CS 787
Fall 2010
Homework 3: Project Proposal

If you are taking CS 787 for credit, you must do a project, involving analysis, literature survey, measurement/simulation of an algorithm, or some combination thereof. I also want you to go through the process of writing up your work for publication (think of this as part of your scientific training). The timetable is:

1. On or before Wednesday, November 24, give me *1-2 page* description of your intended project. This should include a succinct statement of the topic, how you intend to study it, and relevant citations to the literature. Include ideas about possible publication outlets. (Hint: where do the other people interested in your topic publish their stuff?)
2. On or before Monday, December 20 (Monday of exam week), give me your completed *10-20 page* paper. Include a cover letter to the journal editor or conference/workshop chair indicating that you wish it to be considered for publication.

Your topic should relate to the subjects considered in CS 787: efficient methods for solving discrete problems, analysis of algorithms, variations on the algorithm concept (parallel, quantum, randomized, ... algorithms), relevant mathematics (discrete probability, graph theory, etc.). To get your thinking going, here are some ideas. The list below is neither exclusive nor exhaustive.

Sample Project Ideas

1. Ross Johnson has invented a data distribution scheme that can be thought of in the following terms. Start with a set S of positive integers. Now, for $n = 1, 2, ...$ expand S by adding 1 to each element in S, and adding 2^{n+1} to the largest value in S that is congruent to 1 modulo 2^s. It is unknown how long it takes for an initial segment of integers to be covered. Study this algorithm.

2. The isolating lemma in the Mulmuley/Vazirani[2] matching paper looks like a wonderful trick. How else can it be applied? Start by looking this up in the index of Motwani/Raghavan.

3. In class, we presented an algorithm for bipartite matching based on increasing the flow in a network. In graph-theoretic terms, the algorithm uses paths that are alternating, in the sense of having every other edge from the current matching. Alternating-path and alternating-element algorithms are a big theme in combinatorial algorithm design. Study this topic.

4. Survey one of the course topics from a historical or biographical perspective. For example, imagine that your favorite algorithms author has just received a big award and you have been given the job of writing an article describing what they did.

5. Many problems have apparently fast "practical" algorithms, and somewhat slower "rigorous" algorithms. Two examples of this include integer factoring and linear programming. Pick a problem of this type and try to push the practice and the theory closer together. One thing you can do is look for special cases in which a rigorous algorithm can be sped up, or a heuristic algorithm can be proved correct. For an example of this kind of paper, see P.C. Gilmore et al., Well-solved special cases, in E.L. Lawler et al., The Traveling Salesman Problem (1985).

6. Learn about Groebner bases. These are an important algorithmic tool for solving problems involving multivariate polynomials, such as the following: given equations $f_1 = f_2 = \cdots = f_n = 0$, must another equation $g = 0$ always hold?

7. In our study of square root algorithms, we used the Euler criterion for $Zp*$:

$$\exists y(x = y^2) \Leftrightarrow x^{(p-1)/2} = 1.$$

This is an "exactness" theorem, in the following sense: we have two group homomorphisms (squaring and raising to the $(p-1)/2$-th power) and the theorem says that the image of one is the kernel of the other, and their composition is trivial. What other exactness principles have algorithmic uses? (Note: some of the standard linear programming "alternative theorems" look suspiciously like a generalization of exactness.)

8. Investigate streaming algorithms. This is a model inspired by the internet. One place to start is Alon, Matias, and Szegedy, The space complexity of approximating the frequency moments, Journal of Computer and System Sciences 58 (1): 137-147. (First presented at 1996 ACM STOC.)

9. The number of algorithms appearing in the literature far exceeds the number of algorithms that have been implemented in working code. Pick a problem that has an interesting new algorithm, and try to get it to work.

10. [Suggested by Susan Horwitz.] One of the algorithms (node selection) in the paper by Jannsen and Corporal, Making Graphs Reducible with Controlled Node Splitting, ACM Trans. Prog. Langs. Sys., 1997, requires one to test for membership in a certain type of set (called a Shared External Dominator). It is not obvious how this can be done.

11. [Suggested by Jerry Zhu.] We are given $x_1, \ldots, x_n$ in C^d. In machine learning jargon, we have n items, each represented by a d-dimensional feature vector, and each feature has the same discrete domain C.) The labels of the items are $y_1, \ldots, y_n \in C$. Consider a partition of the d features ("feature split") into two disjoint sets with d_1 and d_2 features, respectively, so $d = d_1 + d_2$. We write the feature split as $x = [x^{(1)}, x^{(2)}]$ where $x^{(i)} \in C^{d_i}$.

    If the feature split were conditionally independent given the label, we would have

$$\Pr[x^{(1)}, x^{(2)}|y] = \Pr[x^{(1)}|y] \times \Pr[x^{(2)}|y]$$

    Here, the probabilities would be estimated from the data we have. However, in general an arbitrary feature split will not be conditionally independent given the label. Is there a good algorithm to find a feature split that is as close to conditional independence as possible? Closeness is up to you to define.

13. Michael Jordan's recent UW talk mentioned an interesting discrete probability model, called the Chinese Restaurant Process. Study the algorithmic applications of this.

14. Our version of the Hungarian algorithm for bipartite matching had an initial "greedy" stage, which produced (on average) a fairly large matching. Find other algorithms with greedy initial stages, and analyze them.

Ideas for Literature Searching

a) The *Science Citation Index* allows you to answer the following question: given any paper, what other papers cited it? This has an on-line version called *Web of Science*, available through UW Libraries. Other databases that serve a similar purpose include Google Scholar and Citeseer.

b) For computer science, the most important abstracting journals are *Computer and Control Abstracts* (part of *INSPEC*), *Computing Reviews* (part of *ACM Computing Archive*), and *Mathematical Reviews*. These should all be available on-line now through the UW Libraries.

c) Major conferences in theoretical computer science include IEEE Symposium on Foundations of Computer Science (FOCS), ACM Symposium on Theory of Computing (STOC), ACM-SIAM Symposium on Discrete Algorithms (SODA), International Conference on Automata, Languages, and Programming (ICALP). Recent proceedings from these are all good places to dig for research topics.

d) Get to know the preprint servers in your areas, such as arXiv, ECCC, etc.

e) If you are interested in writing an expository article, you will need to know where such articles get published. MAA has several journals devoted to mathematical exposition, including the American Mathematical Monthly and Mathematics Magazine. CS theory surveys (called "columns") are often published in SIGACT News (ACM). There is (or was) a very nice expository journal devoted to probability and statistics, The American Statistician. Finally, CACM and IEEE Computer publish articles of general interest to the computing community. (Breaking news: SIAM has a new journal, Undergraduate Research Online, see `www.siam.org/students/siuro/`.)