ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Rasmus Kyng, Angelika Steger, David Steurer

Algorithms, Probability, and Computing | Special Assignment 2 | HS24

- Write your solutions using a computer, where we strongly recommend to use LaTeX. We do not grade hand-written solutions.
- The solution is due on December 3rd, 2024 by 14:00 pm. Please submit one file per exercise on Moodle.
- For geometric drawings that can easily be integrated into LaTeX documents, we recommend the drawing editor IPE, retrievable at `http://ipe.otfried.org` or through various package managers.
- Write short, simple, and precise sentences.
- This is a theory course, which means: if an exercise does not explicitly say "you do not need to prove your answer" or "justify intuitively", then a formal proof is always required. You can of course refer in your solutions to the lecture notes and to the exercises, if a result you need has already been proved there.
- We would like to stress that the ETH Disciplinary Code applies to this special assignment as it constitutes part of your final grade. The only exception we make to the Code is that we encourage you to verbally discuss the tasks with your colleagues. However, you need to include a list of all of your collaborators in each of your submissions. It is strictly prohibited to share any (hand)written or electronic (partial) solutions with any of your colleagues. We are obligated to inform the Rector of any violations of the Code.
- There will be two special assignments this semester. Both of them will be graded and the average grade will contribute 20% to your final grade.
- As with all exercises, the material of the special assignments is relevant for the (midterm and final) exams.

Exercise 1

20 **points**

(*Page Ranking and Farkas lemma*)

Consider the following process: a user is browsing a website that contains n pages numbered from 1 to n and when she is on page i, she randomly clicks on a link that brings her to page j with probability p_{ij}.
Suppose that when the user enters the website for the first time, she lands on page i with probability x_i. After her first click (on one of the links), she will end up on page j with probability

$$y_j = \sum_{i=1}^{n} x_i p_{ij}.$$

Assume that $x_i \geq 0$ for all $i = 1, \ldots, n$, $\sum_{i=1}^{n} x_i = 1$, $p_{ij} \geq 0$ for all $i, j = 1, \ldots, n$, and $\sum_j p_{ij} = 1$, for all $i = 1, \ldots, n$.

(a) Prove that the probabilities y_i are well-defined, i.e. show that $y_j \geq 0$, $\forall j \in [n]$ and $\sum_j y_j = 1$.

Let $\mathrm{x} = (x_1, x_2, \ldots, x_n)^\top$ and $\mathrm{y} = (y_1, y_2, \ldots, y_n)^\top$, then we can write

$$\mathrm{y} = P^\top \mathrm{x}, \quad \text{where } P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

We say that x is a *steady state* if

$$\mathrm{x} = P^\top \mathrm{x}, \quad \text{where } P,$$

$x_i \geq 0, \forall i$, and $\sum_i x_i = 1$.

The original Page Rank algorithm used by Google was actually ranking the web pages based on the score given by the steady state. In the remaining part of the exercise, we want to show that such a state always exists using Farkas lemma.

In order to do so, consider a process like the one described above.

(b) Write an LP that has a solution if and only if the steady state exists.

(c) Suppose that the LP is impossible, apply Farkas lemma to the LP and arrive at a contradiction.
Hint: Prove that the vector guaranteed by Farkas lemma does not exist.

Exercise 2

30 **points**

(*NOT-OR Circuits Using Linear Programming*)

A digital circuit consists of a network of wires connected through logic gates. Each wire carries a signal, which can be either 0 (representing *false*) or 1 (representing *true*). In this exercise, we consider a class of circuits called *NOT-OR circuits*, which consist of three sequential layers.

- Input Layer: The circuit has n input wires, where each wire i (for $i = 1, 2, \ldots, n$) carries either a 1 or a 0. The sequence of values on these input wires is called the *input configuration*.
- Negation Layer: The n input wires are passed through a layer that outputs $2n$ wires. For each input x_i, this layer produces both x_i and its negation, denoted as $\text{not}(x_i)$. These values are referred to as *intermediate wires*.
- OR Layer: The intermediate wires are then fed into a layer that outputs m wires, each representing a *OR-gate* value $C_j \in \{0, 1\}$ computed from a subset of the intermediate wires (for $j = 1, 2, \ldots, m$). Each OR-gate value is 1 if at least one of its intermediate wires has value 1. Multiple OR-gates may receive the same intermediate wire as input.

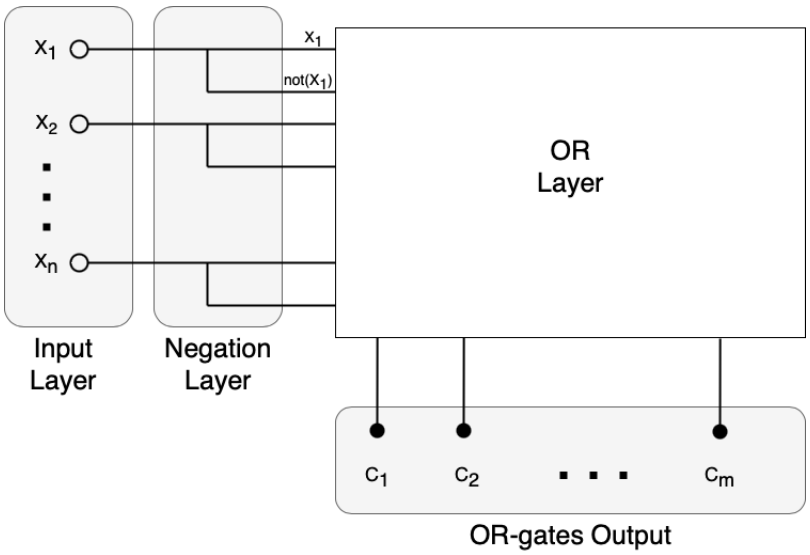

Figure 1: A scheme of a NOT-OR circuit

In the first part of the exercise, we consider circuits where each OR-gate takes as input exactly 5 intermediate wires. It can be shown that it is NP-hard to determine whether there exists an input configuration such that all OR-gates output the value 1. Instead, we aim to find an input configuration that maximizes the number of OR-gates with output value 1. Let OPT represent the maximum number of OR-gates that can have value 1 for any input configuration.

(a) Assign the value 1 or 0 to each input wire independently with probability $\frac{1}{2}$. Prove that, in expectation, this assignment results in at least $\frac{31}{32}$ OPT OR-gates having value 1.

In the second part of the exercise, we consider a NOT-OR circuit where each OR-gate may take as input an arbitrary number of intermediate wires. Each output OR-gate C_j controls a switch that regulates an output current $f_j \geq 0$. The *total output* of the circuit is defined as $\sum_{j=1}^{m} C_j f_j$. Given a circuit and outputs $f_1, \ldots, f_m$, our goal is to find an input configuration that maximizes this total output.

(b) Design a polynomial-time deterministic algorithm that returns an input configuration achieving at least half of the maximum possible output current.
Hint: Consider an input configuration and its negation.

(c) Show that the optimal solution to the following linear program provides an upper bound on the maximum possible output current for the circuit.

$$\max \sum_{j=1}^{m} z_j f_j$$

subject to:

$$\sum_{i:x_i \in C_j} y_i + \sum_{i:(\text{not}(x_i)) \in C_j} (1 - y_i) \geq z_j, \quad \forall j = 1, 2, \ldots, m$$

$$0 \leq y_i \leq 1, \quad 0 \leq z_j \leq 1$$

(d) Let y^*, z^* be an optimal solution to the LP in (c). Construct an input configuration by assigning each input wire i the value 1 with probability y_i^* and the value 0 with probability $1 - y_i^*$. Prove that, in expectation, the total output current of this solution is at least $\left(1 - \frac{1}{e}\right)$ times the optimal output.
Hint (AM-GM Inequality): For non-negative values $x_1, x_2, \ldots, x_\ell$, the inequality

$$\left(\prod_{i=1}^{\ell} x_i\right)^{1/\ell} \leq \frac{\sum_{i=1}^{\ell} x_i}{\ell}$$

holds.
Hint: The function $f(x) = 1 - \left(1 - \frac{x}{\ell}\right)^{\ell}$ is concave for $x \in [0, 1]$ and $\ell \in \mathbb{N}^+$.

Exercise 3

10 **points**

(*Pizza slicing*)
Suppose that we have a rectangular pizza of size $n \times m$. The pizza can be seen as a collection of nm squares, half topped with mushrooms and half topped with peppers. We want to slice the pizza in slices of size 2×1 so that each slice contains a square topped with mushrooms and one with peppers. Devise a polynomial time algorithm that given the topping for each square of the pizza finds the number of ways in which it is possible to slice the pizza into slices that satisfy the above conditions.

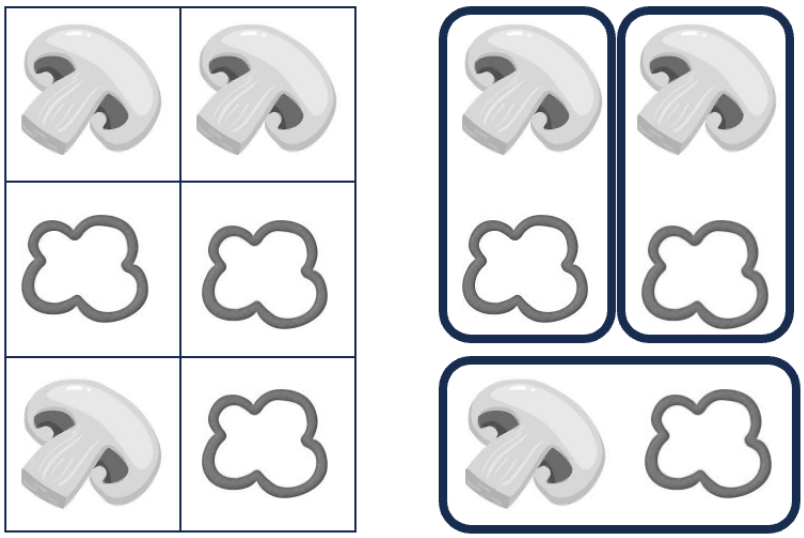

Figure 2: A 3×2 pizza and slicing of the pizza into three slices each containing one mushroom square and one pepper square.

Exercise 4

15 points

(*Shortest cycle in a graph*)
This exercise aims to develop a fast algorithm that finds the weight of the cycle with the minimum weight in a weighted directed graph. Consider a simple ~~un~~directed graph $G = (V, E)$ with a weight function $w : E \mapsto \{1, 2, \ldots, W\}$. The weight of a cycle is defined as the sum of the weights of the edges in the cycle. Let $n = |V|$ and assume that $W = O(n^{100})$ and that the cycle with minimum weight in G is unique. Our algorithm should run in $O(WM(n)n^{o(1)})$ time[1]. To do this, introduce a variable x_{ij} for each edge $(i, j) \in E$ and another extra variable y and define an $n \times n$ matrix C where

$$C_{ij} = \begin{cases} x_{ij}y^{w((i,j))}, & \text{if } (i,j) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let

$$p(x, y) := \det(C + I) - 1,$$

where I is the identity matrix and

$$q(y) := p(\mathbf{1}, y)$$

be the polynomial in y obtained by setting $x_{ij} = 1$ for $i, j = 1, 2, \ldots n$. Finally, let $d_{\min}$ and $c_{\min}$ be the degree and the coefficient of the monomial of smallest degree that appears in q.

(a) Prove that $d_{\min}$ equals the weight of the shortest cycle in G.

(b) Devise an algorithm that finds the weight of the shortest cycle in G with probability at least $1 - 1/n$. The algorithm should run in $O(WM(n)\log^{50}(n))$ time.
Hint: Use Theorem 1 stated below.

Theorem 1. *Let $A(y)$ be an $n \times n$ matrix where the entries $A_{i,j}$ are polynomials over y with coefficients in a finite field F and have degree at most d. Then we can compute* $\det(A(y))$ *in* $O(dM(n)(\log(n) + \log(d))^{50})$ *operations.*

[1] $M(n)$ is the time required to multiply two $n \times n$ matrices.