Course Basics

- A new 4 credit unit course
- Part of *Theoretical Computer Science* courses at the *Department of Mathematics*
- There will be 4 hours of lectures per week
- Weekly exercises start next week
- We will assume familiarity with
  - Necessary mathematics
  - Basic programming
Organization & Timetable

- **Lectures**: Prof. Tapio Elomaa
  - Tue & Thu 12–14 in TB219 & TB223
  - Mar. 17 – May 12, 2015
  - Easter break: Apr. 2–8

- **Exercises**: M.Sc. Juho Lauri
  Mon 10–12 TC315

- **Exam**: Mon. May 25, 2015

Course Grading

- **Exam**: Maximum of 30 points
- **Weekly exercises** yield extra points
  - 40% of questions answered: 1 point
  - 80% answered: 6 points
  - In between: linear scale (so that decimals are possible)
Material

- The textbook of the course is
- There is no prepared material, the slides appear in the web as the lectures proceed
- The exam is based on the lectures (i.e., not on the slides only)

Content (Plan)

1. An introduction to approximation algorithms
2. Greedy algorithms and local search
3. Rounding data and dynamic programming
4. Deterministic rounding of linear programs
5. Random sampling and randomized rounding of linear programs
6. Randomized rounding of semidefinite programs
7. The primal-dual method
8. Cuts and metrics
1. Introduction

The whats and whys
The set cover problem
A deterministic rounding algorithm
Rounding a dual solution
The primal-dual method
A greedy algorithm
A randomized rounding algorithm

2. Greedy algorithms and local search

Scheduling jobs with deadlines
The $k$-center problem
Scheduling jobs on parallel machines
The traveling salesman problem
Maximizing float in bank accounts
Minimum-degree spanning trees
Edge coloring
3. Rounding data and dynamic programming

The knapsack problem
Scheduling jobs on identical parallel machines
The bin-packing problem

4. Deterministic rounding of linear programs

Minimizing the sum of completion times
Minimizing the weighted sum of completion times
Solving large linear programs in polynomial time via the ellipsoid method
The prize-collecting Steiner tree problem
The uncapacitated facility location problem
The bin-packing problem
5. Random sampling and randomized rounding of linear programs

- Simple algorithms for MAX SAT and MAX CUT
- Derandomization
- Flipping biased coins
- Randomized rounding
- Choosing the better of two solutions
- Non-linear randomized rounding
- The prize-collecting Steiner tree problem
- The uncapacitated facility location problem
- Scheduling a single machine with release dates
- Chernoff bounds
- Integer multicommodity flows
- Random sampling and coloring dense 3-colorable graphs

6. Randomized rounding of semidefinite programs

- A brief introduction to semidefinite programming
- Finding large cuts
- Approximating quadratic programs
- Finding a correlation clustering
- Coloring 3-colorable graphs
7. The primal-dual method

The set cover problem: a review
Choosing variables to increase
Cleaning up the primal solution
Increasing multiple variables at once
Strengthening inequalities
The uncapacitated facility location problem
Lagrangian relaxation and the $k$-median problem

8. Cuts and metrics

The multiway cut problem and a minimum-cut-based algorithm
The multiway cut problem and an LP rounding algorithm
The multicut problem
Balanced cuts
Probabilistic approximation of metrics by tree metrics
An application of tree metrics: Buy-at-bulk network design
Spreading metrics, tree metrics, and linear arrangement
1.1 The whats and whys

- Many interesting discrete optimization problems are NP-hard
- If P≠NP, we can’t have algorithms that
  1) find optimal solutions
  2) in polynomial time
  3) for any instance
- At least one of these requirements must be relaxed in any approach to dealing with an NP-hard optimization problem

**Definition 1.1:** An $\alpha$-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of $\alpha$ of the value of an optimal solution.

- We call $\alpha$ the performance guarantee of the algorithm
- It is also often called the **approximation ratio** or **approximation factor** of the algorithm
- We will follow the convention that $\alpha > 1$ for minimization problems, while $\alpha < 1$ for maximization problems
Definition 1.2: A polynomial-time approximation scheme (PTAS) is a family of algorithms \( \{ A_\varepsilon \} \), where there is an algorithm for each \( \varepsilon > 0 \), such that \( A_\varepsilon \) is a \((1 + \varepsilon)\)-approximation algorithm (for minimization problems) or a \((1 - \varepsilon)\)-approximation algorithm (for maximization problems).

- E.g., the knapsack problem and the Euclidean traveling salesman problem, both have a PTAS
- A class of problems that is not so easy is MAX SNP
- It contains problems, such as the maximum satisfiability problem and the maximum cut problem

Theorem 1.3: For any MAX SNP-hard problem, there does not exist a PTAS, unless \( P = NP \).

- E.g., maximum clique problem is very hard:
  - **Input:** an undirected graph \( G = (V, E) \)
  - **Goal:** find a maximum-size clique; i.e., find \( S \subseteq V \) that maximizes \( |S| \) so that for each pair \( i, j \in S \), it must be the case that \( (i, j) \in E \)
- Almost any nontrivial approximation guarantee is most likely unattainable:

Theorem 1.4: Let \( n \) be the number of vertices in an input graph, and consider any constant \( \varepsilon > 0 \). Then there does not exist an \( O(n^{\varepsilon - 1}) \)-approximation algorithm for the max clique problem, unless \( P = NP \)
• Observe that it is very easy to get an \( n^{-1} \)-approximation algorithm for the problem:
  – just output a single vertex
• This gives a clique of size 1, whereas the size of the largest clique can be at most \( n \), the number of vertices in the input
• The theorem states that finding something only slightly better than this completely trivial approximation algorithm implies that \( P = NP \)

1.2 Linear programs: The set cover problem

• Given a ground set of elements \( E = \{e_1, \ldots, e_n\} \), some subsets of those elements \( S_1, \ldots, S_m \) where each \( S_j \subseteq E \), and a nonnegative weight \( w_j \geq 0 \) for each subset \( S_j \)
• Find a minimum-weight collection of subsets that covers all of \( E \); i.e., we wish to find an \( I \subseteq \{1, \ldots, m\} \) that minimizes \( \sum_{j \in I} w_j \) subject to \( \bigcup_{j \in I} S_j = E \)
• If \( w_j = 1 \) for each \( j \), the problem is unweighted
The SC was used in the development of an antivirus product, which detects computer viruses.

It was desired to find salient features that occur in viruses designed for the boot sector of a computer, such that the features do not occur in typical computer applications.

These features were then incorporated into a neural network for detecting these boot sector viruses.

The elements of the SC were the known boot sector viruses (about 150 at the time).

Each set corresponded to some three-byte sequence occurring in these viruses but not in typical computer programs; there were about 21,000 such sequences.

Each set contained all the viruses that had the corresponding sequence somewhere in it.

The goal was to find a small number of such sequences (≤150) that would be useful for the neural network.

By using an approximation algorithm to solve the problem, a small set of sequences was found, and the neural network was able to detect many previously unanalyzed boot sector viruses.
The set cover problem also generalizes the vertex cover problem. In the vertex cover problem, we are given an undirected graph \( G = (V, E) \) and a nonnegative weight \( w_i \geq 0 \) for each vertex \( i \in V \). The goal is to find a minimum-weight subset of vertices \( C \subseteq V \) such that for each edge \( (i, j) \in E \), either \( i \in C \) or \( j \in C \). Again, if \( w_i = 1 \) for each vertex \( i \), the problem is unweighted.

To see that the vertex cover problem is a special case of the set cover problem:

- For any instance of the vertex cover problem, create an instance of the set cover problem in which
  - the ground set is the set of edges, and
  - a subset \( S_i \) of weight \( w_i \) is created for each vertex \( i \in V \) containing the edges incident to \( i \).
- It is not difficult to see that for any vertex cover \( C \), there is a set cover \( I = C \) of the same weight, and vice versa.
Linear programming plays a central role in the design and analysis of approximation algorithms.

Each linear program (LP) or integer program (IP) is formulated in terms of some number of decision variables that represent some sort of decision that needs to be made.

The variables are constrained by a number of linear inequalities and equalities called constraints.

Any assignment of real numbers to the variables such that all of the constraints are satisfied is called a feasible solution.
In the set cover problem, we need to decide which subsets $S_j$ to use in the solution.

- A decision variable $x_j$ to represent this choice.
- We would like $x_j$ to be 1 if the set $S_j$ is included in the solution, and 0 otherwise.
- Thus, we introduce constraints $x_j \leq 1$ and $x_j \geq 0$ for all subsets $S_j$.
- This does not guarantee that $x_j \in \{0,1\}$, so we formulate the problem as an IP to exclude fractional solutions.
- Requiring $x_j$ to be integer along with the constraints suffices to guarantee that $x_j \in \{0,1\}$.

We also want to make sure that any feasible solution corresponds to a set cover, so we introduce additional constraints.

- In order to ensure that every element $e_i$ is covered, it must be the case that at least one of the subsets $S_j$ containing $e_i$ is selected.
- This will be the case if
  \[ \sum_{j: e_i \in S_j} x_j \geq 1, \]
  for each $e_i$, $i = 1, \ldots, n$. 
In addition to the constraints, LPs and IPs are defined by a linear function of the decision variables called the **objective function**.

The LP or IP seeks to find a feasible solution that either maximizes or minimizes this objective function.

Such a solution is called an **optimal solution**.

The value of the objective function for a particular feasible solution is called the **value** of that solution.

The value of the objective function for an optimal solution is called the **value** of the LP or IP.

We say we **solve** the LP if we find an optimal solution.

In the case of the set cover problem, we want to find a set cover of minimum weight.

Given the decision variables $x_j$ and constraints described above, the weight of a set cover given the $x_j$ variables is

$$\sum_{j=1}^{m} w_j x_j$$

Thus, the objective function of the IP is $\sum_{j=1}^{m} w_j x_j$, and we wish to minimize this function.
IPs and LPs are usually written in a compact form stating first the objective function and then the constraints.

The problem of finding a minimum-weight set cover is equivalent to the following IP:

\[ \text{minimize } \sum_{j=1}^{m} w_j x_j \]

subject to
\[ \sum_{j : e_i \in S_j} x_j \geq 1, \quad i = 1, \ldots, n, \]
\[ x_j \in \{0, 1\}, \quad j = 1, \ldots, m \]

Let \( Z_{IP}^* \) denote the optimum value of this IP for a given instance of the set cover problem.

Since the IP exactly models the problem, we have that \( Z_{IP}^* = \text{OPT} \), where \( \text{OPT} \) is the value of an optimum solution to the set cover problem.

In general, IPs cannot be solved in polynomial time.

This is clear because the set cover problem is NP-hard, so solving the IP above for any set cover input in polynomial time would imply that \( P = NP \).
However, LPs are polynomial-time solvable
We cannot require the variables to be integers
Even in cases such as the set cover problem, we are still able to derive useful information from LPs
If we replace the constraints $x_j \in \{0,1\}$ with the constraints $x_j \geq 0$, we obtain the following LP, which can be solved in polynomial time

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{m} w_j x_j \\
\text{subject to} & \quad \sum_{i \in \mathcal{S}_j} x_j \geq 1, \quad i = 1, \ldots, n, \\
& \quad x_j \geq 0, \quad j = 1, \ldots, m
\end{align*}
\]

We could also add the constraints $x_j \leq 1$, for each $j = 1, \ldots, m$, but they would be redundant:
– in any optimal solution to the problem, we can reduce any $x_j > 1$ to $x_j = 1$ without affecting the feasibility of the solution and without increasing its cost

The LP is a relaxation of the original IP:
1. every feasible solution for the original IP is feasible for this LP; and
2. the value of any feasible solution for the IP has the same value in the LP
To see that the LP is a relaxation, note that any solution for the IP such that
\[ x_j \in \{0,1\} \text{ for each } j = 1, \ldots, m \] and
\[ \sum_{j:e_i \in S_j} x_j \geq 1 \text{ for each } i = 1, \ldots, m \]
will certainly satisfy all the constraints of the LP.

Furthermore, the objective functions of both the IP and LP are the same, so that any feasible solution for the IP has the same value for the LP.

Let \( Z_{LP}^* \) denote the optimum value of this LP.

Any optimal solution to the IP is feasible for the LP and has value \( Z_{LP}^* \).

Thus, any opt. solution to the LP will have value \( Z_{LP}^* \leq Z_{IP}^* = \text{OPT} \), since this minimization LP finds a feasible solution of lowest possible value.

Using a poly-time solvable relaxation in order to obtain a lower bound (in min) or an upper bound (in max) on the optimum value of the problem is a common concept.

We will see that a fractional solution to the LP can be rounded to a solution to the IP of objective function value that is within a certain factor \( f \) of the value of the LP \( Z_{LP}^* \).

Thus, the integer solution will cost \( \leq f \cdot \text{OPT} \).
1.3 A deterministic rounding algorithm

- Suppose that we solve the LP relaxation of the set cover problem
- Let \( x^* \) denote an optimal solution to the LP
- **How then can we recover a solution to the set cover problem?**
- Given the LP solution \( x^* \), we include subset \( S_j \) in our solution iff \( x_j^* \geq 1/f \), where \( f \) is the maximum number of sets in which any element appears.

More formally, let \( f_i = \left| \{j : e_i \in S_j\} \right| \) be the number of sets in which element \( e_i \) appears, \( i = 1, \ldots, n \); then \( f = \max_{i=1,\ldots,n} f_i \).

- Let \( I \) denote the indices \( j \) of the subsets in this solution
- In effect, we round the fractional solution \( x^* \) to an integer solution \( \hat{x} \) by setting \( \hat{x}_j = 1 \) if \( x_j^* \geq 1/f \), and \( \hat{x}_j = 0 \) otherwise
- It is straightforward to prove that \( \hat{x} \) is a feasible solution to the IP, and \( I \) indeed indexes a set cover.
**Lemma 1.5:** The collection of subsets $S_j, j \in I$, is a set cover.

*Proof:* Consider the solution specified by the lemma. An element $e_i$ is covered if this solution contains some subset containing $e_i$. Show that each element $e_i$ is covered. Because the optimal solution $x^*$ is a feasible solution to the LP, we know that $\sum_{j: e_i \in S_j} x_j^* \geq 1$ for element $e_i$. By the definition of $f_i$ and $f$, there are $f_i \leq f$ terms in the sum, so at least one term must be $\geq 1/f$. Thus, for some $j$ such that $e_i \in S_j, x_j^* \geq 1/f$. Therefore, $j \in I$, and element $e_i$ is covered. 

**Theorem 1.6:** The rounding algorithm is an $f$-approx. algorithm for the set cover problem.

*Proof:* It is clear that the algorithm runs in polynomial time. By our construction, $1 \leq f \cdot x_j^*$ for each $j \in I$. From this, and the fact that each term $f w_j x_j^*$ is nonnegative for $j = 1, \ldots, m$, we see that

$$\sum_{j \in I} w_j \leq \sum_{j=1}^m w_j \cdot (f \cdot x_j^*) = f \sum_{j=1}^m w_j \cdot x_j^* = f \cdot Z_{LP} \leq f \cdot \text{OPT}$$

where the final inequality follows from the fact that $Z_{LP}^* \leq \text{OPT}$. 

In the vertex cover, $f_i = 2$ for each $i \in V$, since each edge is incident to exactly two vertices.

Thus, the rounding algorithm gives a 2-approx. algorithm for the vertex cover problem.

This particular algorithm allows us to have an a fortiori guarantee for each input.

While we know that for any input, the solution produced has cost at most a factor of $f$ more than the cost of an optimal solution, we can for any input compare the value of the solution we find with the value of the linear programming relaxation.

If the algorithm finds a set cover $I$, let

$$\alpha = \sum_{j \in I} w_j / Z_{LP}^*$$

From the proof above, we know that $\alpha \leq f$.

However, for any given input, it could be the case that $\alpha$ is significantly smaller than $f$; in this case we know that

$$\sum_{j \in I} w_j \alpha Z_{LP}^* \leq \alpha \text{OPT},$$

and the solution is within a factor of $\alpha$ of optimal.

The algorithm can easily compute $\alpha$, given that it computes $I$ and solves the LP relaxation.
1.4 Rounding a dual solution

• Suppose that each element $e_i$ is charged some nonnegative price $y_i \geq 0$ for its coverage by a set cover.

• Intuitively, it might be the case that some elements can be covered with low-weight subsets, while other elements might require high-weight subsets to cover them.
  – we would like to be able to capture this distinction by charging low prices to the former and high prices to the latter.

In order for the prices to be reasonable, it cannot be the case that the sum of the prices of elements in a subset $S_j$ is more than the weight of the set, since we are able to cover all of those elements by paying weight $w_j$.

• Thus, for each subset $S_j$ we have the following limit on the prices:

\[ \sum_{i: e_i \in S_j} y_i \leq w_j \]

• We can find the highest total price that the elements can be charged by the following LP:
maximize \( \sum_{i=1}^{n} y_i \)
subject to \( \sum_{i:e_i \in S_j} y_i \leq w_j, \ j = 1,...,m, \)
\( y_i \geq 0, \ i = 1,...,n \)

- This is the **dual** LP of the set cover LP relaxation
- If we derive a dual for a given LP, the given program is called the **primal** LP
- This dual has a variable \( y_i \) for each constraint of the primal LP (i.e., for \( \sum_{i:e_i \in S_j} x_j \geq 1 \)), and has a constraint for each variable \( x_j \) of the primal
- This is true of dual linear programs in general

- Dual LPs have a number of very interesting and useful properties
- E.g., let \( x \) be any feasible solution to the set cover LP relaxation, and let \( y \) be any feasible set of prices (any feasible solution to the dual LP)
- Then consider the value of the dual solution \( y \):

\[
\sum_{i=1}^{n} y_i \leq \sum_{i=1}^{n} y_i \sum_{j:e_i \in S_j} x_j ,
\]

- since for any \( e_i, \sum_{j:e_i \in S_j} x_j \geq 1 \) by the feasibility of \( x \)
Rewriting the RHS of this inequality, we have
\[ \sum_{i=1}^{n} y_i \sum_{j: e_i \in S_j} x_j = \sum_{j=1}^{m} x_j \sum_{i: e_i \in S_j} y_i \]

Finally, noticing that since \( y \) is a feasible solution to the dual LP, we know that \( \sum_{i: e_i \in S_j} y_i \leq w_j \) for any \( j \), so that
\[ \sum_{j=1}^{m} x_j \sum_{i: e_i \in S_j} y_i \leq \sum_{j=1}^{m} x_j w_j \]

So we have shown that
\[ \sum_{i=1}^{n} y_i \leq \sum_{j=1}^{m} x_j w_j \]
i.e., any feasible solution to the dual LP has a value \( \leq \) any feasible solution to the primal LP

In particular, any feasible solution to the dual LP has a value \( \leq \) the optimal solution to the primal LP, so for any feasible \( y \), \( \sum_{i=1}^{n} y_i \leq Z^*_{LP} \)

This is called the weak duality property of LPs

Since we previously argued that \( Z^*_{LP} \leq \text{OPT} \), we have that for any feasible \( y \), \( \sum_{i=1}^{n} y_i \leq \text{OPT} \)
• Additionally, there is a quite amazing **strong duality** property of LPs
  – as long as there exist feasible solutions to both the primal and dual LPs, their optimal values are equal

• Thus, if $x^*$ is an optimal solution to the set cover LP relaxation, and $y^*$ is an optimal solution to the dual LP, then

\[
\sum_{j=1}^{m} w_j x_j^* = \sum_{i=1}^{n} y_i^*
\]