COMP3821/9801: Extended Algorithm Design and Analysis

Course Description

2024, Term 1

Welcome to COMP3821/9801! By the end of the course, we hope that this course gives you a deeper appreciation for the theory of algorithms, and hope that you continue your exploration into this rich field in your future endeavours. This course aims to build on top of the knowledge acquired in COMP2521 or COMP9024. Indeed, part of the design of an efficient algorithm is to use the appropriate data structure. This document will aim to answer all of your questions about COMP3821/9801.

Contents

1 What is this course all about?

Theoretical Computer Science is a branch of computer science that concerns itself with the theoretical underpinnings of computer science itself. In a way, we can broadly categorise theoretical computer science into two main areas: *algorithm theory* and *hardness theory*. This course focuses on *algorithm theory*.

Algorithm theory primarily focuses on describing methods for which we can show that a given problem is *easy* to solve. For example, if I gave you a map of Sydney and two suburbs, how easy is it to find the shortest path to drive to reach suburb A from suburb B? What if I don't want to drive through a toll road? To show that this problem is easy, I need to give you the solution to any input (input being the map and two suburbs) you give me *quickly*.[1] We do need to be a bit careful with what we mean by *quickly*, but the moral of the story is: I should not need to wait *too* long; otherwise, I may as well just try every possible path and that's not particularly insightful.

Towards the end, we will see that some problems are, indeed, fundamentally harder than others. Without stronger assumptions, we cannot conclude whether such problems are actually hard, or if we are too stupid to show that these problems are easy. For this to be useful, we will develop a framework to show that certain problems are actually hard under the assumption that other problems have been deemed hard. This will bring us full circle – if there is one message to take away, it's this: *it is easy for us to show that certain problems are easy, but it is hard for us to show that other problems are hard.*

1.1 Design vs Analysis

A large portion of our discussions will lie in two main components: the *design* phase and the *analysis* phase. Brute-force solutions *are* solutions: they are correct by design, but they do not give us important insights about the problems at hand. What we would like to do is to introduce methods that are *provably* faster than the brute-force methods.

Designing an algorithm means to unambiguously describe the steps that an algorithm takes to return the output. Given a particular input, what does the algorithm actually do and what is the output of the algorithm when it has finished executing? As such, it should be clear and succinct. For example, if I want an algorithm to return the *number* of items, I should not expect my algorithm to return the actual list.

Analysing an algorithm is to take the algorithm and do two things:

1. *Is the algorithm correct?* Your first step is to convince the reader that your algorithm is, indeed, correct. Does it need to be correct on *every* instance? Usually yes, we will see something slightly different when introducing approximation algorithms. If you don't argue why your algorithm is correct, then it may as well be *incorrect*. The onus is on you to *show* the reader that your algorithm does give the correct (or *a* correct) answer to the problem, and we will see ways to show that your algorithm is correct.
2. *Does the algorithm run efficiently?* Your second step is to analyse the running time of the algorithm. The running time of an algorithm is measured by the number of *primitive operations* it takes for an algorithm to finish executing. Of course, what we classify as primitive operation depends on the underlying computation model. For our purposes, we will assume that all arithmetic operations are constant-time operations. Additionally, any lookup to an array is also constant-time.

1.2 Pseudocode?

Yeah... this is a bit of contentious matter. Some people work better given pseudocode since they work better with code structures, and this is completely okay. To put simply, pseudocode is anything that is not *actual* code but is designed in a way that resembles code. You have for loops, while loops, if-else statements in pseudocode. However, it is not meant to be compilable and is designed to give you a high-level view of a feasible solution without needing to deal with fiddly implementation issues.

One major gripe with using pseudocode is in its style. Pseudocode, while good in practice, does not give you the opportunity to express your ideas succinctly and concisely. Part of our assessment is in your ability to effectively communicate your ideas and this is quite difficult to do with code. In these situations, you

[1] "Quickly" does not necessarily mean *fast* in practice; it just means that the runtime does not blow up exponentially.

are better off communicating your ideas in full sentences. It becomes clearer to read in this way. We aren't saying pseudocode is bad in general, and you should use it in your day-to-day practices. But for the purposes of communication, we suggest sticking with plain English.

2 Before the course...

We generally try to teach all of the material from the ground up, however, the trimester system is not on our side. As a result, we need to assume a few things about your knowledge from previous courses you must do before enrolling into COMP3821/9801.

2.1 Mathematical Prerequisites

Okay... contrary to popular belief, there isn't a whole lot of mathematics that you really need for the course. Yes, you will be doing proofs. Yes, you will need to read mathematics language. No, you will not need linear algebra. No, you will not need calculus (well okay... maybe just *basic* differentiation). No, you will not need statistics.

Here's a basic checklist to get you up to speed with what you ought to know for COMP3821/9801.

- Know how to read set theory – let $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ be two sets. Tell me what $A \cap B$ looks like. What about $A \cup B$? What about $A - B$?
- Know the basic proof techniques – can you prove a result by induction, contrapositive, contradiction? You may need to sometimes prove a result by exhaustively searching through all possibilities.
- Be comfortable with thinking about abstract problems. Newsflash: many real-world problems are just special kind of abstract problems. Therefore, as computer scientists, it is in our best interest to solve problems from a more abstract model. You'd want to start getting comfortable with thinking about problems more abstractly rather than concretely, and this often comes with time and practice.
- Know how to work with limits and basic differentiation. The most amount of calculus that we will do in this course is differentiating polynomials. These are often useful for proving properties of Big-Oh.
- Know basic properties of graphs and combinatorics. You will be working with graph structures a lot, so you should try to get comfortable with the properties of connectedness, (a-)cyclicity, Hamiltonianicity, etc. Additionally, it is sometimes useful to think about certain problems combinatorially. In fact, *many* real-world problems are really just combinatorial problems. Therefore, it is a good idea to revise over basic combinatorial properties.

The formal prerequisite for COMP3821/9801 is a $70+$ in MATH1081 or equivalent, and having a firm grasp on the discrete mathematics concepts will be useful in your future endeavours within the theory of algorithms.

2.2 Computer Science Prerequisites

The basic prerequisite for the course is a $70+$ in COMP2521 or COMP9024, or equivalent. What this means is that you need to have a solid grasp on some of the data structure foundations. You will need to be comfortable with working with a wide range of data structures; for example, the running time of an algorithm changes when you're using an array as opposed to a linked list or a hash table. Using an appropriate data structure will improve the running time (and space complexity, but we do not discuss this too much in this course) of your algorithm.

Here's a basic checklist to get you up to speed with what you ought to know for COMP3821/9801.

- Know the basic time complexities of the operations for an array, hash table, graphs, (binary) trees. You may, occasionally, see AVL trees but binary-search trees are more than enough for our purposes.
- Know the different representations of a graph data structure and what the pros/cons are for each representation – you mainly need to be aware of the adjacency matrix and adjacency list representations, but other representations, such as incidence matrix and compressed sparse row, may be helpful.

- Know how to read Big-Oh notation. Big-Oh is useful for talking about the asymptotic behaviour of an algorithm. We will see Big-Theta and Big-Omega in this course, so first being familiar with Big-Oh will make the transition to lower bounds and tight bounds much easier to digest.

3 COMP3121/9101 vs COMP3821/9801

In this section, we will try and answer some common questions about the two courses and which course is the *right* course for you.

- *What is the difference between COMP3121/9101 and COMP3821/9801?*

  Answer: This differs each year depending on the lecturer involved but commonly, COMP3821/9801 will cover a few more topics, such as matroids and approximation algorithms, and algorithms, such as Dinic's algorithm. As a result, the Extended course places greater emphasis on theory than on application. This also means that assessments are generally harder, but this should not deter you from taking COMP3821/9801, hopefully at least!

- *Should I take COMP3121/9101 or COMP3821/9801?*

  Answer: ~~COMP3821/9801~~. It really depends on what you hope to get out of your algorithm journey. COMP3821/9801 has been redesigned to focus more specifically on algorithm *theory*, so while it can still be useful for job interviews, much of the focus will shift towards abstraction. By and large, you will be working with problems from an abstract setting and solving general problems, which require deeper fundamental understanding of the topics.

  Therefore, COMP3821/9801 is good for students who are interested in the academia route in theoretical computer science. Additionally, COMP3821/9801 is good for students who want to *learn* the theory of algorithms. Also, the IMC COMP3121/3821 Prize is usually awarded exclusively to COMP3821/9801 students... so make of that what you will.

- *How does COMP3821/9801 scale compared to COMP3121/9101?*

  Answer: Um... next question. Just kidding, this is an interesting question because scaling is never pre-determined. The bulk of the scaling occurs right at the end of the course and we look at how each assessment was marked. Were the marks where we expected them to be? Were the marks convincingly lower than our expectations? You will never be scaled down, though.

  Don't let scaling be a primary factor for choosing between COMP3121/9101 or COMP3821/9801, because it is not guaranteed in any offering.

4 Course Schedule

The following is a *tentative* plan of the course schedule. This is subject to change depending on how we progress throughout the term.

Week	Topic
1	divide and conquer
2	divide and conquer; greedy algorithms
3	greedy algorithms; matroids
4	greedy approximation algorithms; flow networks
5	applications of flow
6	optional interlude: expander graphs and hardness of approximation
7	dynamic programming
8	dynamic programming; computational complexity
9	computational complexity
10	fun and games in computational complexity; presentations

4.1 Lectures

There are two 2 hour lectures each week that cover the scope of the course. Lectures will be recorded and available to students each week. Full lecture notes will be available at the start of the term and these will be the full lecture notes for the entire course, so you will have full access to the lecture resources for your own preparation. Lectures will only cover a subset of the notes, the rest of the lecture notes will be additional reading for your own entertainment. There will be no lecture slides for the course; therefore, it is highly advised to come to the lectures and read through the lecture notes.

The lecture notes will clearly mark which parts are assessed material, and which parts are extra optional reading. Additional notes may be uploaded throughout the term.

4.2 Tutorials

Each week, there will be a 1 hour tutorial. There will be two tutors per tutorial who will guide you through the design and analysis process. At the start of each tutorial, there will be an assigned *guided problem* in which the tutor will cover in full. For the rest of the tutorial, you will work in small groups to solve as many problems as you can in the tutorial sheets. It is highly advised to come to the tutorials since this is the best opportunity for you to get help from your peers and demonstrators. Full solutions will be posted at the end of each week. Additional problems can be found in the lecture notes for your own practice. However, the lecture notes problems do not have solutions.

4.3 Labs

Each week, you will also have a 2 hour lab. In these lab sessions, you will individually work on a small set of tasks. At the start of the term, you need to pick an appropriate grade that you will primarily work towards. You'll then be given a set of tasks that match the grade that you have chosen, and you will need to complete them in full by the end of the term. There will be two lab demonstrators who will assist you in the process and mark you off when they deem that the quality of your work is at a satisfactory level. These problems do not have assigned marks to them *per se* but rather, you will receive a delayed mark for your labs. At the end of the term, you will construct a portfolio of tasks that you have completed and these will be submitted as part of your lab grade.

4.4 Drop-in Sessions

COMP3821/9801, along with COMP3121/9101, will be offering drop-in sessions where you can ask tutors questions about the course or material of the course. In addition to your labs, you can also get your lab problems marked by the tutor on duty. These will run every week. Specific timetables will be posted closer to the start of term.

4.5 Project Drop-in Sessions

There will be a project drop-in session each week where you can ask project mentors about guidance regarding the project. This is also a good opportunity to talk to fellow peers and your group about your project. Specific timetables will be posted closer to the start of term.

4.6 Forum

As usual, we will also have a public forum. In theory, anyone can join the forum. This is the perfect place to ask any questions about the course material or administrative things about the course. You can either opt to anonymously or non-anonymously ask any questions. We will have a forum moderator who will help facilitate discussions. However, we strongly recommend students to answer other student's questions. This is the best way to learn.

5 Assessments

In this offering of the course, the assessments are split into three main components: *lab problems*, *group-based project*, *final exam*.

5.1 Lab Problems

Weighting: 50%.

The main assessment is in the completion of lab problems. These lab problems are done on formatif. You will be given a set of tasks each week to complete. For the top marks, you are required to complete all tasks to a satisfactory level. The tasks can, in theory, be done within the lab. However, you will have ample time to make any corrections and rewrite your solutions so that the lab demonstrators are happy with your work. The marking system works as follows: you complete all of the problems and submit them at the end of term in a portfolio. Tasks are not worth marks themselves but rather, we grade your tasks based on the completion rate.

For a task to be deemed *complete*, the lab demonstrator needs to mark the task as *complete* and you will see the status of a task change to a green tick to signify that the lab demonstrator is happy with your work. If a task is submitted as *partially completed*, then we will still mark it but it will not be marked as complete and therefore, marks may be lost. Therefore, it is in your best interest to submit each task early to get as much feedback as possible. You do not necessarily need to be present at the lab to have your tasks marked off. However, it is in your best interest to do so to get the best experience.

5.2 Project

Weighting: 15%.

Since this is a research-driven course, the project is a great way to get your hands dirty in the process of research. The project will be a guided project in groups of 2 to 4. We highly discourage individual groups – you will not be graded differently to a group of 2 to 4. However, if you *really really* want to do a solo group, then you can. Please consult us beforehand, though.

The project is deliberately open-ended, giving you free-range to explore new areas on your own. We will provide sample projects and give you resources along the way. However, we encourage that you find your interests and do a project on the topics that motivate you. Of course, it should have some relation to algorithm theory.

There are three main checkpoints: week 3, week 7, week 10. In these checkpoints, you need to write a report detailing what you've learned and what you hope to focus your project on. Specific information regarding each of these checkpoints will be posted at least a week beforehand. However, there will be at least one formatif task each week for you to write about what you've learned in your process of discovery. These are not graded but rather, think of these tasks as *research journals*.

There will be a dedicated lecture (or two, depending on how many groups are presenting) where groups will present their work to the rest of the cohort. We may invite industry representatives and academics alike. Alternatively, your group may opt to write a report instead. We strongly encourage you to pick a project idea that you'd be proud to showcase to industry mentors and academics alike. More information on this will be released closer to the end of the term.

Throughout the term, we also encourage you to talk to staff members, your peers (even if they are not in your group), and students outside of the course. This is to help you simulate the research environment where everyone shares their own ideas. However, your group should try and do a project that is dissimilar to other groups, so that all groups are doing something unique.

5.3 Exam

Weighting: 35%.

There will be an invigilated exam done on Inspera during the exam period. You will be provided with practice exams which will be similar to the actual final exam. You will be allowed to bring hard-copy notes. More information about the exam will be posted in due time.

6 Future Courses and Resources

So, you've decided that you want to learn *more* about what goes in the research world of algorithms and its related fields. Well, I'm glad to have convinced you and I'd be happy to share!

6.1 Courses

Perhaps, the first place to start is to find courses that build on top of COMP3821/9801. Here are courses that come to mind, I most definitely will be missing a few.

- **COMP4121**: *Advanced Algorithms*. The course covers novel algorithm paradigms, such as randomised algorithms, and important algorithms used in the field of data science. You will learn about high-dimensionality (think the Gaussian Annulus and Johnson-Lindenstrauss lemma) and other methods for machine learning. The course requires deeper mathematical foundation. A solid background in linear algebra and statistics is preferred, in addition to COMP3821/9801.

  Offered in T3.

- **COMP4128**: *Programming Challenges*. If you are more of a programming kind of person, then COMP4128 is a good balance between algorithm theory and programming in practice. Since the course primarily teaches concepts often found in competitive programming, COMP4128 simulates the competitive programming environment, giving you first-hand experience in such a setting. Perhaps, you will be ready to take on ICPC after doing the course.

  Offered in T3.

- **COMP4141**: *Theory of Computation*. This course builds on top of the last topic of COMP3821/9801. We saw that certain problems can be solved efficiently because there exist an algorithm that correctly solves every instance of the problem. On the other side of the story, there exist problems that do not immediately have an efficient algorithm that solves every instance. And even with our major developments in algorithm theory, we still do not know if such an algorithm even exists. This course explores the complexity of these problems, which we can *intractable*. In particular, if we assume that certain problems are hard to solve, then we can easily show that other problems are also just as hard to solve. The running theme of reductions will become critical to the study of intractable problems.

  Offered in T1.

- **COMP6741**: *Algorithms for Intractable Problems*. This course covers material adjacent to the content taught in COMP4141 and therefore, will serve as a good complementary course. Much like COMP4141, the course primarily concerns itself with intractable problems. However, while COMP4141 teaches techniques to prove conditional lower bounds, COMP6741 introduces techniques to solve these problems *exactly*, even if the running time is exponential. This is still useful to study, because an algorithm with running time 2^n is much faster than algorithms with running time $(2+\epsilon)^n$ for $\epsilon > 0$. The course also covers potential tight lower bounds under certain conditions, such as the *Exponential Time Hypothesis*.

  Offered in T1.

- **MATH3171/5171** *Linear and Discrete Optimisation Modelling*. Many combinatorial problems can, indeed, be modelled as a linear program. You have a collection of *discrete* elements with a target function that you want to optimise. These are enough for you to construct a linear program. And by a reduction to a linear program, our goal is to now study the properties of the linear program: if we can solve the linear program, then we can obtain our optimal solution to the original combinatorial problem. Fortunately for us, the world of linear programming has been extensively studied and this course serves as a great introduction. While we focus on modelling problems as linear programs, MATH3171/5171 teaches you the theory that complements the modelling done in this course. Doing this course will give you a deeper appreciation of the theory of algorithms more broadly.

Offered in T3.

- **MATH5425**: *Graph Theory*. Graphs play a ubiquitous role in computer science and mathematics. Many problems in fields of networks, algorithms, data science, databases can be modelled more abstractly as a graph. Therefore, it is natural to study properties of the underlying abstract model of graphs. MATH5425 serves as a brilliant introduction to the rich field of graph theory. In this course, you'll learn properties of connectivity, matching theory, and colouring, all of which are elements of graph theory that we also exploit in the theory of algorithms: deciding if a graph is bipartite is merely deciding if a graph can be coloured using at most two colours, deciding if each man and woman can be paired together from a pool of candidates is merely deciding if a graph has a perfect matching, and so on. The course does require some mathematical maturity as it is targeted as a graduate-level course. However, completing the course will give you a much more profound perspective of graphs in computer science.

  Offered in T1.

- **MATH5505**: *Combinatorics*. Much of computer science theory concerns itself with discrete objects. Moreover, many problems that we study in algorithm theory are, by its nature, combinatorial. Therefore, it is natural to study combinatorial objects and their properties. MATH5505 covers one area among a vast literature of the field: *extremal combinatorics*. In this field, we study how large or small certain combinatorial objects can be under certain restrictions. For example, if I have a collection of sets that all have common pair-wise intersections, how large can this family of sets become? This is known as the *Sunflower Lemma* and surprisingly, this has significant applications to the field of boolean functions. This course will give you an introduction to such studies.

  Offered in T3.

6.2 Resources

There are many textbooks on introductory algorithm theory. The course notes should be ample resources. Each chapter will have a bibliography for further reading, so feel free to consult the texts and papers. Here are just a few resources for further reading:

- *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein.
- *Algorithm Design*, Kleinberg and Tardos.
- *Algorithms*, Erickson.
- *Extremal Combinatorics*, Juknas.
- *Graph Theory*, Diestel.
- *Introduction to the Theory of Computation*, Sipser.