6.856 — Randomized Algorithms

David Karger

March 26, 2021 — Course Project

This handout provides a description of the kinds of things I am looking for in the course project. The basic goal of the project is for you to independently apply some of the advanced algorithmic thinking you have (hopefully) been developing in this class. The course project involves a combination of goals—encountering and independently absorbing new material, doing some thinking about it, and presenting what you have done. Different projects will attain these goals in different proportions.

Remember that the target reader is *another student in this class*. You can't assume they know more than a typical student who is taking this class. In particular, no other grad-level knowledge. So if you're explaining something quite foreign, you're going to have to take time to explain a lot of the background.

There are 3 main ways to approach the project.

Reading project. Find a few interesting but challenging theoretical papers on a coherent topic, and write up an exposition that synthesizes their ideas. You will be graded on how much clearer/more informative your exposition is than that of the original papers (so the original papers should be pretty tough). The papers should be new—not yet digested into the journal and textbook literature.

While you need not give all proofs in full detail, the reader should come away with a good understanding of why it all works. Unless the paper is extensive, one is usually not sufficient; two or three usually are. And simply summarizing each one isn't sufficient: you need a synthesis that relates the different parts. A common failing here is to engage in a lengthy enumeration of results without proof. This isn't particularly useful. Better to take some of the most important results and focus on presenting the key insights and techniques that lead to them.

Think about the contents of a typical lecture in this class: I don't provide all details, but (hopefully) give you the big ideas. You should aim for the same in your papers. I should finish without having slogged through a huge amount of mathematical formalism, but with a feeling that I understand the approach well enough that I could recreate the necessary formalism if trapped on a desert island for a few years.

Remember that your reading projects must *add* to what is in the papers you are writing about. So if the papers are super clear and well explained, your paper will have to do something more than just be super clear and well explained. You have to explain connections that weren't described, or explain things better than they were explained. This is why I recommend working off recent papers, on work that hasn't been well digested yet.

Theoretical research. Develop an interesting new solution to an algorithmic problem. "Interesting" may mean more efficient or may mean simpler. While you will presumably need to do some background reading, there is no set lower bound on it so long as you are able to advance beyond what is known. This kind of project will be graded on the extent of the improvement on previous results.

As with all research, you need to be prepared for the risk of not making any progress—what is your backup plan? Often, it can be a reading project built around your background reading for the research.

It is *not* appropriate to submit some theory project that you've been working on for a few months and just submitted to a conference; first because the scope is wrong (it's two big) and second because you should tackle something new related to the class material.

Implementation project. We have studied many algorithms in theoretical form. Their behavior when implemented can be quite surprising. Grab one that interests you, implement and test it as above. This kind of project will be graded on how well you explain the algorithm you are implementing, what kind of interesting heuristics you developed, what interesting test cases you ran them against, and how well you interpret the results.

Algorithms papers tend to leave out a lot of little implementation details that turn out not to be so little when the time comes to implement. You may also explore implementation of heuristics that have "no theoretical value" but may lead to enormous improvements in practice. Once you have an implemented algorithm, you can test it to see how it performs in practice. This involves devising interesting "hard" inputs that make the algorithm perform poorly. Study of the algorithm's behavior may lead you to make changes in the algorithm and test them. A typical implementation paper says "we implemented algorithm X and it was awful, then we added heuristic Y and it was great!"

Be cautious—implementations can take a lot of time. Make sure what you are trying to implement is a reasonably "small" algorithm. Give serious thought to test inputs—ideally, they will come from real-world problems, or will be specially designed to "stress" some aspect of the algorithm. You probably also need a control (i.e. dumb) algorithm to compare yours to.

Also, realize that your reader (me) may well not be familiar with the algorithms you are implementing; thus, your paper must give an adequate description of the algorithm, not just jump to the implementation results.

Finally, be aware that there is prior literature in this area: before you implement algorithm X, hunt for papers about implementations of algorithm X. It's fine if you replicate previous results, but you should be clear that's what you're doing. In practice, since computer architectures keep changing, you may discover interesting differences from prior implementations.

Warning: if you just dive in and implement, you are risking a B. If you don't read an exemplary implementation paper (e.g. by Goldberg) you won't understand what is needed to do one well. It isn't any harder than doing other kinds of papers, you just have to know what you are doing.

Regardless of which route you take, the end result should be a roughly 10 page paper describing the results. Presentation quality will be a factor in your grade. The paper should be written to be understood **by any other student in this advanced algorithms class**: you should not assume the reader has more knowledge than an undergraduate CS curriculum plus this class. Based on many papers that missed this mark, I **strongly recommend** that you trade readings with someone else in the class—you'll be surprised at how incomprehensible other students find your writeup, and will hopefully be able to make changes to fix that problem.

I will schedule some office hours during which we can discuss your projects. I won't have time to read anything, but if you want to try out an explanation of something you're reading, discuss some choices of experimental data sets, or float some ideas around a research problem you are tackling, I'll be happy to discuss.

FAQ

What topic should I work on? The best topic to pick is one you are interested in anyway. Many of you are already involved in some research project; some thought may reveal an algorithmic component. Perhaps your system is presently using a naive algorithm for X and could be improved by using a more sophisticated one. You can do background reading on the topic (reading project), develop a theoretical model of the problem and devise a solution (theoretical project) or take some extant algorithm and try it out (experimental project).

If you aren't working on anything, try browsing through what we've covered in class as well as papers (see below) to identify an area that sounds like fun.

You are **not** limited to the topics we have covered in class. Anything that uses ideas from combinatorial algorithms and a theoretical framework to devise more efficient solutions is fair game. However, being theoretical is not sufficient. You should steer clear of out-of-scope areas such as numerical analysis, distributed/asynchronous computing, complexity theory, etc. Rather, you should write a paper that draws heavily on the ideas of this class, even if applied to different topics. Again, the focus should be on algorithms; work proving e.g. the existence of certain combinatorial objects is discouraged.

Where can I find papers? There are numerous sources of papers on algorithms. The best work in the area is published in three conferences, each with yearly proceedings:

- The ACM Symposium on Theory of Computing (STOC).
- The IEEE Symposium on Foundations of Computer Science (FOCS).
- The ACM-SIAM Symposium on Discrete Algorithms (SODA).

There are lots of other sources, of course. There are also journals but they tend to be rather behind.

Is collaboration allowed? Yes, in fact recommended—group projects are much more fun, and you will all be able to work less. As with problem sets, group size should be limited to three so all can participate fully. All members should participate in the entire project. One collaboration strategy that has frequently *failed* in the past is to have each collaborator read and write up a single paper and staple the writeups together. The writeup needs to be coherent, drawing together the different elements of the project.

On the other hand, using from a group research project outside of class is discouraged. It's hard to be clear that sufficient work from the class was contributed in that case.

Can I write up research project X that I'm already doing? It is rare for a previous research project to fit exactly into the scope of this class. It's far preferable for you to take a brief break from whatever you were doing and try something new. On the other hand, trying a new direction *connected* to a project you are already working on is ideal—it may yield something you can bring back to your main research.

Which kind of project should I choose? A reading project is by far the least risky, the one where there's a pretty straightforward pathway to getting an A. On the other hand, if you can accept the risk and want to do a project that stands out, an implementation or research/application project is the way to go. Those are the ones I remember best when writing recommendation letters!

Which implementation language? For implementation projects, languages with automatic garbage collection can really impede any efforts to get at the underlying runtime behavior of an algorithm. You should avoid such languages, or measure something less language dependent than time, such as number of primitive operations (e.g. sums computed, edges added/removed, or hashes evaluated, depending on your problem).

How about lower bounds? This course focuses on the design of algorithms, and your project should as well. Sometimes those algorithms are informed by lower bounds, and conversely some lower bounds are developed using algorithms, but algorithms should be the focus.

What about tools from algebra and real analysis? Recently, linear algebra and convex optimization have been having a tremendous impact on theoretical computer science with e.g., the development of powerful new algorithms for (approximate) max-flow. However, this class has focused on combinatorial algorithms, and that's what you should do as well. We played with a bit of linear algebra in looking at fingerprinting, but you'll note that was simply addressing basic questions of zero-testing, not using more sophisticated arguments about polytopes or convexity. Similarly, we obviously use linear systems in solving linear programs, but in this class we look at LP as either a black box (for randomized rounding) or a geometry problem.

Traps to Avoid

- don't fill your paper so full of descriptions of algorithms that you don't have room to provide insight about why they work. Anyone can describe an algorithm; the purpose of this work is to justify the ones that you describe.

- Remember that this is a theory project. Studies (even implementation studies) need to begin from work that has analytic results. It is not appropriate to take some interesting algorithmic problem and then apply un-analyzed heuristics to it, no matter how much mathematics is involved in describing the heuristics.

- Conversely, make sure it is a *course-relevant* theory project. Just because it's formal/mathematical doesn't make it suitable. The project should be using the discrete combinatorial techniques taught in the class.

- On a specific topic: linear algebraic methods and convex optimization have recently become very important in algorithms, but they are *not* within the scope of this class. This class intentionally focuses on discrete/combinatorial topics, and your project should too.

- I am *very bad* at math. If your project has big sequences of equations, I will get confused and lose the thread of your argument (and may not be able to get it back). Try not to use any math more substantial than what you saw in lecture!

- Don't bite off more than you can chew. You have a limited amount of space; if you try to describe too much then you won't be able to provide enough detail to describe it well. I have had cases of one group submitting a project that covered a *subset* of what another group covered and getting a *significantly better grade* because less material allowed them to explain it better.

- Remember that I will be reading all these projects in limited time. Make sure what you've written can be fully absorbed in an hour, or I'll have to stop partway through your paper!

- Old material: as a rule of thumb, at least one of the papers you are writing about should be less than 5 years old. It may lead you to draw in older work, but if everything is old then you aren't looking at the cutting edge.

- In an implementation study, it's often a mistake to measure runtime, since that depends a lot on the implementation language, hardware, caching strategies, etc. It's a lot more meaningful and generalizable to test something that is "infrastructure independent"—for example, if I was implementing max-flow, I might count the number of shortest path computations, or the number of heap operations or number of changes to an edge's flow value. These will be unchanged regardless of hardware. Of course, runtime can be meaningful as well, but a non-time measure can give more confidence in the meaningfulness of the time measures.

- Avoid appendices. If it's in the appendix I won't have time to look at it. If it's not important enough to be in the main paper then it shouldn't be present at all.

- Don't go off topic. Your paper should refer to/leverage/relate to/build on the ideas taught in this class. Having a lot of math in your paper doesn't automatically put it in scope. Even algorithms is not automatically in scope, as we've steered clear of for example number theoretic algorithms and numerical analysis.