Randomised Algorithms: Linear Programming and Approximation Algorithms

February 2023

1 LPs and Simplex

Exercise 1 Convert the following LP into slack form. Also state the set of basic and non-basic variables.

$$\begin{array}{lrcrcrcl}
\text{maximise} & 2x_1 & & & - & 6x_3 & & \\
\text{subject to} & & & & & & & \\
& x_1 & + & x_2 & - & x_3 & \leq & 7 \\
& 3x_1 & - & x_2 & & & \geq & 8 \\
& -x_1 & + & 2x_2 & + & 2x_3 & \geq & 0 \\
& & & & x_1, x_2, x_3 & & \geq & 0
\end{array}$$

[Source: CLRS: 29.1-5]

Exercise 2 Show that the following LP is infeasible:

$$\begin{array}{lrcrcl}
\text{maximise} & 3x_1 & - & 2x_2 & & \\
\text{subject to} & & & & & \\
& x_1 & + & x_2 & \leq & 2 \\
& -2x_1 & - & 2x_2 & \leq & -10 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

[Source: CLRS: 29.1-6]

Exercise 3 Show that the following LP is unbounded:

$$\begin{array}{lrcrcl}
\text{maximise} & x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& -2x_1 & + & x_2 & \leq & -1 \\
& -x_1 & - & 2x_2 & \leq & -2 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

[Source: CLRS: 29.1-7]

Exercise 4 Find a linear program which has more than one optimal solution.

Exercise 5 Solve the following linear program using Simplex:

$$\begin{array}{lrcrcl}
\text{maximise} & 5x_1 & - & 3x_2 & & \\
\text{subject to} & & & & & \\
& x_1 & - & x_2 & \leq & 1 \\
& 2x_1 & + & x_2 & \leq & 2 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

[Source: CLRS: 29.3-6]

Exercise 6 Solve the following linear program using SIMPLEX:

$$\begin{array}{lcccl} \text{maximise} & x_1 & + & 3x_2 & \\ \text{subject to} & & & & \\ & x_1 & - & x_2 & \leq 1 \\ & 2x_1 & + & x_2 & \leq 2 \\ & & x_1, x_2 & & \geq 0 \end{array}$$

[Source: CLRS: 29.3-6]

Exercise 7 Attempt **[2015P7Q2]**(a)-(b).

Exercise 8 Attempt **[2018P27Q1]**(a)-(b).

Exercise 9 [Duality] Attempt **[2018P27Q1]**(c).

Exercise 10 Attempt **[2019P8Q1]**.

Exercise 11 Attempt **[2020P8Q1]**(a).

2 Formulating problems as Linear Programs

Exercise 12 [Shortest path] Consider the linear program for the shortest path problem from s to t.

1. What happens if there is a negative-weight cycle?
2. Prove that, if there are no negative-weight cycles, the optimal solution $\overline{d}_t$ of the linear program equals the correct distance d_t.
3. How would your formulate the single-source shortest path problem as a linear program?

Exercise 13 [Multi-commodity flow] In the *multi-commodity flow* problem, there is a graph $G = (V, E)$ where edges have capacities $c(u, v)$. there are k commodities $K_1, \ldots, K_k$, where $K_i = (s_i, t_i, d_i)$, meaning that there is a demand of d_i from source s_i to destination d_i. Formulate this as a linear program (LP).

Exercise 14 [Minimum Spanning Tree]

(a) Formulate the (undirected) Minimum Spanning Tree problem as an integer program (possibly using an exponential number of constraints).

Hint: Think about subtour eliminations.

(b) Formulate the linear relaxation of the integer program.

(c) Show that given a fractional solution to the linear relaxation, there is a polynomial time algorithm to convert the fractional solution into an integer one with the same cost. What does this show?

(*Answer*)

(a) Let $x_{uv} \in \{0, 1\}$ indicate whether the edge (u, v) is in the output MST. Then the cost of the MST is $\sum_{uv \in E} x_{uv} w_{uv}$, We need to add constraints so that the indicators do not form a cycle. This means that every set of vertices $S \subseteq V$ must have at most $|S| - 1$ internal edges:

$$\sum_{(u,v) \in E(S,S)} x_{uv} \leq |S| - 1.$$

Hence, putting it all together, we have that:

$$\begin{array}{lll} \text{maximise} & \sum_{uv \in E} x_{uv} w_{uv} & \\ \text{subject to} & & \\ & x_{uv} \in \{0,1\} & \forall uv \in E \\ & \sum_{uv \in E} x_{uv} = |V| - 1 & \\ & \sum_{(u,v) \in E(S,S)} x_{uv} \leq |S| - 1 & \forall \emptyset \subset S \subseteq V \end{array}$$

(b) If all values x_{uv} are integer then it means that the induced solution is indeed a spanning tree and it indeed has the minimum value.

Otherwise, a cycle is formed and this cycle must have one edge edge e with $x_e < 1$. By picking the heaviest edge e' on the cycle and moving as much of $x_{e'}$ to x_e making one of the two either 0 or 1, we obtain a solution with one integer solution value and the same (or smaller) cost.

Note: If the edge weights are unique, then the MST is unique and we can argue about integrality since there can be no cycle.

Exercise 15 [Min-Cost Bipartite Matching] In the *min-cost bipartite matching* problem, we are given a bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y|$ and a weight function $w : X \times Y \to \mathbb{R}$ and our goal is to find the bipartite matching with the minimum cost.

(a) Formulate this as an integer program.

(b) Formulate the linear relaxation of the integer program.

(c) (+) Give a polynomial-time algorithm for converting a fractional solution of the linear relaxation to a solution to the original problem.

Hint: Given a fractional solution, modify the solution so that the number of integral values (0 or 1) in the solution increases, and the objective remains the same.

Exercise 16 [Bounded Degree MST] Formulate as an integer program the bounded degree MST problem, where we want to find the MST where each vertex has at most d neighbours.

Exercise 17 Attempt **[2017P7Q1]**(a)-(b).

Exercise 18 Attempt **[2020P8Q1]**(c).

Exercise 19 Prove that the LP formulation of the Set-Cover problem (Lecture 10, slide 4) is feasible if and only if the Set-Cover instance has a feasible solution.

3 Convex sets

Extended Note 1 [Convex set] Recall a set S is **convex** if for every $x, y \in S$, $\lambda x + (1 - \lambda) y \in S$ for all $\lambda \in [0, 1]$.

Exercise 20

(a) Show that the intersection of two convex sets C_1 and C_2 are convex.

(b) Show that the union of two convex sets need not be convex.

Exercise 21

(a) Prove that for any $a_1, \ldots, a_n, b \in \mathbb{R}$, we have that the set

$$\mathcal{A} := \{x \in \mathbb{R}^n : a_1 x_1 + \ldots + a_n x_n \leq b\}$$

is convex.

(b) Prove that the set of feasible solutions of a linear program forms a convex set.

4 (non-randomised) approximation algorithms

Exercise 22 [Vertex Cover]

(a) Analyse the greedy algorithm for the unweighted Vertex cover problem that achieves an approximation ratio of 2 (Slide 14 of Lecture 9).

(b) **Bonus-Question:** What is the problem behind the "more natural" greedy approach where instead of both endpoints of an uncovered edge, we only include one of the two endpoints into our cover?

Exercise 23 [Minimum Cardinality Maximal Matching] Design a factor 2 approximation algorithm for the problem of finding a minimum cardinality maximal matching in an undirected graph. A matching is *maximal* if no other edge can be added to the matching.
Hint: Use the fact that any maximal matching is at least half the maximum matching.

Exercise 24 [Maximum acyclic graph] Given a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic. Design an approximation algorithm for this problem.

(*Answer*) Consider the following algorithm:

1. Arbitrarily number of the vertices of the graph.
2. Let F be the set of edges (u, v) with $u < v$ and B the set of edges (u, v) with $v < u$.
3. Return the largest of the two sets B and F.

It is clear that both edge sets induce an acyclic graph. The two sets satisfy $|B| + |F| = |E|$ and so one of the two must have size at least $|E|/2$. Since the optimal answer may have at most $|E|$ edges, this proves an approximation ratio of 2.

Exercise 25 [Next-Fit for Bin-Packing] Consider the Next-Fit heuristic for the bin-packing problem. Show that it is a 2-approximation algorithm.

Exercise 26 Attempt **[2017P9Q1]**.

Exercise 27 [Metric TSP] Consider TSP problem on a graph $G = (V, E)$ with cost function $c : V \times V \to \mathbb{R}$, which satisfies $c(u, v) + c(v, w) \geq c(u, w)$ for all $u, v, w \in V$.
Consider the algorithm that picks an arbitrary root r and then finds an MST from that root, and as a solution returns a walk of the MST (e.g. the pre-order traversal). Show that this algorithm achieves a 2-approximation for this version of the TSP.

(*Answer*) See these old slides.

Exercise 28 Attempt **[2015P7Q2]**(c). *Hint: First look at Exercise 27.*

Exercise 29 [Christophides' algorithm] (+) Read about Christophides' algorithm for the TSP problem from these slides. Show that it achieves a 3/2-approximation ratio.

Exercise 30 Attempt **[2018P9Q1]**.

Exercise 31 Consider an instance of the unweighted SET-COVER problem with the condition that no element $x \in X$ appears in more than k many subsets. Design an approximation algorithm based on deterministic rounding which achieves an approximation ratio of at most $\mathcal{O}(k)$.

5 Randomised approximation algorithms

Exercise 32 Consider a MAX-SAT formula where each clause has at least 4 literals. Design a randomised approximation algorithm and analyse its approximation ratio.

Exercise 33 [MAX-4-CNF] Attempt **[2016P9Q1]**(b).

Exercise 34 Consider the randomised approximation algorithm for the weighted SET-COVER problem. Translate the algorithm from the course into one based on non-linear randomised rounding such that, given the LP solution y, we directly round this LP solution to get a cover C which (i) covers all elements with probability $1 - 1/n$, and (ii) has an expected cost which is at most $\mathcal{O}(\log n)$ times the cost of the optimal cover.
Hint: By non-linear we refer to the way of choosing the probability of setting a variable to 1. The randomised rounding rules in the lecture is linear in the sense that the probability is equal to the fractional value of the LP solution.

Exercise 35 Attempt **[2017P7Q1]**(c).

Exercise 36 Attempt **[2020P9Q1]**.

Exercise 37 Recall the randomised algorithm for SET-COVER presented in the lecture. As input, we have a SET-COVER instance with n elements; and let us additionally assume we have at most $\mathrm{poly}(n)$ many subsets (and also that we can cover all n elements, i.e., $\bigcup_{s \in \mathcal{F}} S = X$). The algorithm achieves with probability $1/3$ that the returned cover is correct and the cost of the cover is at most a factor of $4 \ln(n)$ away from the optimal cost (see Lecture 10, slide 9.2). Turn this into a randomised algorithm such that:

1. The algorithm terminates in a time that is polynomial in the input size, with probability 1 (i.e., always).
2. The algorithm returns a correct solution, with probability 1 (i.e., always).
3. The expected approximation ratio is $\mathcal{O}(\log n)$.