If you are interested in taking this course and your appeal has not yet been approved, but you want access to Canvas, email me your login/SIS ID (starting with “e”). I will add you to Canvas as a guest student.

CS5275 – The Algorithm Designer’s Toolkit (S2 AY2025/26)

Lecture 0: course logistics

Discord link: https://discord.gg/Uj6E8Rh5qD

Teaching mode

- **Time/date:** Friday 18:30–21:30, Seminar Room 1 (COM1-0206).
  - There will be two 10-minute breaks.
  - We will aim to finish the lecture at least 10–15 minutes before 21:30.

Teaching mode

- **Tutorial sessions:**
  - During Weeks 4–13, we will hold a tutorial session in the final hour of the lecture (20:30–21:30).
  - In these sessions, we will discuss exam and homework problems, as well as additional practice problems prepared specifically for the tutorials.
  - Students will be invited to present their solutions, and any reasonable attempt at presenting will receive **bonus points**.

Detail will be given later.

Course website

- Most of the important information (e.g., tentative schedule) about this course can be found in this Google Sheet.

https://docs.google.com/spreadsheets/d/1SXME5AxwDCXhZ1rNcfihZTZZycQaa0O9v2y1AjG37Fs

Instructor

- CHANG Yi-Jun
  - Office: COM3-02-24
  - Email: cyijun@nus.edu.sg
  - Office hour: by appointment, online or in person.

- Feel free to let me know by **email** or **Discord** if you have any questions, suggestions, comments, or anything to discuss.

Teaching assistant

- Ivan Adrian Koswara:
  - Email: ivanak@comp.nus.edu.sg

- **Responsibilities:**
  - Grading all 4 take-home problem sets.
  - Grading the midterm exam.
  - Conducting 5 tutorial sessions:
    - Discussing the questions from the take-home problem sets and the midterm exam.

Course description

- This course introduces students to a diverse variety of algorithmic and mathematical techniques that are commonly encountered in computer science.

Highly useful and relevant for research in algorithms and theory, but less useful for solving LeetCode problems.

Course description

- This course introduces students to a diverse variety of algorithmic and mathematical techniques that are commonly encountered in computer science.

Highly useful and relevant for research in algorithms and theory, but less useful for solving LeetCode problems.

- Part 1: Expanders
- Part 2: Analysis of Boolean functions
- Part 3: Multiplicative weights update

Note: I might not be able to finish all of them.

Part 1: Expanders

- Informally:
  - An expander graph is a **sparse** graph with **strong connectivity**.

Think of it as a sparse version of cliques.

- Key property:
  - Any subset of vertices that is not too large has a **large boundary**.

(many edges leaving the set)

- Why this matters:
  - Combines <u>efficiency</u> with <u>robustness</u>.

Sparsity → **low cost** (few edges)

Part 1: Expanders

- Direct practical Application in **network design**:
  - Connections (edges) are expensive to build.
  - Still want the network to be highly connected.

- Benefits in networks:
  - **Fault tolerance**
    - Resilient to vertex or edge failures.
    - No small group of vertices can be easily isolated
  - **Efficient communication**
    - Short paths between any pair of vertices.
    - Efficient routing.

Part 1: Expanders

- Many further applications of expander graphs:
  - Reducing the number of random bits needed in randomized algorithms.
  - Designing error-correcting codes.
  - Designing graph algorithms (for both general graphs and expander graphs).

(Might be) covered in this course

- There are also other more abstract uses in areas like complexity theory (e.g., circuit lower bounds).

Not covered in this course

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - Linear programming (relaxation, rounding, and duality).
  - Metric embeddings.
  - Random walks.
  - Spectral graph theory.

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - **Linear programming (relaxation, rounding, and duality).**
  - Metric embeddings.
  - Random walks.
  - Spectral graph theory.

- Formulate a relaxed version of the problem as a linear program.
- Solve the linear program in polynomial time.
- Convert the fractional solution into an integral one.

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - Linear programming (relaxation, rounding, and duality).
  - **Metric embeddings.**
  - Random walks.
  - Spectral graph theory.

- Computing distances in some metrics is more expensive than in others.
- This motivates the following task: Map points from a more complex metric space into a simpler one while incurring only small distortion in distances.

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - Linear programming (relaxation, rounding, and duality).
  - Metric embeddings.
  - **Random walks.**
  - Spectral graph theory.

- In an expander graph, a short random walk quickly converges to the stationary distribution
- This yields applications for many tasks, including reducing random bits and sampling.

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - Linear programming (relaxation, rounding, and duality).
  - Metric embeddings.
  - Random walks.
  - **Spectral graph theory.**

- Study graphs via eigenvalues and eigenvectors of associated matrices:
  - e.g., Adjacency matrix and Laplacian.
- Key Insight:
  - Algebraic structure reveals deep combinatorial information in graphs.
  - Linear algebra becomes a tool for graph algorithms.

Part 1: Expanders

- In the study of expanders, we will learn various broadly useful algorithm design tools:
  - Linear programming (relaxation, rounding, and duality).
  - Metric embeddings.
  - Random walks.
  - Spectral graph theory.

- **Big Picture**
  - Expanders act as a unifying theme where optimization, geometry, probability, and linear algebra come together to design and analyze powerful algorithms.

Part 2: Analysis of Boolean functions

- Boolean functions arise in many areas of computer science and beyond:

In **circuit design**, a Boolean function may represent the desired behavior of a circuit with n inputs and one output.

In **graph theory**, one can identify n-vertex graphs G with length-$\binom{n}{2}$ strings indicating which edges are present, so a Boolean function may represent a property of such graphs.

Part 2: Analysis of Boolean functions

- Boolean functions arise in many areas of computer science and beyond:

In **learning theory**, a Boolean function may represent a concept with n binary attributes.

In **social choice theory**, a Boolean function may represent a voting rule for an election with two candidates named 0 and 1.

Part 2: Analysis of Boolean functions

- We will explore how **Fourier analysis** can be applied to Boolean functions and how it leads to several interesting applications:
  - BLR linearity test (sublinear algorithms)
  - Arrow's impossibility theorem (social choice theory)
  - Goldreich–Levin algorithm (cryptography and learning theory)

Part 2: Analysis of Boolean functions

- We will explore how **Fourier analysis** can be applied to Boolean functions and how it leads to several interesting applications:
  - BLR linearity test (sublinear algorithms)

    A foundational result in property testing and sublinear-time algorithms:
    Testing whether a function is linear using only a few queries.

  - Arrow's impossibility theorem (social choice theory)

  - Goldreich–Levin algorithm (cryptography and learning theory)

Part 2: Analysis of Boolean functions

- We will explore how **Fourier analysis** can be applied to Boolean functions and how it leads to several interesting applications:
  - BLR linearity test (sublinear algorithms)

    A foundational result in property testing and sublinear-time algorithms:
    Testing whether a function is linear using only a few queries.

  - Arrow's impossibility theorem (social choice theory)

    It is impossible to design a ranked voting system with three candidates
    that simultaneously satisfies a set of seemingly fair conditions.

  - Goldreich–Levin algorithm (cryptography and learning theory)

Part 2: Analysis of Boolean functions

- We will explore how **Fourier analysis** can be applied to Boolean functions and how it leads to several interesting applications:
  - BLR linearity test (sublinear algorithms)

    A foundational result in property testing and sublinear-time algorithms:
    Testing whether a function is linear using only a few queries.

  - Arrow's impossibility theorem (social choice theory)

    It is impossible to design a ranked voting system with three candidates
    that simultaneously satisfies a set of seemingly fair conditions.

  - Goldreich–Levin algorithm (cryptography and learning theory)

    A core tool in learning Boolean functions and cryptographic constructions:
    Efficiently finding all significant Fourier coefficients.

Part 2: Analysis of Boolean functions

- **Big Picture:**
  - Boolean functions arise in many areas of computer science and beyond.
  - Fourier coefficients capture useful information in Boolean functions.
  - Fourier analysis becomes a useful tool for the design and analysis of algorithms.

Part 3: Multiplicative weights update

- Consider the problem of prediction from expert advice:
  - There are n experts.
  - Over T rounds, each expert gives advice or a prediction.
  - At each round, we choose an expert to follow.

- **Goal:** Perform nearly as well as the best expert in hindsight.

Part 3: Multiplicative weights update

- Consider the problem of prediction from expert advice:
  - There are n experts.
  - Over T rounds, each expert gives advice or a prediction.
  - At each round, we choose an expert to follow.

Weather Forecasting
- Experts: different forecasting models or meteorologists
- Each day: predict rain or sunshine

Stock Market & Portfolio Management
- Experts: trading strategies or financial analysts
- Each round: recommend buy/sell/hold decisions

Part 3: Multiplicative weights update

- Consider the problem of prediction from expert advice:
  - There are n experts.
  - Over T rounds, each expert gives advice or a prediction.
  - At each round, we choose an expert to follow.

Medical Diagnosis
- Experts: diagnostic tests, doctors, or AI models
- Each case: recommend a diagnosis or treatment

Sports Betting / Match Prediction
- Experts: prediction models, commentators, or statistical systems
- Each game: predict winner or score

Part 3: Multiplicative weights update

- Consider the problem of prediction from expert advice:
  - There are n experts.
  - Over T rounds, each expert gives advice or a prediction.
  - At each round, we choose an expert to follow.

- **Multiplicative weights update:**
  - Assign identical initial weights to the experts.
  - Update these weights multiplicatively and iteratively according to the feedback of how well an expert performed.
  - Select the expert advice to follow according to the weights.

Part 3: Multiplicative weights update

- We will study the multiplicative weights update method and its use in designing approximation algorithms for several problems:
  - Linear classifiers (machine learning)
  - Computing Nash equilibria of zero-sum games (game theory)
  - Solving packing/covering linear programs (optimization)

Summary

- The target audience of the course:

Those who want to do algorithms and theory research.

This course provides a bridge between undergraduate-level algorithms and research-level algorithms.

Those who have a genuine interest in algorithms and theory.

You will see a lot of beautiful and amazing ideas in algorithms and theory.

Textbooks / lecture notes

- Part 1: Expanders
  - Lectures 1.1–6.1 of https://sites.google.com/site/thsaranurak/teaching/Expander
  - Chapters 1–5, 9–11, and 22 of https://lucatrevisan.github.io/books/expanders-2016.pdf
- Part 2: Analysis of Boolean functions
  - Chapters 1–3 of https://arxiv.org/pdf/2105.10386
- Part 3: Multiplicative weights update
  - Sections 1–3.3 of https://theoryofcomputing.org/articles/v008a006/v008a006.pdf

Be aware that there might be errors…

Prerequisites

- Undergraduate-level understanding in:
  - Algorithms.
  - Probability theory.
  - Linear algebra.

- **Mathematical maturity**.
  - Being able to read and write formal proofs.

Everything in this course is proof-based.

Prerequisites

- **Mathematical maturity**.
  - Being able to read and write formal proofs.

If you are not good at reading and writing proofs, then it is likely that you will not get a good grade.

Assessment components

- Four take-home problem sets (35%)
  - Considering only the best 3 out of 4.
  - Tutorial participation bonus points.
- Midterm exam (25%)
- Final exam (40%)

Assessment components

- Four take-home problem sets (35%)
  - Considering only the best 3 out of 4. ← No deadline extension is allowed, regardless of the reason.
  - Tutorial participation bonus points. ← Detail will be given later.
- Midterm exam (25%) Week 7
- Final exam (40%) Examination week

All guest students are welcome to submit solutions to the problem sets and to take the midterm exam; however, they are not permitted to take the final exam.

Problem sets

- There will be **four** take-home problem sets.

- The solution for each problem will be graded on a $0/\frac{1}{2}/1$ scale:

  - 0: an incorrect answer or no answer.

  - $\frac{1}{2}$: a partially correct answer or a potentially correct answer with a bad presentation.

  - 1: a correct or a nearly correct answer.

Problem sets

We will try to follow this guideline for most of the problems.
However, there may be instances where we choose to deviate from it (without prior notice).

- There will be **four** take-home problem sets.

- The solution for each problem will be graded on a $0/\frac{1}{2}/1$ scale:

  - 0: an incorrect answer or no answer.

  - $\frac{1}{2}$: a partially correct answer or a potentially correct answer with a bad presentation.

  - 1: a correct or a nearly correct answer.

Problem sets

- You are allowed to:
  - Discuss the problem sets with other people.
  - Search for an answer or a hint on the internet.

- You must write your solution by yourself.

Not allowed: Write your solution by a simple copy-and-paste from an existing solution, even with some paraphrasing.

Allowed: You ask your friend for hints to solve the problem, and then you prepare your solution independently, without looking at your friend's solution.

- You are **strongly recommended** to typeset your solution in Latex.

https://www.overleaf.com/

Problem sets

NUS has a very strict plagiarism policy.

Sharing your solution with other people is also considered plagiarism.

According to the policy, we must report any plagiarism case, regardless of how small it is.

Consequences:
- Failing the course.
- An internal record in NUS.
- …

- You are allowed to:
  - Discuss the problem sets with other people or.
  - Search for an answer or a hint on the internet.

- You must write your solution by yourself.

Not allowed: Write your solution by a simple copy-and-paste from an existing solution, even with some paraphrasing.

Allowed: You ask your friend for hints to solve the problem, and then you prepare your solution independently, without looking at your friend's solution.

- You are **strongly recommended** to typeset your solution in Latex.

https://www.overleaf.com/

AI policy

- AI tools may be used to **polish writing and presentation**.

- **Not allowed**:
  - Asking AI for solutions to take-home problem sets.

  - You are allowed to ask AI for solutions to other problems.

  - However, keep in mind:
    - AI-generated solutions may be incorrect or incomplete.

Exams

- For both midterm and final exams:
  - F2F - Hardcopy (pen and paper)
  - Open Book (Only hardcopy materials are allowed)
  - Calculators are not allowed.
  - The exams consist of multiple-choice, true/false, and essay questions (mostly proof-based).

Exams

- For both midterm and final exams:
  - F2F - Hardcopy (pen and paper)
  - Open Book (Only hardcopy materials are allowed)
  - Calculators are not allowed.
  - The exams consist of multiple-choice, true/false, and essay questions (mostly proof-based).

- **Midterm exam:**
  - 18:30–20:15 (105 minutes), March 6 (Friday)
  - In our classroom COM1-0206.

- **Final exam:**
  - 17:00–19:00 (120 minutes), April 30 (Thursday)