CS264: Beyond Worst-Case Analysis
The Top 10 List*

Tim Roughgarden†

December 3, 2014

We covered a *lot* of concepts in this course. To appreciate this, let's review the overarching narrative of the course via a top 10 list.

1. *Instance optimality.* We began the course with this concept, an input-by-input guarantee that is the strongest-possible notion of optimality. This guarantee is so strong that instance-optimal algorithms don't exist for many problems, but it sure is great when they do exist. Our primary example was a proof that the Kirkpatrick-Seidel algorithm is instance-optimal for the 2D Maxima problem (among all "order-oblivious" algorithms); the homework also covered the "Threshold Algorithm" for searching sorted lists.

2. *The spectacular failure of worst-case competitive analysis of online paging algorithms.* This motivated introspection about the purpose of algorithm analysis frameworks. We singled out three goals: the Prediction/Explanation Goal, which strives for accurate performance predictions of an algorithm; the Comparison Goal, which aims to recommend the "best" algorithm for a problem; and the Design Goal, where the point is to organize brainstorming for new algorithmic ideas. In the context of online paging, we introduced some analysis techniques — resource augmentation and loose competitiveness — to get closer to meeting some of these goals.

3. *Parameterized analysis.* The goal here is a fine-grained understanding of the performance of an algorithm on all inputs, not just on worst-case inputs. A good parameterized analysis is a prerequisite for an instance-optimality guarantee, but is also broadly useful. Our "killer app" was a parameterized analysis of the page fault rate of several online paging algorithms in terms of the locality of the page request sequence. This yielded guarantees that could actually be taken at face value, and finally separated the performances of the LRU and FIFO paging policies. We also gave a parameterized analysis of heuristics for the Maximum-Weight Independent Set problem (in terms of

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

the minimum degree, of the input graph or of the optimal solution), as well as further examples later on in the context of smoothed analysis (in terms of a "condition number").

4. *Clustering is difficult only when it doesn't matter.* That is, in most clustering applications we only care about instances in which there is a "meaningful clustering," and such instances exhibit non-worst-case structure that is exploitable by an efficient algorithm. We saw multiple formalizations of this principle. For example, in metric k-median instances where the optimal solution is invariant under multiplicative perturbations of the distances (up to a factor of 3), the single-link++ algorithm (which combined single linkage/Kruskal with pruning by dynamic programming) recovers the optimal solution in polynomial time.

5. *Exact recovery of planted/ground truth solutions via linear programming.* We saw several examples: perturbation-stable solutions to cut problems like multiway cut, via a proof that uses randomized rounding in the analysis (but not in the algorithm!); sparse solutions to underdetermined systems of linear equations (i.e., compressive sensing), via almost Euclidean subspaces; and LP decoding of expander codes up to a constant fraction of errors, via a "dual certificate" argument.

6. *Semi-random and hybrid data models.* These are a sweet spot for algorithm analysis for many problems, with assumptions strong enough to enable interesting positive results, yet weak enough that only "robustly good" algorithms do well. For example, in the semi-random planted clique problem, a simple degree-based heuristic fails where smarter algorithms succeed.

7. *Smoothed analysis of algorithms.* This is a semi-random model for running time analysis, where nature adds a small perturbation to an adversarially chosen input. The simplex algorithm for linear programming and many different local search algorithms run in expected polynomial time on smoothed instances, in line with extensive empirical evidence, despite running in exponential time on worst-case instances.

8. *Smoothed complexity of problems.* The key result here is that, within a natural class of problems, a problem is solvable in (randomized) pseudopolynomial time if and only if it is solvable in smoothed polynomial time. We first proved one direction of this statement for the Knapsack problem, via a smoothed Pareto curve analysis, and then proved the general result using a novel algorithm based on the "Isolation Lemma."

9. *Hashing with pseudorandom data.* We gave a theoretical explanation for the good empirical performance of simple hash functions, using another semi-random input model. As long as the data has "a little bit of randomness" in it, universal hash functions perform just as well as (unimplementable) completely random functions.

10. *Unknown and partially known input distributions.* We saw several algorithms that perform well despite knowing little to nothing about the assumed distribution over

inputs — almost as well as the optimal algorithm for the distribution. For the problems of sorting, pricing, and online network design, we gave algorithms that need only a few samples from the distribution to do well. In our final lecture, we closed the course's circle of ideas by showing how instance optimality-type results can be viewed as a natural strengthening of a guarantee for an unknown distribution, with the class of "competing algorithms" taken as all algorithms that are optimal with respect to a distribution of a certain type. One instantiation of this general idea gives a novel interpretation of the standard "fixed-action benchmark" used to define and analyze no-regret algorithms for online decision-making, such as the "Follow the Perturbed Leader" algorithm.