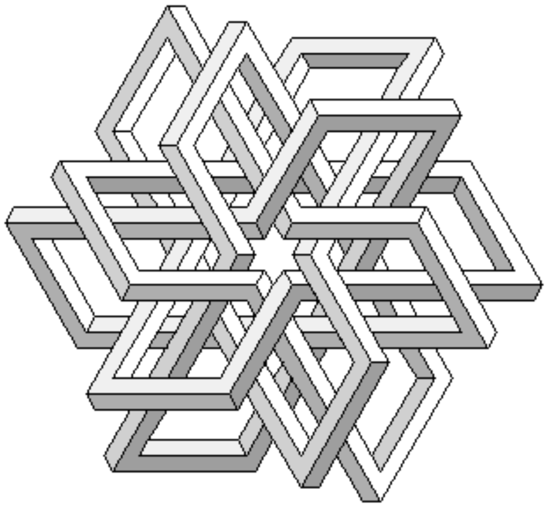


# Approximation algorithms

**IM, room 324**
**Thursday 11:40**

Kononov Alexander Veniaminovich

We will study

*NP*-hard optimization problem

# What you should know!

- Problem
- Instance
- Optimization problem
- Input size of an instance
- Algorithm
- Running time
- Polynomial time algorithm
- Linear programming (a linear program)
- NP-hard problem

# Some books in Combinatorial Optimization

- ***M. R. Garey*, *D. S. Johnson***, Computers and Intractability: A Guide to the Theory of NP-Completness, W. H. Freeman, 1979.
- ***C. H. Papadimitriou*, *K. Steiglitz***, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall INC, Englewood Cliffs, New Jersey, 1982.
- ***Korte B.*, *Vygen J.***, Combinatorial Optimization: theory and algorithms, (Algorithms and Combinatorics 21), Springer, Berlin, 2010.

# Problem

A **problem** will be a general question to be answered, usually possessing several **parameters**, or free variables, whose values are left unspecified.

A **problem $\Pi$** is described by giving:

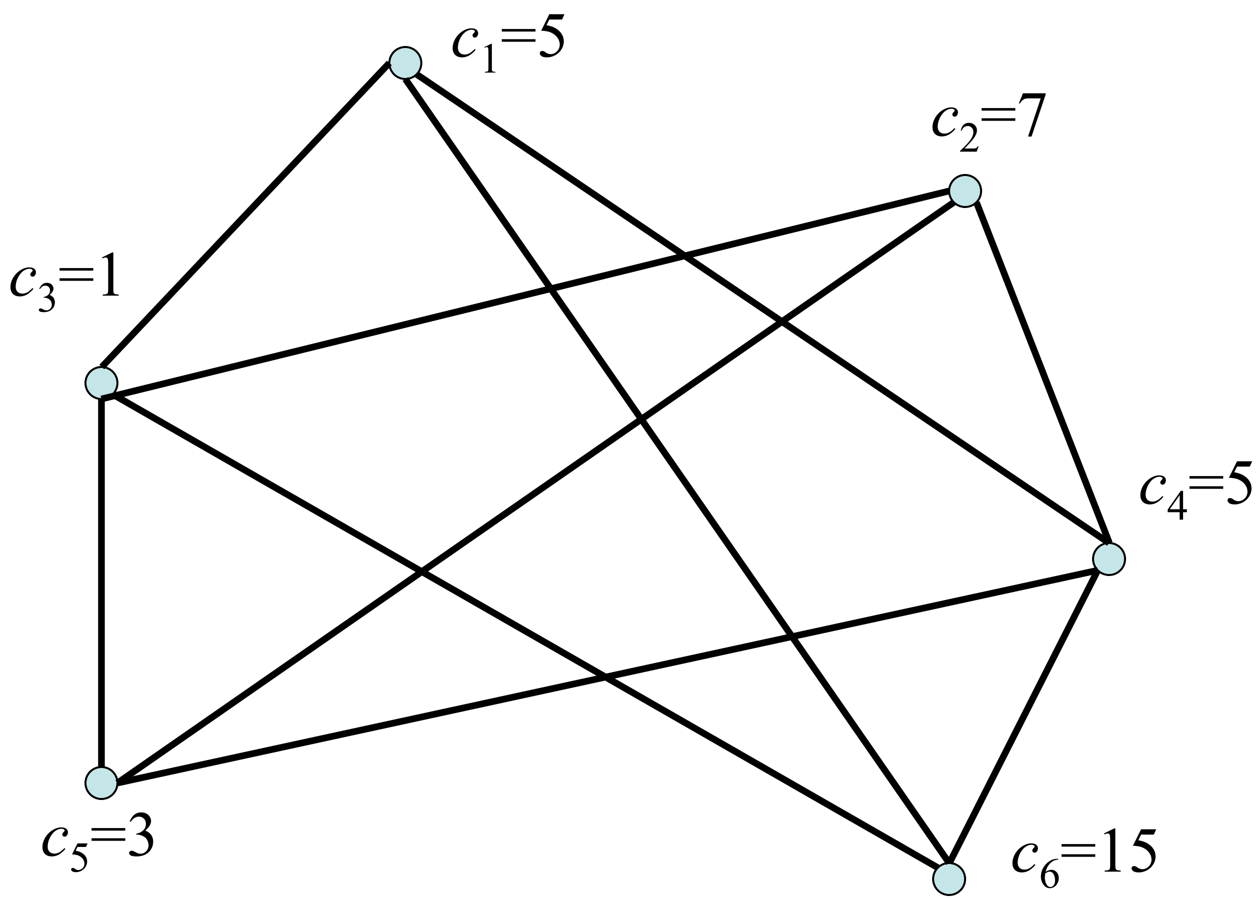
- a general description of all its parameters,
- a statement of what properties the answer, or *solution*, is required to satisfy.

An **instance *I*** of a problem is obtained by specifying particular values for all the problem parameters.

# Vertex cover

- *Given* an undirected graph $G = (V, E)$, and a cost function on vertices $c$: $V \to \mathbf{Q}^+$.
- *Find* a minimum cost vertex cover.
- **Vertex cover** is a set $V' \subseteq V$ such that every edge has at least one endpoint incident at $V'$.

# An instance of Vertex cover



# Input size

The input to an algorithm usually consists of a list of numbers. If all these numbers are integers, we can code them in binary representation, using $O(\log(|a|+2))$ bits for storing an integer $a$.

The **input size** of an instance with rational data is the total number of bits needed for the binary representation.

# Optimization problem

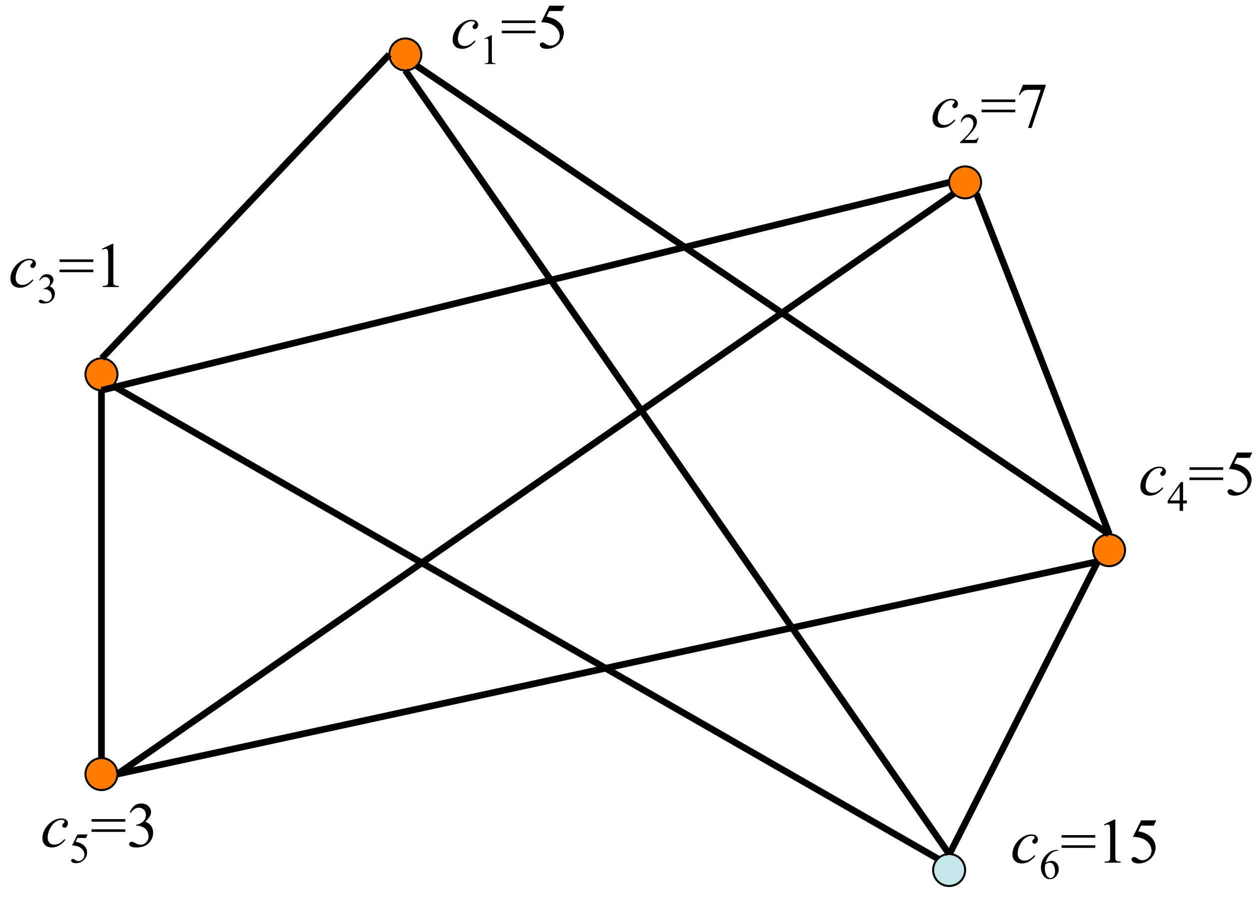
**An NP-optimization problem Π** is either a minimization or a maximization problem. It consists of:

- A set of valid instances, $\Omega_\Pi$, recognizable in polynomial time. We will assume that all numbers specified in an input are rationals.
- Each instance $I \in \Omega_\Pi$ has a set of feasible solutions $Sol_\Pi(I)$. We require that $Sol_\Pi(I) \neq \emptyset$, and that every solution $\sigma \in Sol_\Pi(I)$ is of length polynomially bounded in $|I|$. This means that there is a polynomial time algorithm that, given a pair $(I, \sigma)$, decides whether $\sigma \in Sol_\Pi(I)$.
- There is a polynomial time computable **objective function** $h_\Pi$, that assigns a nonnegative rational number to each pair $(I, \sigma)$. The objective function is frequently given a physical interpretation, such as *cost, length, weight*, etc.

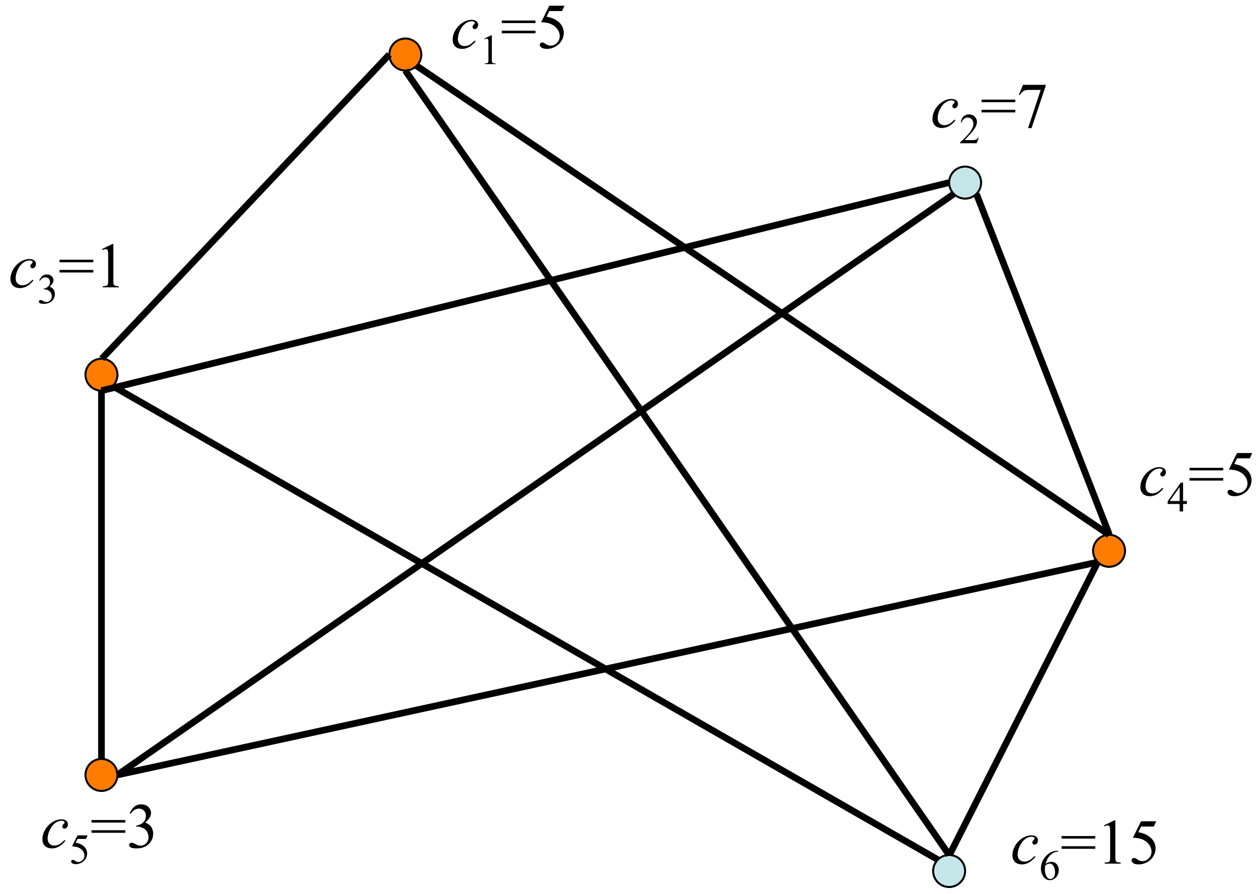
# Optimal solution

- An optimal solution for an instance $I \in \Omega_\Pi$ of minimization (maximization) problem is a feasible solution $\sigma^* \in Sol_\Pi$ that achieves the smallest (largest) objective function value, i.e. $h_\Pi(I, \sigma^*) \leq h_\Pi(I, \sigma)$ for all $\sigma \in Sol_\Pi\,(I)$.
- We will use $\mathrm{OPT}_\Pi(I)$ or $\mathrm{OPT}(I)$ to denote the objective function value of an optimal solution to instance $I$.

# A feasible solution



# An optimal solution



# Algorithm

An **algorithm** consists of

- a set of valid inputs,
- a  sequence of instructions each of which can be composed of elementary steps (variable assignments, conditional jumps (if – then – go to), and simple arithmetic operations like addition, subtraction, multiplication, division and comparison of numbers),
- For each valid input the computation of the algorithm is a uniquely defined finite series of elementary steps which produces a certain output.

# Running time

- The time requirements of an algorithm are conveniently expressed in terms of a single variable, the “size” of a problem instance, which is intended to reflect the amount of input data needed to describe the instance.
- The time complexity function for an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.

# Polynomial algorithm

- An algorithm with rational input is said to run in **polynomial time** if there is an integer $k$ such that it runs in $O(x^k)$ time, where $x$ is the input size, and all numbers in intermediate computations can be stored with $O(x^k)$ bits.
- An algorithm with arbitrary input is said to run in **strongly polynomial time** if there is an integer $k$ such that it runs in $O(n^k)$ time for any input consisting of $n$ numbers and it runs in polynomial time for rational input.
- In the case $k = 1$ we have a **linear-time algorithm**.

# NP-hard problem

- An optimization problem $\Pi$ is called **NP-hard** if all problems in NP polynomially reduce to $\Pi$.
- For any NP-hard problem, there does not exist an exact polynomial-time algorithm, unless **P = NP**.

**Almost all interesting optimization problems are NP-hard.**

# What we can do with NP-hard problems?

- Solve by enumeration algorithms.
- Solve by approximation algorithms:
  - heuristics, metaheuristics
  - approximation algorithms with guaranteed worst-case performance ratio.

We will study approximation algorithms with guaranteed approximation ratio.

# Approximation algorithm

An **ρ-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 $\rho$ of the value of an optimal solution.

# Approximation schemes

Let $\Pi$ be a minimization problem.

- An **approximation scheme** for problem $\Pi$ is a family of $(1+\varepsilon)$-approximation algorithms $A_\varepsilon$ for problem $\Pi$ over all $\varepsilon > 0$.
- A **polynomial-time approximation scheme (PTAS)** for problem $\Pi$ is an approximation scheme whose time complexity is polynomial in the input size for the fixed $\varepsilon$.
- A **fully polynomial-time approximation scheme (FPTAS)** for problem $\Pi$ is an approximation scheme whose time complexity is polynomial in the input size and also polynomial in $1/\varepsilon$.

# Algorithm

- How to design an approximation algorithm?
  - The study of the combinatorial structure of the problem
  - The study of properties of optimal solutions
  - The design of algorithms, based on these properties
- Generalization and extension of techniques accumulated in the construction of algorithms for polynomially solvable problems.

# Linear Programming

$$z(x) = c_1 x_1 + \ldots + c_n x_n \to \min$$

$$\begin{aligned} a_{11} x_1 + \ldots + a_{1n} x_n &\geq b_1 \\ &\vdots \\ a_{m1} x_1 + \ldots + a_{mn} x_n &\geq b_m \\ x_i \geq 0 \ \text{ for } \ i &= 1, \ldots, n. \end{aligned}$$

# Polynomially solvable problems

- The minimum spanning tree problems
- The maximum flow problem
- The assignment problem
- The maximum weight matching problem

●●●

# How do we establish the approximation guarantee?

- Can we compare the cost of the solution produced by the algorithm with the cost of an optimal solution?

# How do we establish the approximation guarantee?

- Can we compare the cost of the solution produced by the algorithm with the cost of an optimal solution?
- However, for such problems, not only is it NP-hard to find an optimal solution, but it is also NP-hard to compute the cost of an optimal solution.

# Lower bound

- We should find a “good” polynomial time computable lower bound on the cost of an optimal solution.
- Moreover, it is interesting that a “good” lower bound usually provides a key step in the design of approximation algorithms.

# Cardinality vertex cover

- *Given* an undirected graph $G = (V, E)$.
- *Find* a minimum cardinality vertex cover.

# Maximum and maximal matching

Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a **matching** if no two edges of $M$ share an endpoint.

- A matching of maximum cardinality in $G$ is called a **maximum matching**.
- A matching that is maximal under inclusion is called a **maximal matching**.

The size of a maximal matching in G provides a lower bound on the size of any vertex cover. This is so because any vertex cover has to pick at least one endpoint of each matched edge.

# Simple Algorithm

1. Find a maximal matching in $G$.
2. Output the set of matched vertices.

# Approximation ratio of the Simple Algorithm

**Theorem 1.1**

The Simple Algorithm is a factor 2 approximation algorithm for the cardinality vertex cover problem.

# Proof:

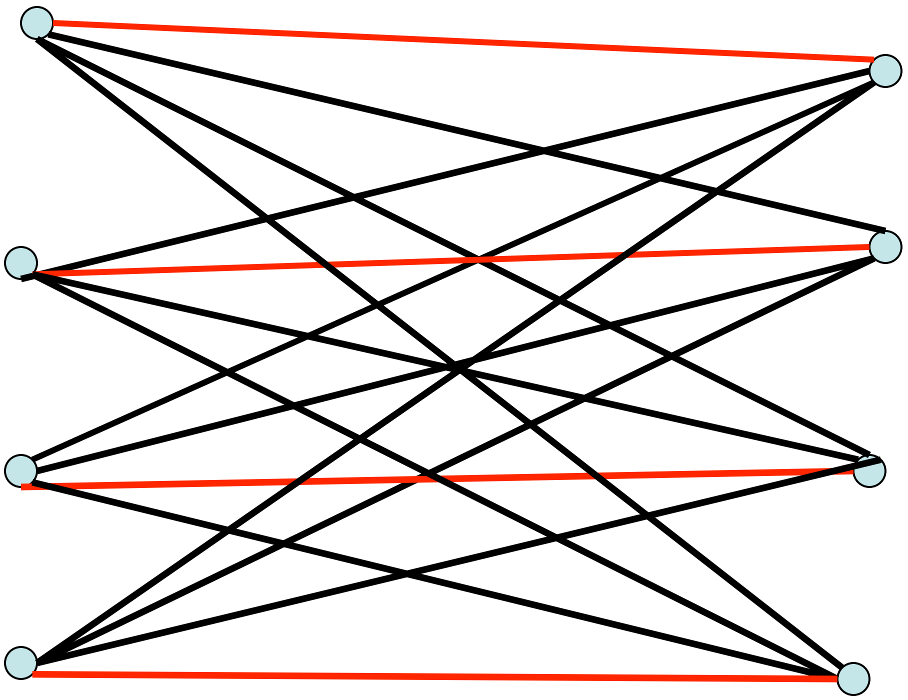
- No edge can be left uncovered by the set of vertices picked — otherwise such an edge could have been added to the matching, contradicting its maximality.
- Let $M$ be the matching picked. As argued above, $|M| \leq \mathrm{OPT}$.
- The approximation factor follows from the observation that the cover picked by the algorithm has cardinality $2\,|M|$.

# Can we improve the approximation guarantee?

- Can the approximation guarantee of the Simple Algorithm be improved by a better analysis?
- Can an approximation algorithm with a better guarantee be designed using the lower bounding scheme of the Simple Algorithm, i.e. size of a maximal matching in $G$?
- Is there some other lower bounding method that can lead to an improved approximation guarantee for vertex cover?

# Tight example

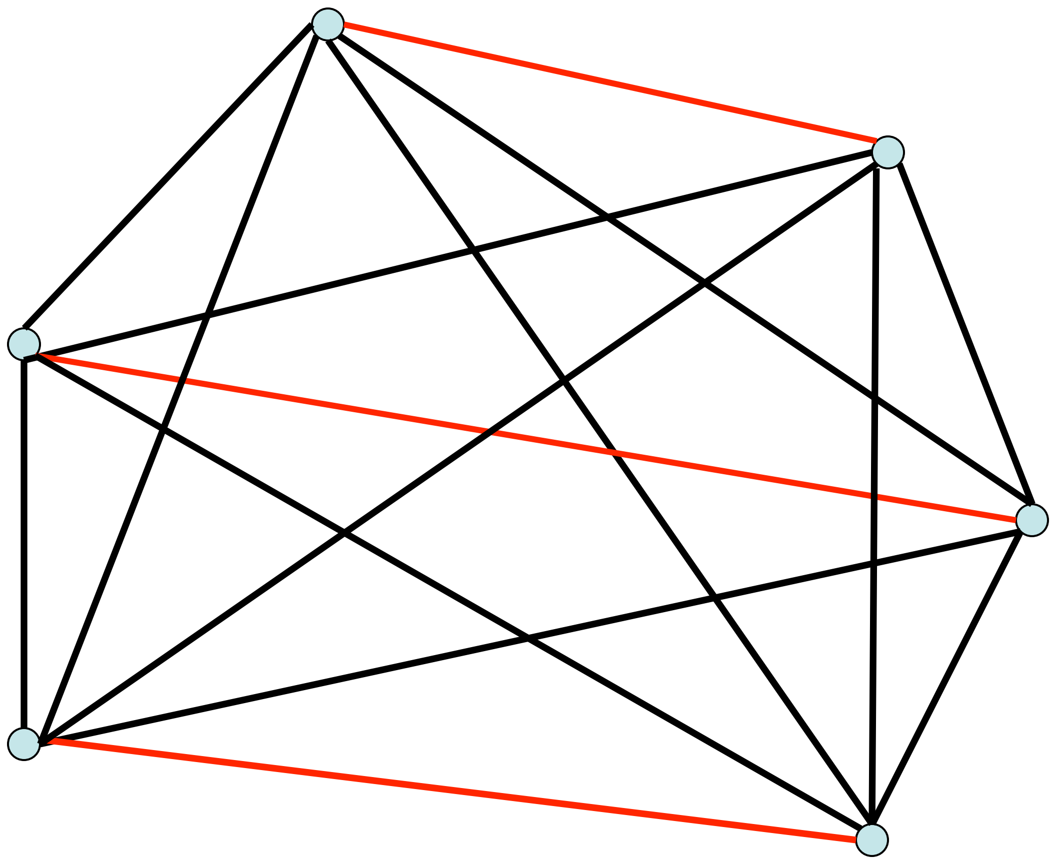
The analysis presented in Theorem 1.1 is tight.



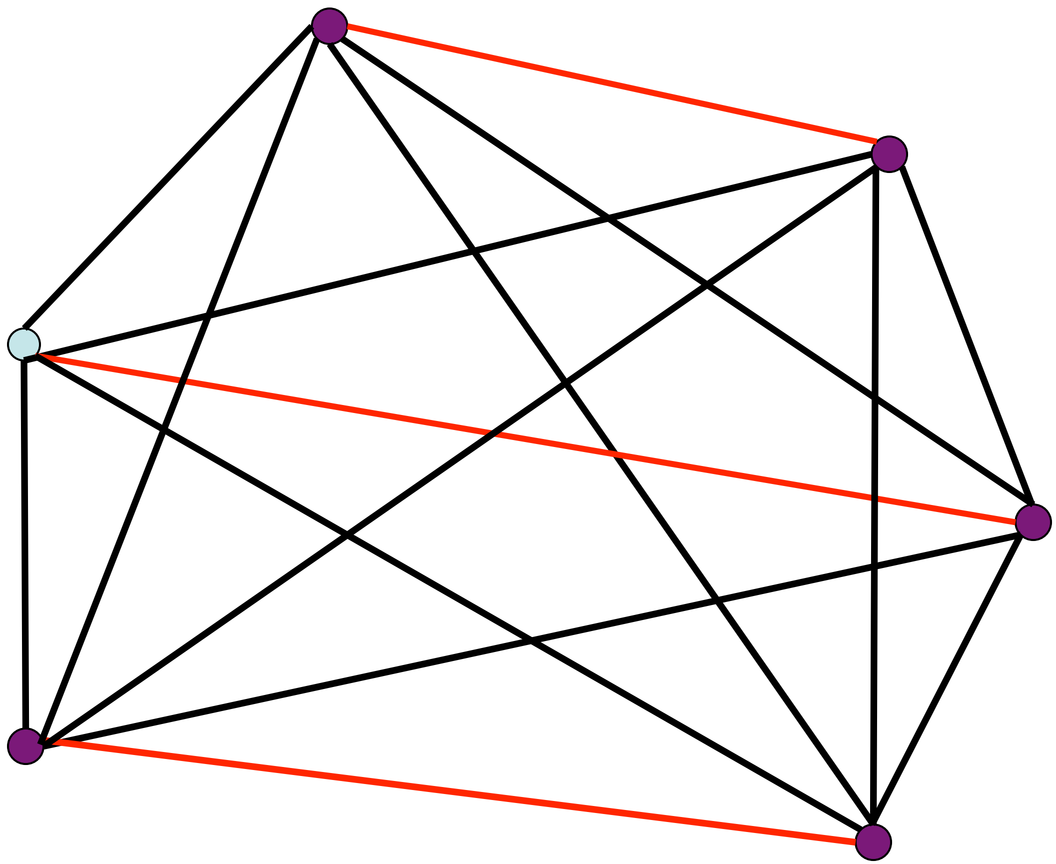
# Can we improve the approximation guarantee?

- Can the approximation guarantee of the Simple Algorithm be improved by a better analysis? NO!
- Can an approximation algorithm with a better guarantee be designed using the lower bounding scheme of the Simple Algorithm, i.e. size of a maximal matching in $G$?
- Is there some other lower bounding method that can lead to an improved approximation guarantee for vertex cover?

# Comparing the cost of the solution with the lower bound



# Comparing the cost of the solution with the lower bound



# Can we improve the approximation guarantee?

- Can the approximation guarantee of the Simple Algorithm be improved by a better analysis? NO!
- Can an approximation algorithm with a better guarantee be designed using the lower bounding scheme of the Simple Algorithm, i.e. size of a maximal matching in $G$? NO!
- Is there some other lower bounding method that can lead to an improved approximation guarantee for vertex cover?

# Books

- ***Кононов А.В., Кононова П.А.*** Приближенные алгоритмы для NP-трудных задач, Учебное пособие, НГУ, 2014.
- Approximation Algorithms for NP-hard problems, edited by ***D. Hochbaum***, PWS Publishing Company, 1997.
- ***V. Vazirani*** Approximation Algorithms, Springer-Verlag, Berlin, 2001.
- ***P. Schuurman, G. Woeginger*** Approximation Schemes – A Tutorial, chapter of the book “Lecture on Scheduling”, to appear in 2008.
- ***D. P. Williamson, D. B. Shmoys*** The Design of Approximation Algorithms, Cambridge University Press, 2011.

# Exercises

1. Consider the following problem.
   Problem MST:
   *Given* an undirected graph $G = (V, E)$, weights of edges
   $c: E \to \mathbf{Q}$ and positive rational number $B$.
   *Is there* a spanning tree of weight $B$ or less in $G$.
   Whether problem MST belongs to NP. Explain your answer.
2. Formulate the cardinality vertex cover problem as an integer problem.
3. Obtain the dual program for the LP-relaxation of the integer problem from exercise 2.