University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 01

Combinatorial and Approximation Algorithms

Exercise 1 (Edges in Minimum Spanning Trees)

Let G be a connected undirected graph on the vertex set V and the edge set E. Furthermore, we are given a cost function $c : E \to \mathbb{R}$ on the edges. Prove or disprove the following:

(a) Let e be an edge in G with minimal cost. Then there is a minimum spanning tree T which contains e.

(b) Let e be an edge in G with maximal cost. Then there is a minimum spanning tree T which does not contain e.

Exercise 2 (Cycle Elimination Algorithm)

Consider the following Minimum Spanning Tree algorithm suggested by one student:
Let G be a connected undirected graph on the vertex set V and the edge set E; let $c : E \to \mathbb{R}$ be a cost function on the edges. While G has cycles, choose an arbitrary cycle, say, C. Let e be an edge of C that has largest cost. Delete e from G.
Prove or disprove that this algorithm finds a minimum spanning tree.

Exercise 3 (Vertex Cover)

Given a simple undirected Graph $G = (V, E)$, a set $C \subseteq V$ is called a *vertex cover*, if each edge in E has at least one end-vertex in C. The problem Vertex Cover is: Find a vertex cover of G having minimal cardinality.

(a) Formulate Vertex Cover as a Set Cover problem.

(b) An edge set M is called *matching* if any two distinct edges of M do not have a common end-vertex. A matching M is called *maximal*, if there is no matching M' in G with $M \subset M'$.

Consider the following algorithm for Vertex Cover: Choose a maximal matching M of G and define C as the set of all end-vertices of the edges in M.

Show that the algorithm is a 2-approximation, i.e., show that (1) C is indeed a vertex cover and (2) $|C| \leq 2 \cdot |C^*|$, where C^* is a minimal (optimal) vertex cover.

Exercise 4 (Vertex Cover Challenge)

Visit the page `https://pacechallenge.org/2019/vc/index` and download the instances.
Visit Wolfram Mathematica `https://www.wolfram.com/mathematica/trial/` and download a trial version.
There will be a small demo during the exercise lesson.

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 02

Combinatorial and Approximation Algorithms

Exercise 1 (Multi-Source Multi-Sink)

The MAXIMUM FLOW problem can be generalized to multiple sources $s_1, \dots, s_p$ and sinks $t_1, \dots, t_q$. The value of a flow f is then defined as $\text{value}(f) = \sum_{1 \le i \le p} \text{bal}_f(s_i)$.
Show that the MULTI-SOURCE MULTI-SINK MAXIMUM FLOW problem can be reduced to the ordinary MAXIMUM FLOW problem.

Exercise 2 (Guest Shuffle)

Suppose you are organizing a dinner and lay n tables. You invite m families to join the dinner and family i has a_i members. Furthermore table j has b_j seats. In order to boost the inter-family-communication you want to make sure that no two members of the same family are at the same table (if this is possible). Formulate this seating arrangement problem as a MAXIMUM FLOW problem.

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 03

Combinatorial and Approximation Algorithms

Exercise 1 (Greedy Algorithm for Knapsack)

Give an instance which shows that the approximation guarantee of 1/2 for the GREEDY algorithm for KNAPSACK is tight.

Exercise 2 (Fractional Knapsack)

Let $c, w \in \mathbb{R}^n$ be non-negative vectors with $c_1/w_1 \geq c_2/w_2 \geq \cdots \geq c_n/w_n$. The FRACTIONAL KNAPSACK problem is the following mathematical program:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j, \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \leq W, \\ & 0 \leq x_j \leq 1 \quad j = 1, \ldots, n. \end{aligned}$$

Let $k = \min\left\{ j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i > W \right\}$. Show that an optimum solution for the FRACTIONAL KNAPSACK problem is given by the vector x with

$$\begin{aligned} x_j &= 1 && \text{for } j = 1, \ldots, k-1, \\ x_j &= \frac{W - \sum_{i=1}^{k-1} w_i}{w_k} && \text{for } j = k, \text{ and} \\ x_j &= 0 && \text{for } j = k+1, \ldots, n. \end{aligned}$$

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 04

Combinatorial and Approximation Algorithms

Exercise 1 (Bin Packing Lower Bounds)

Give examples that establish lower bounds for the approximation factor of

(a) 5/3 for First Fit

(b) 3/2 for First Fit Decreasing

for Bin Packing.

Exercise 2 (Next Fit with Bounded Sizes)

Let $0 < \gamma < 1$. Let $I = \{1, \ldots, n\}$ be an instance of Bin Packing with $s_i < \gamma$ for $i \in I$. Denote the number of bins used by Next Fit on instance I by $\textsc{nf}(I)$.
Show that

$$\textsc{nf}(I) \leq \left\lceil \frac{s(I)}{1-\gamma} \right\rceil \leq \left\lceil \frac{\textsc{opt}(I)}{1-\gamma} \right\rceil,$$

where $\textsc{opt}(I)$ denotes the optimal number of bins for the instance I.

Exercise 3 (Set Cover Greedy)

Give an instance showing that the Greedy algorithm for Set Cover is a H_n-approximation.

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 05

Combinatorial and Approximation Algorithms

Exercise 1 (Scheduling with Release Dates)

Consider a scheduling problem for m identical machines where jobs arrive over time: Any job j must not start before its release date r_j, and denote its processing time by p_j. The goal is to minimize the makespan.
Assume that we are given a c-approximation algorithm ALG for the problem when all $r_j = 0$. (For example, the LIST SCHEDULING algorithm from the lecture is a $(2-1/m)$-approximation.) Show that there is a $2 \cdot c$-approximation algorithm ALG′ for the problem with arbitrary release dates.
Hint. Let S_0 be the set of jobs released at time 0. Apply ALG to schedule S_0, finishing at time F_0. Let S_1 be the set of jobs released in time $(0, F_0]$. Again, apply ALG to schedule S_1, finishing at time F_1. Continue.

Exercise 2 (Preemptive Makespan Scheduling)

Consider the *preemptive* version of MAKESPAN SCHEDULING on identical machines. That is, we allow that the computation of a job to be partitioned into parts that can each run on any machine, but no two parts of the same job can run at the same time.
Give an algorithm that solves this problem optimally in polynomial time and determine its running time.

Exercise 3 (List Scheduling Revisited)

Show that the LIST SCHEDULING algorithm from the lecture for MAKESPAN SCHEDULING on identical machines is actually a $(2 - 1/m)$-approximation algorithm, where m is the number of machines.