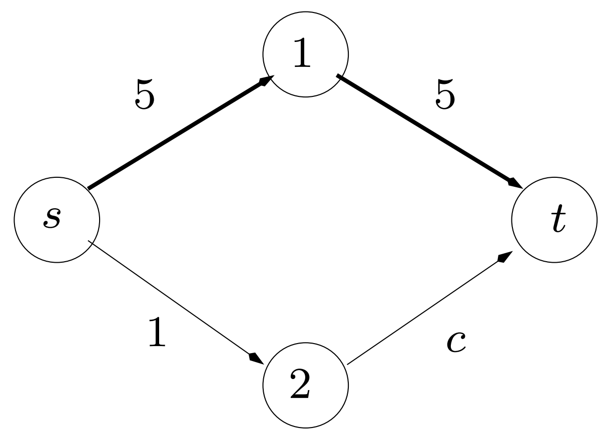


Figure 2: Greedy



Optimal

- Not optimal when $c \gg 0$.

- Example VI: Semi-matching:

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

- Semi-assignment: replace maximum $\Longrightarrow$ minimum; "$\leq$" $\Longrightarrow$ "$=$"; $m = n$.
- Algorithm: For each i: take best w_{ij}, set $w_{ij} = 1$ for that j, and $w_{ij} = 0$ 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; 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 is the intersection of two partition matroids.
- The intersection of two matroids can not be solved by using the greedy algorithm.
- There exist polynomial algorithms for them. For example, 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).
- Conclusion: Matroid problems are extremely easy; two-matroid problems are polynomial; three-matroid problems are very difficult!

The traveling salesman problem—three formulations

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

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

- Tree-based formulation. (1)–(2): Assignment; (3): cycle-free.
- Lagrangian relax (3): Assignment.
- Lagrangian relax (1)–(2): 1-MST, if adding redundant constraints from the original problem.

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

- Node adjacency based formulation. (1): Adjacency condition; (2): Redundant; (3): cycle-free (alternative version). [Hamilton cycle is a spanning tree + one link, such that every node is adjacent to two nodes.]
- Lagrangian relax (1), except for node s: 1-tree relaxation.
- Lagrangian relax (3): 2-matching.

For directed graphs:

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

- Tree-based formulation. (1)–(2): assignment; (3): Redundant; (4) Cycle-free.
- Lagrangian relax (1) or (2), plus (4): semi-assignment.
- Lagrangian relax (3) plus (4): assignment.
- Lagrangian relax (1), and (2) except for node s: directed 1-tree relaxation.