CS369M: Algorithms for Massive Data Set Analysis - Lecture 1 - 09/21/09

Introduction and Overview

Lecturer: Michael Mahoney

**Unedited Notes*

1 A few introductory thoughts

In this class, we will be interested in the analysis of algorithms for large-scale data analysis problems. In recent years, researchers from a number of related research areas (*e.g.*, computer science, statistics, machine learning, and applied mathematics to name just a few) have focused on problems that fall under this general umbrella, motivated in large part by technological developments (in scientific, Internet, etc. domains) that permit the automatic generation of extremely large data sets. Thus, there are a number of complementary perspectives one can adopt on these problems. In this class, we will be primarily interested in theoretical questions having to do with the worst-case analysis of algorithms that are of interest in real applications.

An important theme in the class will be that some of the mathematical structures that give us worst-case theoretical guarantees also give us other things that are at least as important (like good statistical properties), and thus we will be interested in understanding "when" and "why" different types of algorithms work in terms of those structures. Since real data sets are often (but certainly not always) modeled as graphs or matrices, we will be primarily interested in the analysis of algorithms for very large graph and matrix problems or for data sets that can be well-modeled as such.

Today, we will mostly introduce some general themes to which we will return later, and next time we will start to get into some of the more technical details.

2 Massive and non-massive data sets

There is a lot of interest in massive data sets and in the development of algorithms for massive data sets. Thus, let's start with a few words on what we might mean when we talk about massive versus non-massive data sets. There are a number of ways that one can try to formalize the distinction between small and medium and large data. One of my favorite is the following.

- Small: We will call the size of a data set *small* if you can look at it and fairly easily find a good solution to problems of interest with any of a number of algorithmic tools.

- Medium: We will call the size of a data set *medium* if it fits in the RAM on a reasonably-priced laptop or desktop machine and if you can run computations of interest on it in a reasonable length of time.
- Large: We will call the size of a data set *large* if it doesn't fit in RAM or if you can't relatively-easily run computations of interest in a reasonable length of time.

These distinctions are clearly informal and vary with time. With respect to small data sets, the intuitive point is that if there is structure of interest then almost any "reasonable" algorithm should work for small data sets. (That is not to say that small data sets are not of interest; they can be very valuable for testing algorithms and, in particular, of understanding in a controlled environment where and why algorithms succeed or fail.)

With respect to large data sets, the point is that as we go from medium-sized data to large data sets, the main issue is that we don't have random access to the data, and so details of memory access, etc., are often the primary issues. This can create enormous difficulties that are currently the focus of a great deal of research. These issues, e.g., dealing with database issues underlying trying to make large-scale vector space and graph and machine learning (or other) computations work in distributed MapReduce-style or multicore-style environments will *not* be our primary concern. That being said, much of our discussion will be motivated by and will keep an eye on those issues, and there are a number of interesting questions related to this that we will at least touch upon toward the end of the semester, time and interest permitting.

Thus, most of the class will be on algorithms that, to the extent that they have been implemented, have been implemented and tested on medium-sized data sets. But they have been chosen to be to be representative of types of algorithms that are currently being scaled up or could be particularly broadly-applicable for large-scale data applications in upcoming years.

3 A high level view of large data

Examples of some of the perspectives or approaches to large-scale applications that people in recent years have take include:

- Streaming: Think of this as when data are continuously generated or they are stored externally "on tape" and you can pass over the data one or two or a small number of times. This originated in telecommunications and Internet applications, and it was particularly popular theoretically in the 90s due to the simplicity and tractability of the model.
- Multi-core: This arises since now we can have multiple simple CPUs on a single chip since that is not the bottleneck to cost. This is of particular interest at places like national labs and in applications of scientific computing, it is related to some variants of parallelism; and cache issues, etc., are particularly of interest here.

- MapReduce paradigm: This arises since in many large-scale applications you have large clusters of relatively cheap machines. It leads to different parallelism issues, and arises in Internet applications, astronomy and high energy physics applications, and increasingly in other applications that generate and have to store large quantities of data.

Some general properties of large data sets that will influence what types of algorithms are appropriate to consider include that they are:

- Large and often distributed.
- Dynamic—e.g., created at different times.
- Heterogeneous–e.g., created by many users or under different conditions.
- Noisy—"random" versus "adversarial" noise lead to different issues.
- Unstructured—must "model" the data to run algorithms.

Different application areas tend to generate very different types and quantities of data and tend to find very different types of questions of interest. These are worth keeping in mind as we discuss when and why different types of algorithms work well or don't work well. Although far from exhaustive, here are a few application areas to keep in the back of your mind for when we discuss different types of algorithms that are appropriate in different application domains:

- Internet—Google, Yahoo, Facebook, etc., including text, images, links, bids, etc.
- Business and Telecommunications—customer transactions at places like WalMart, ATandT, etc.
- Finance—consumer behavior data and high-frequency financial transactions data on Wall Street and in hedge funds.
- Biology and Medicine—microarrays, DNA SNPs, medical images, etc
- Astronomy and HEP—sky images, HEP experimental data, etc.

To oversimplify the situation, areas toward the top of this list had their original more in computer science, algorithms, and databases, while areas toward the bottom of the list tended to have more "statistical" perspective (in a generalized sense, i.e., they may have had a good implicit or explicit model for where the data came from, and thus they could draw heavily from that knowledge, even if it wasn't formalized as a traditional statistical model). It is important to note that as the ability to generate large data has become more ubiquitous, there has been more of a convergence between these two perspectives. For example, physics and biological applications now increasingly deal with database issues, while in recent years there is much more sophisticated modeling for problems like behavioral targeting in Internet applications.

4 Different perspectives on large-scale data problems

Traditionally, and at a high level, there have been two very different perspectives on how to approach large-scale data problems. They are:

- CS and Databases—worries more about worst case analysis, models of data access, running times, etc.
- Statistics and more recently ML—worries more about models for the data, inference, prediction, etc.

Although much of our discussion will be framed within the usual perspective of worst-case TCS analysis, it is important to realize that large-scale data problems are forcing a fundamental shift in the way people think about worst-case analysis of algorithms. This shift is even more significant than saying that large data problems require us to reevaluate the traditional polynomial versus exponential-time distinction and instead consider "streaming" or "multi-pass" models of data access. The traditional perspective in computer science is something like the following. The study of algorithms for large data problems is of interest since large data applications clearly require the use of fast algorithms. Moreover, most problems of interest are intractable, either in general in the sense of being NP-hard or NP-complete or hard within the models of data access that are appropriate given the size of the data. Thus, we have to "settle" for obtaining a suboptimal approximation to the exact but intractable solution, or we have to "settle" for using randomness in our algorithms.

On the other hand, in many applications, we don't care about whether (say) the cluster we found has an objective function value that is such-and-such close to the optimum; instead, we care whether there is some control on the approximation quality and whether the cluster is robust, etc. so that what we found is useful in some downstream sense. Moreover, whereas the worst-case P versus NP distinction might capture a useful qualitative notion of "easy" versus "hard" problems, worst-case analysis bounds for approximation algorithms are not analogously useful—they are extremely pessimistic and have been much less successful at providing even qualitative guidance for the appropriateness of approximation algorithms in data analysis applications.

Thus, a theme that will be underlying much of that we do is the following. The tools or mathematical structures that are used in worst-case analysis to obtain quality-or-approximation guarantees, *e.g.*, eigenvectors, vector space concepts, convexity, LPs, SDPs, metric embeddings, etc., often have fortuitous side benefits that are good for big data, *e.g.*, geometry, smoothing, implicit regularization, good boosting properties, etc. (Said another way, similar structures and tools arise in and are used in other areas to guarantee these good properties.) Thus, far from simply giving us worst-case bounds, these structures provide all sorts of other statistical-like or data-like things that can make these algorithms and their output particularly useful for large-scale data applications. Thus, rather than simply obtaining worst-case bounds, we will be also interested in what is going on "under the hood" to tell us when and why those algorithms should be used in applied environments.

5 Modeling data as matrices and graphs

The first thing to note is that data are whatever they are—images; snippets of text; links between web pages; a table with bids, bid terms, and time stamps; a list of DNA base pairs; color-coding on a microarray chip; etc. Thus, in order to do something useful algorithmically with data, typically the first step is to model data in some way. *This is often the most difficult step*—if you come up with a good representation for you data, then often any of a wide range of algorithms will reveal qualitatively similar insights, while if you fail to come up with a meaningful representation, then basically no algorithm will find anything useful. The criteria for such for such a modeling step are twofold:

- (1) the representation should be sufficiently flexible that it can capture meaningful properties of the data; and
- (2) the representation should be sufficiently structured that computations of interest can be performed on it in order to obtain meaningful answers in a reasonable time.

There are numerous ways in which to structure data, but two classes of structures that are at a "sweet spot" in this trade-off are matrices and graphs:

- Matrices: Assume that we have m pieces of data, each of which can be described by n features. For example, we could be considering microarray data, in which case we might be considering m genes, each of which can be described by its response in one of n environmental conditions. Alternately, one might be considering a corpus of documents, in which case we have term-document data. Then we can view this as an $m \times n$ matrix A. Note that the issues that arise here are very different that in traditional NLA—there you worry about backward error analysis, while here if you can get one or two digits of precision, you have gained a lot.
- Graphs: Assume that we have a set V of things or entities, and we have a set E describing binary interactions between pairs of those entities. For example, this could be an interaction graph model of a social/information network or something else. Then we can view this as a graph $G = (V, E)$. Note that the issues that arise here are very different than in traditional TCS analysis of graph algorithms. At root, this is since traditional graph algorithm analysis assumes there is "no noise"—in most applications, if data are anything then they are noisy.

Although graphs and matrices are only two ways to model data, they are by far the most common in many application areas, in large part since they provide a nice trade-off between descriptive flexibility and algorithmic structure. In addition, they provide two quite different perspectives, although there are a lot of connections between the two approached, e.g., as discussed in spectral graph theory. Finally, it is worth noting that these two approaches are so powerful that people go to a lot of work to put the data into one of these two places. For example, at several points, we will point out how our discussion of matrix and/or graph algorithms couples to much of this work in machine learning.

(As an aside, although it will not be our main focus, the area of *machine learning* is a first cousin of much of what we will talk about. Machine learning is an area that studies computer algorithms for learning to do something. For example, one might be interested in a classification problem, e.g. predict the answer to "Will it rain tomorrow?", in which case the goal is to learn Yes/No, or a $\{-1, +1\}$ function. Alternatively, one might be interested in a regression problem, e.g., predict the answer to "How much will it rain tomorrow?", in which case the goal is to learn a number or a real-valued function. The goal is to have automatic methods, i.e., with little or no human intervention, and to do the learning based on seeing a bunch of observations/examples, typically defined as a feature vector. In this case, the algorithms clearly have space, time, communication, number of passes, etc. as scarce resources, but in addition another scarce resource is the amount of data that the algorithm needs in order to learn.)

For those of you familiar with the topic, until we get to that point, you might wonder how general some of the matrix or graph algorithms we will discuss are in terms of dealing with real data, especially since real data might not have the "nice" properties that are assumed for matrix and graph algorithms. The topic is a complex one, but to oversimplify consider the following. You can think of a Reproducing Kernel Hilbert Spaces (RKHS) as a place that is a generalization of the nice fixed-dimensional Euclidean space that (1) provides much greater descriptive flexibility, and at the same time (2) is not too much worse algorithmically. Defining features such that you get an appropriate RKHS for the problem on hand is something people spend a lot of time on.

6 Overview of topics to be discussed

Here is a brief description of what we will cover: Algorithmic and statistical methods for large-scale data analysis: matrix and graph algorithms; strengths and weaknesses of theoretical techniques for practical scientific and Internet data analysis; overlap with related problems in statistics, optimization, numerical analysis, and machine learning. In particular, representative topics include:

- Matrix problems (numerical and statistical perspectives; algorithmic approaches, including Johnson-Lindenstrauss lemma and randomized projection and sampling algorithms; novel matrix factorizations);
- Graph problems (graph partitioning algorithms, including spectral methods, flow-based methods, and recent geometric methods; local graph algorithms and approximate eigenvector computation); and
- Applications to machine learning and statistical data analysis (motivating applications; algorithmic basis of the RKHS method; geometric data analysis, regularization, and statistical inference; boosting and its relationship to conjugate gradient methods, duality, convexity, online learning, and approximation algorithms).
- Basics of implementing these ideas in medium and large-scale applications.

In terms of prerequisites, we will assume some familiarity with and understanding of Algorithms (e.g., CS161), Linear Algebra (e.g., Math51), and Probability Theory (e.g., CS109), or equivalent. Please contact me if you have any questions about this.

7 References

There are a number of relatively nontechnical references that would be useful to read to get a feel for the perspectives we will adopt, including:

- [3] is Fayyad, Piatetsky-Shapiro, and Smyth on "From data mining to knowledge discovery in databases"
- [6] is Smyth on "Data mining: data analysis on a grand scale?"
- [1] is Breiman on "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)"
- [4] is Lambert on "What use is statistics for massive data?"
- [2] is Donoho on "Aide-Memoire. High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality"
- [5] is Poggio and Smale on "The Mathematics of Learning: Dealing with Data"

References

[1] L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[2] D.L. Donoho. Aide-memoire. High-dimensional data analysis: The curses and blessings of dimensionality, 2000. `http://www-stat.stanford.edu/ donoho/Lectures/AMS2000/Curses.pdf` Accessed December, 17 2008.

[3] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[4] D. Lambert. What use is statistics for massive data? In J. E. Kolassa and D. Oakes, editors, *Crossing boundaries: statistical essays in honor of Jack Hall*, pages 217–228. Institute of Mathematical Statistics, 2003.

[5] T. Poggio and S. Smale. The mathematics of learning: Dealing with data. *Notices of the AMS*, 50(5):537–544, May 2003.

[6] P. Smyth. Data mining: data analysis on a grand scale? *Statistical Methods in Medical Research*, 9(4):309–327, 2000.