University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 01

Combinatorial and Approximation Algorithms

Exercise 1 (Running Times)

Suppose you are given an algorithm, which receives just a single number n as its input in binary encoding. Assume that the algorithm executes always exactly n^2 many elementary operations on input n. Is this a polynomial time algorithm?

Exercise 2 (Spanning Trees)

Let G be a connected undirected graph on n vertices. Show that any spanning tree T of G has exactly $n-1$ edges.

Exercise 3 (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 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 4 (Maximum 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. A *maximum spanning tree* is a spanning tree in G having largest total edge cost (among all spanning trees).
Modify the algorithms Kruskal and Prim from the lecture such that maximum spanning trees are computed. Prove the correctness of your modification.

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 02

Combinatorial and Approximation Algorithms

Exercise 1 (Vertex Cover)

The problem Vertex Cover is defined as follows: Given a simple undirected Graph $G = (V, E)$ find a set $C \subseteq V$ with minimal cardinality, such that each edge in E has at least one endvertex in C.

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

(b) Give a 2-approximation for Vertex Cover (not using (a)).
Hint. Consider the edges of the graph. For some of the edges, include both endvertices.

Exercise 2 (Multi-Source Multi-Sink)

The Maximum Flow problem can be generalized to multiple sources $s_1, \ldots, s_p$ and sinks $t_1, \ldots, t_q$. The value of a flow f is then defined as $\text{value}(f) = \sum_{1 \leq i \leq p} \text{bal}_f(s_i)$. Show that the Multi-Source Multi-Sink Maximum Flow problem can be reduced to the ordinary Maximum Flow problem.

Exercise 3 (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

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

Exercise 3 (Fractional Multi-Knapsack)

In the MULTI-KNAPSACK problem, we are given m knapsacks, each having a capacity W_i for $i = 1, \ldots, m$, n items each having weight w_j for $j = 1, \ldots, n$, and costs c_{ij} when item i is packed into knapsack j. We may assume that $\sum_{i=1}^{m} W_i \geq \sum_{j=1}^{n} w_j$. The task is to pack *all* items into knapsacks such that all knapsack capacities are obeyed and the total cost is minimized.

In the FRACTIONAL MULTI-KNAPSACK problem, it is allowed to assign parts of the items to the knapsacks.

Give a combinatorial polynomial-time algorithm that solves this problem (without using linear programming).

Hint. Reduction to MINIMUM COST FLOW.

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 $\text{NF}(I)$.
Show that

$$\text{NF}(I) \leq \left\lceil \frac{s(I)}{1-\gamma} \right\rceil \leq \left\lceil \frac{\text{OPT}(I)}{1-\gamma} \right\rceil,$$

where $\text{OPT}(I)$ 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 (Maximum Coverage)

The Maximum Coverage problem is the following: Given a universe $U = \{u_1, \ldots, u_n\}$ of n elements, with non-negative weights $w : U \to \mathbb{R}^+$, a collection of subsets $\mathcal{S}$ of U, and an integer k, pick k sets so as to maximize the weight of covered elements.
Consider the following Greedy algorithm:

Step 1. Set $G_0 = \emptyset$.

Step 2. For $i = 1, \ldots, k$:

(a) Select S that maximizes $\sum_{u \in G_{i-1} \cup S} w(u)$.

(b) Set $G_i = G_{i-1} \cup S$.

Step 3. Return G_k.

Show that this algorithm achieves an approximation guarantee of $1 - (1 - 1/k)^k$. Notice that this is at least $1 - 1/e \approx 0.632$.
Hint. Firstly, show that the weight added in each iteration is at least a $1/k$ fraction of the weight difference to the optimal solution G^*, i.e., $\text{weight}(G_i) - \text{weight}(G_{i-1}) \geq 1/k \cdot (\text{weight}(G^*) - \text{weight}(G_{i-1}))$. Secondly, prove $\text{weight}(G_i) \geq (1 - (1 - 1/k)^i) \cdot \text{weight}(G^*)$.

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.

University of Zurich
Institute of Computer Science
Dr. Alexander Souza

Spring Term 2017
Sheet 06

Combinatorial and Approximation Algorithms

Exercise 1 (Lower Bound for Sorted List Scheduling)

Give a class of examples for the Sorted List Scheduling algorithm from the lecture for Makespan Scheduling on identical machines that shows a lower bound of 4/3.

Exercise 2 (Lower Bound for Unrelated Machines)

Show that the analysis of the 2-approximation algorithm from the lecture for Makespan Scheduling on unrelated machines is tight.

Exercise 3 (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.