## 1 Course Information

- **Instructor:** Quanquan C. Liu
- **Time:** MW 11:35am-12:50pm
- **Location:** DL 316 - Dunham Laboratory 316

## 2 Course Description

What techniques can we use to deal with modern real-world data with billions of data points? How do we account for strong adversaries that violate the privacy of users providing this data? This course will provide you with the knowledge to tackle research questions in these domains. We will propose answers and techniques to these broad questions from an algorithmic standpoint, presenting foundational topics such as:

- The **parallel, distributed, and streaming** models and algorithmic techniques commonly used within these models
- **Differential privacy** and mechanisms for private data analysis
- **Implementation techniques, tools, and examples** that demonstrate the practicality of these algorithms in real-world systems

This course focuses on advanced topics in practical graph algorithms with provable guarantees beyond the sequential model used in most introductory algorithms classes. Specific topics include local graph techniques for problems such as maximal matching, independent set, k-core decomposition, densest subgraphs, and coloring as well as global techniques for problems like connectivity, shortest paths, and spanners. Introductory lectures will also feature techniques used beyond graph algorithms. Students are asked to read and present influential recent research papers on these topics. Papers come from prominent CS theory conferences such as STOC, FOCS, SODA as well as database and data mining conferences like VLDB, PODS, and WWW. In addition to these presentations, students also work on a final project which may be theoretical or implementation-based.

The course will also feature voluntary ***open problem sections*** where we discuss (known) practice problems and open-ended research questions related to the topics in this course in a collaborative group setting.

## 3 Format and Workload

- **Two class presentations** on subjects of their choice related to the topics of the course. 50% of grade.
  - Finalize dates and topics for presentations before April 1: due Feb. 5.
  - Finalize dates and topics for presentations on and after April 1: due Feb. 26.
- **One final project** (individual or with a partner) 50% of grade.
  - Project proposal (1 page): due Feb. 26.
  - Progress report (2-3 pages): due March 27.
  - In-class presentation: last two weeks of class.
  - Final report (at least 8 pages, less than 20): April 24.

- **Optional:** Open problem session, time TBD.

Students will complete 1-2 class presentations and an independent final project on the topic of their choice. Visitors are welcome to present topics; simply let me know. In-class attendance is required. There will be an optional additional weekly open problem section. This course emphasizes the final project which can be a research, implementation, or reading project. Students can work individually or in teams of 2 and will receive regular feedback from me.

## 4 Prerequisites

This course is aimed at Ph.D. and M.S. students and, with my permission undergraduates with a strong background in algorithms and/or mathematics. In particular, students must have done well in an advanced algorithms course like CPSC 366.

Because the course is research-intensive, prerequisites will be strictly enforced. Undergrads who wish to take the course: please send a short email to quanquan.liu@yale.edu detailing your algorithms background, including relevant courses and performance in those courses, before enrolling.

## 5 Tentative Schedule

This is a very rough plan of the topics covered in this course; subject to change, specifically, student presentations can be shifted to other lecture dates by request.

1. *Jan 17*: Overview of Class and Logistics; Introduction to Parallel, Distributed, and Streaming Models
2. *Jan 19 (Monday Schedule)*: Local Graph Algorithms: Part 1 (Coloring, MIS, Network Decomposition)
3. *Jan 22*: Local Parallel/Distributed Graph Algorithms: Part 2 (Coloring, MIS, Network Decomposition)
4. *Jan 24*: Local Parallel/Distributed Graph Algorithms: Part 3 (Coloring, MIS, Network Decomposition)
5. *Jan 29*: Global Parallel/Distributed Graph Algorithms: Part 1 (MST, Connectivity, Shortest Paths)
6. *Jan 31*: Global Parallel/Distributed Graph Algorithms: Part 2 (MST, Connectivity, Shortest Paths)
7. *Feb 5*: Streaming Graph Algorithms
8. *Feb 7*: Dynamic/Batch-Dynamic Graph Algorithms
9. *Feb 12*: Learning-Augmented Graph Algorithms
10. *Feb 14*: Introduction to Differential Privacy
11. *Feb 19*: Private Graph Algorithms: Part 1
12. *Feb 21*: Private Graph Algorithms: Part 2
13. *Feb 26*: Implementing Graph Algorithms in the Real World
14. *Feb 29, Mar 4, 6, 25, 27, April 1, 3, 8, 10*: Student Presentations
15. *Apr 15, 17, 22, 24*: Final Project Presentations

## 6 Example Student Presentation Topics and Papers

Below is a non-comprehensive list of example papers and topics from which to choose to give your presentations. Students may also choose from papers not listed here for their presentations. The below example papers are organized under broad topics.

**Parallel, Distributed, Streaming and Dynamic Graph Algorithms**

- Improved Parallel Algorithms for Density-Based Network Clustering; ICML 2019.
- Greedy Sequential Maximal Independent Set and Matching are Parallel on Average; SPAA 2012.
- Improved Deterministic Network Decomposition; FOCS 2021.
- Densest Subgraph in Streaming and MapReduce; VLDB 2012.
- Parallel Batch-Dynamic Graphs: Algorithms and Lower Bounds; FOCS 2020.
- An Auction Algorithm for Bipartite Matching in Streaming and Massively Parallel Computation Models; SOSA 2021.
- New Diameter-Reducing Shortcuts and Directed Hopsets: Breaking the $O(\sqrt{n})$ Barrier; SODA 2022.
- Triangle and Four Cycle Counting with Predictions in Graph Streams; ICLR 2022.
- An Optimal Algorithm for Triangle Counting in the Stream; APPROX 2021.
- Parallel Double Greedy Submodular Maximization; NeurIPS 2014.
- Dynamic Graph Connectivity in Polylogarithmic Worst Case Time; SODA 2013.

**Differential Privacy**

- Lipschitz Extensions for Node-Private Graph Statistics and the Generalized Exponential Mechanism, FOCS 2016.
- Differential Privacy from Locally Adjustable Graph Algorithms: $k$-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs, FOCS 2022.
- Private Matchings and Allocations; STOC 2014.
- What Can We Learn Privately? FOCS 2008.
- Differentially Private Graph Learning via Sensitivity-Bounded Personalized PageRank; NeurIPS 2022.

**Practical Implementations with Strong Theoretical Guarantees**

- Ligra: A Lightweight Graph Processing Framework for Shared Memory; PPoPP 2013.
- Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing; SPAA 2017.
- Parallel Batch-Dynamic Algorithms for $k$-Core Decomposition and Related Graph Problems; SPAA 2022.
- Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable; SPAA 2020.
- Locally Differentially Private Analysis of Graph Statistics; USENIX Security 2021.

## 7 Final Project Guidelines

This section explains what I expect from the final course project which accounts for the majority of your grade. The main goal of the project is for you to use some of the advanced topics and thinking skills you have learned in this class to solve (potentially novel) problems. The course project has multiple objectives: finding and learning new material on your own, thinking critically about it, and presenting your work. Different projects will achieve these objectives in different ways. There are 3 main types of projects you can choose from. The structure of the final project in this course and the information below follows from the 6.854 Advanced Algorithms course at MIT. The below guidelines are paraphrased from the final project guidelines in the Fall 2014 iteration of the course.

**Reading Project** Pick a few challenging and interesting papers on a related topic to this course, and write a summary that combines their ideas. You will be graded on how much better/more informative your summary is than the original papers (so the original papers should be hard to read). The papers should be recent and not already covered by this class. You don't have to give all proofs in full detail, but the reader should understand how it all works. One paper is usually not enough; two or three are usually better. And don't just recap each paper separately: you need to show how they all connect. A common mistake here is to list a lot of results without proof. This is not very helpful. Instead, focus on some of the most important results and explain the main ideas and methods that lead to them.

**Theoretical Research** Come up with a new and interesting solution to an algorithmic problem in scalable and private graph algorithms. "Interesting" could mean faster or simpler. You will probably need to do some background reading, but there is no minimum requirement as long as you can go beyond what is known. This type of project will be graded on how much you improve on previous results. Like all research, you need to be ready for the possibility of not making any progress—what is your backup plan? It could be a reading project based on your background reading for the research. You should not submit a theory project that you have been working on for a while and just sent to a conference; first because it is too big and second because you should try something new related to the class material. This final project itself could even develop into an independent conference submission!

**Algorithm Engineering/Implement Project** We have seen many algorithms in theory. Their performance when implemented can be unexpected. Choose one that interests you, implement and test it. Or come up with a simpler version that gives the same guarantees as the state-of-the-art but is more readily implementable. This type of project will be graded on how well you describe the algorithm you are implementing, what kind of clever heuristics you used, what kind of hard test cases you ran them on, and how well you analyze the results. Algorithm papers often omit a lot of small implementation details that turn out to be important when you try to implement them. You can also try implementing heuristics that have "no theoretical value" but may make a huge difference in practice. Once you have an implemented algorithm, you can test it to see how it behaves in practice. This involves creating hard inputs that make the algorithm perform badly. It is also interesting to test on real-world datasets which I can provide. Studying the algorithm's behavior may lead you to modify the algorithm and test them. Be careful, however, as implementations can take a lot of time. Also, remember that there are existing papers in this field: before you code algorithm X, look for papers that have coded algorithm X. It's okay if you repeat previous results, but you should state that clearly. In reality, as computer systems keep evolving, you might find notable variations from previous codes.

More details about the final project will be given as we approach the midpoint of this class.