CS264: Beyond Worst-Case Analysis
Lecture #19: Online Algorithms and Random Permutations[*]

Tim Roughgarden[†]

December 1, 2014

1 Preamble

Our final two lectures bring together multiple themes of the course to reason about a couple of new problem domains. In this lecture we discuss online network design. The next lecture covers online decision-making. We've discussed online algorithms before, in the context of paging algorithms (Lectures #2 and #4). While paging is a paradigmatic online problem, online algorithms are relevant in many other settings as well.

2 The Online Steiner Tree Problem

2.1 Problem Statement

In an instance of the online Steiner tree problem, the following ingredients are known up front:

- an undirected graph $G = (V, E)$;
- a nonnegative edge c_e for every $e \in E$;
- a root vertex $r \in V$.

Recall that in an online problem, some part of the input arrives piecemeal, and an irrevocable decision must be made at each step. Here, a sequence of *terminals* $s_1, s_2, \ldots, s_t \in V$ arrives online, one by one. The goal is to maintain a low-cost tree that spans the root r and all of the terminals that have arrived so far. Each time a new terminal s_i arrives, the current tree

[*]©2014, Tim Roughgarden.

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

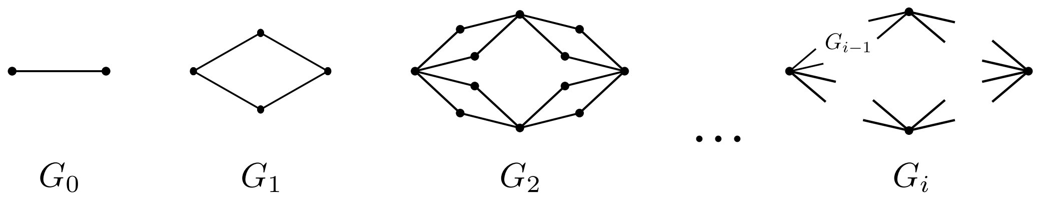

Figure 1: Diamond graphs used to create hard instances for online Steiner tree algorithms. For each $i > 0$, G_i comprises 4 copies of G_{i-1}. All edges have cost 1.

must be expanded so that it spans s_i as well. Previously built edges can never be deleted. The cost incurred by the algorithm is the sum of the costs of all of the edges chosen by the algorithm, over all iterations. One can imagine, for example, an Internet service provider laying cable as new suburbs pop up.

2.2 The Worst-Case Model

Before considering alternatives to worst-case analysis, let's examine what the worst-case model has to say about the problem [7]. Here, everything about the input, and in particular the terminals and their order, is adversarially chosen.

2.2.1 A Lower Bound

Let's begin, as usual, by drawing a line in the sand and understanding what we can hope for. We describe next a bad sequence of instances for all online Steiner tree algorithms. These instances occur in the following *diamond graphs*.

The 0th diamond graph G_0 is a single edge. For $i > 0$, the ith diamond graph G_i is obtained from four copies of G_{i-1} by identifying endpoints in a diamond shape; see Figure 1. Note that G_k has exactly 4^k edges; the number of vertices is slightly fewer but similar. Observe that we can naturally view the vertices V_{i-1} of G_{i-1} as a subset of the vertices V_i of G_i.

We consider diamond graphs in which all edges have cost 1 and the root r is the rightmost vertex. By a "line," we mean a "left-to-right" path from the leftmost vertex to the root. Every line in G_k has total cost 2^k.

For every online algorithm A, one can describe a sequence of terminals in G_k such that (i) all of the terminals lie on a common line (and hence the optimal solution has cost at most 2^k); and (ii) A suffers cost $\Omega(k2^k)$ on the sequence. This implies a worst-case lower bound of $\Omega(k) = \Omega(\log n)$ on the competitive ratio of every online algorithm.

The idea of the sequence is as follows; Homework #10 asks you to provide the details. Suppose the first request is for the leftmost vertex (which corresponds to a vertex of V_0); this is "phase 0." The algorithm A builds edges P_0. The cost of these edges is at least 2^k. Also, P_0 either skips one of the two vertices of $V_1 \setminus V_0$ (e.g., if it is just a left-to-right path) or has cost significantly more than 2^k; assume the former. Then, in phase 1, there is a request for the skipped vertex of $V_1 \setminus V_0$. This forces A to incur an additional cost of at least 2^{k-1}. Also, unless A incurs additional cost by building extra edges, two of the four vertices of $V_2 \setminus V_1$ are not spanned by the paths chosen by A so far; predictably, these will be the requests that show up in phase 2 (the order doesn't matter). In general, in every phase $i > 0$, there are 2^{i-1} requests, chosen so that A has to incur an additional cost of at least 2^{k-1} to span them all.

2.2.2 An Upper Bound

A matching upper bound is provided by the natural greedy algorithm that, upon the arrival of a new terminal s_i, connects s_i to the tree-so-far via a minimum-cost path. For every online Steiner tree instance with t terminals, this algorithm produces a feasible solution with cost $O(\log t)$ times that of an optimal Steiner tree. The proof is reminiscent of another argument that you might know, that the (offline) MST-based approximation algorithms for the Steiner Tree and Traveling Salesman Problems yield solutions with total cost at most twice that of the minimum-possible.[1] For the details of the proof, see Homework #10.

2.2.3 Discussion

The last time we applied traditional worst-case competitive analysis to an online problem, the paging problem, the results were disastrous. (Remember what we learned: every (deterministic) algorithm might well fault every single time step!) For the online Steiner tree problem, it yields more reasonable results. First, in the spirit of our Comparison Goal (recall Lecture #1), we learned a sensible-sounding lesson: without any knowledge of the input, no algorithm does better (in the worst case) than the greedy algorithm. The skeptic might ask: what else could we do? Conceivably, it could be better to build some extra edges, beyond what is necessary to connect a newly-arriving terminal, to make it cheaper to connect several other terminals in the future. Without any knowledge about the input, however, it's hard to know where such additional edges would prove useful, and the optimality of the greedy algorithm reflects this. Below, when we assume a distribution over the terminals, we'll see that non-greedy algorithms can perform better than the greedy algorithm.

We also discovered that the best-possible worst-case guarantee of an online algorithm is $\Theta(\log t)$, where t is the number of terminals. While super-constant, this is far less dire than the inevitable guarantee of k (the cache size) for (deterministic) online paging algorithms.

Overall, we conclude that worst-case competitive analysis yields some interesting insights about how to design and what to expect from online Steiner tree algorithms. Still, it's well motivated to consider non-worst-case models of online Steiner tree inputs, and to understand how modifying the input model changes how we should reason about and solve the problem.

[1] For TSP, this guarantee applies to the following two versions of the problem: (i) when the input graph G is complete and the edge costs satisfy the triangle inequality ($c_{uv} + c_{vw} \leq c_{uw}$ for every $u, v, w \in V$) and (ii) when the input graph and edge costs are arbitrary but repeated visits to vertices are allowed.

2.3 The Unknown I.I.D. Model

2.3.1 The Model

The following distributional model for the online Steiner tree problem is from [5]. The graph, edge costs, and root vertex are again known up front. Terminals arrive online, with each terminal s_i drawn i.i.d. from a distribution D over V. Our goal is to design an online algorithm that outputs a Steiner tree with expected cost at most a constant factor times the expected cost of a minimum-cost Steiner tree. Here, all expectations are over the randomness in the input (i.e., over the s_i's). We know from Section 2.2 that such a guarantee is impossible for worst-case inputs.

The first thing to check is whether our reigning champion, the greedy algorithm, already achieves the desired guarantee. Does the extra distributional assumption on inputs drop the algorithm's approximation guarantee from a logarithmic factor to a constant factor? The answer is no: relatively simple examples show that the expected cost of the greedy algorithm can exceed the expected cost of an optimal solution by an $\Omega(\log t)$ factor (see Homework #10). That is, the greedy algorithm's worst-case performance is also its average-case performance for a suitable choice of an input graph and a distribution D. We conclude that, to achieve our goal of a constant approximation factor, we need to design a new algorithm.

A second point is that our performance goal in unachievable without some a priori knowledge of the distribution D over terminals. The intuitive reason is that, for every algorithm A, there is an input graph and a terminal distribution D that "simulates on average" the bad worst-case example for A from Section 2.2. (The details are somewhat tricky; see [5].) We conclude that, not only do we need a new algorithm, but we need one that exploits advance knowledge of the distribution D.

Returning to the model, we now make two additional assumptions. First, we assume that the number t of samples that is going to show up is known a priori. (The identities of these terminals are not, of course, known in advance.) The positive results that follow continue to hold without this assumption, although they require modifications to the algorithm and analysis, resulting in worse constant-factor approximation guarantees (see [5]). Second, in the spirit of the previous lecture on pricing with an unknown distribution, we assume that the algorithm is given t i.i.d. samples $q_1, q_2, \ldots, q_t \sim D$ "for free." All of these samples are available to the algorithm before the "real" terminals $s_1, s_2, \ldots, s_t \sim D$ show up.

2.3.2 Discussion

There are two natural interpretations of the final assumption above, that the algorithm is privy to t i.i.d. samples from D. The first is literal, and is the same as last lecture: the algorithm designer simply does not know the underlying distribution D over terminals, and has only partial information in the form of samples (e.g., from comparable scenarios in the past). Given this information, we seek an online algorithm that has performance almost as good as the optimal online algorithm with full advance knowledge of the distribution D.

In the second interpretation of the model, the algorithm designer *does* know the distribution D and thus could, in principle, deploy the online algorithm with the minimum-possible expected cost. The problem is that this optimal algorithm might be very complicated, essentially an exponential-size look-up table that maps each possible history-so-far (the first $i-1$ terminal realizations) to the set of edges to build next. The obvious problem with such an algorithm is that it might be infeasible to implement. Another possible problem is a lack of robustness: if the designer's understanding of D is incorrect, or if this distribution changes over time, it's not clear that an optimal algorithm for an incorrect or outdated distribution is a good one to use. In this interpretation, the goal of the model is to guide us to a "simple" online algorithm — one that only depends on the (known) distribution D through t samples and, as we'll see, is also simple in other respects — while sacrificing as little as possible in expected solution quality.

2.3.3 The Augmented Greedy Algorithm

We next describe an *augmented greedy algorithm* that uses the given t i.i.d. samples from D in a natural way and achieves the desired performance guarantee. The algorithm begins with a preprocessing phase, before any of the actual terminals have shown up. Recall that at this stage, the algorithm knows the graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, the root vertex $r \in V$, and a set Q of t i.i.d. samples from D. The augmented greedy algorithm uses a version of the "MST heuristic," implemented using a Prim-like greedy rule, to build a low-cost tree spanning $\{r\} \cup Q$. Formally:

1. Initialize $R = \{r\}$, $S = Q$, and $T = \emptyset$.

2. For $i = 1, 2, \ldots, t$:

   (a) Let q_i denote a terminal of S that is closest to a vertex of R, where distance is measured in G using the edge costs c_e.
   [Comment: let L_i denote this distance.]

   (b) Let P denote a minimum-cost path from q_i to a vertex spanned by T (or to r, if $T = \emptyset$).
   [Comment: since all vertices of R are spanned by T, the cost of P is at most L_i.]

   (c) Add P to T.

   (d) Remove q from S and add it to R.

See Figure 2 for an illustration.

The second phase of the augmented greedy algorithm just runs the greedy algorithm, relative to the edges already built in the preprocessing phase.

1. Inherit T from the preprocessing phase.

2. For $i = 1, 2, \ldots, t$, on the arrival of the terminal s_i:

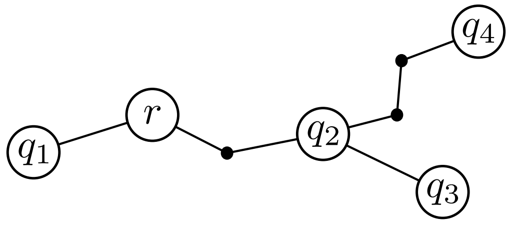

Figure 2: Preprocessing step of the augmented greedy algorithm. Sampled terminal q_1 is the closest one to r, q_2 the closest to $\{r, q_1\}$, q_3 the closest to $\{r, q_1, q_2\}$, and so on.

(a) Let P denote a shortest path from s_i to a vertex spanned by T.

(b) Add P to T.

To understand the high-level intuition behind the augmented greedy algorithm, recall our earlier idea of building additional edges as insurance so that future terminal arrivals could be accommodated at low cost. This idea bore no fruit in the worst-case model. Here, the augmented greedy algorithm uses the t samples from D to build infrastructure in the locations where future demand is most likely to be. The main theorem of this lecture shows that this is a good idea in the current model: the augmented greedy algorithm produces a solution with low expected cost, no matter what the underlying distribution D is.

Theorem 2.1 ([5]) *For every graph $G = (V, E)$, nonnegative edge costs, root vertex r, and distribution D over V, the expected cost of the output of the augmented greedy algorithm is at most 4 times that of the expected cost of an optimal solution.*

2.3.4 Proof of Theorem 2.1

The proof of Theorem 2.1 has two steps. The first step is to prove that the expected cost incurred by the augmented greedy algorithm in its preprocessing step (over the random choice of the samples Q) is at most twice the expected cost of a minimum-cost Steiner tree (over the random choice of terminals $s_1, \ldots, s_t$). Since the samples and the "real" terminals are distributed identically, this reduces to proving that, with probability 1 over the random choice of Q, the cost incurred by the augmented greedy algorithm in its preprocessing step is at most twice that of a minimum-cost tree spanning $\{r\} \cup Q$. This is simply the well-known fact that the "MST heuristic" is a 2-approximation algorithm for the Steiner tree problem. In fact, we'll use the stronger guarantee that $\sum_{i=1}^{t} L_i$ is at most twice the cost of an optimal Steiner tree spanning $\{r\} \cup Q$. (Recall the definition of L_i from the algorithm's description, and that $\sum_{i=1}^{t} L_i$ is an upper bound on the cost of the final tree.) If you haven't seen this argument before, see Homework #10.

The second and more novel step is to prove that the expected cost incurred by the augmented greedy algorithm in its second phase (with the s_i's) is at most the expected value of our upper bound $\sum_{i=1}^{t} L_i$ on the cost incurred in its first phase (with the q_i's). In light of our existing bound on the latter cost, this second step completes the proof of Theorem 2.1.

To argue this second step, first note that in every iteration $i = 1, 2, \ldots, k$ of the preprocessing phase, L_i equals the distance (in G) between q_i and some vertex of $\{r\} \cup Q \setminus \{q_i\}$. Thus, if we define X as a random variable equal to the shortest distance in G between one vertex randomly sampled from D and a set of $k-1$ such vertices, together with r, then $\mathbf{E}[L_i] \geq \mu$ for every i, where $\mu = \mathbf{E}[X]$. (Equality holds when $i = t$, but not otherwise.) Using linearity of expectation, we conclude that the expected value of our upper bound $\sum_{i=1}^{t} L_i$ on the cost incurred by the augmented greedy algorithm in its preprocessing phase is at least $t \cdot \mu$.

In each iteration $i = 1, 2, \ldots, t$ of the second phase of the augmented greedy algorithm, the new terminal $s_i \sim D$ is connected to the tree-so-far T via a shortest path. This tree spans r as well as at least $t-1$ (t, even) terminals that were sampled i.i.d. from D. Thus, the expected cost incurred by the augmented greedy algorithm in an iteration i of its second phase is at most μ. By linearity of expectation, the expected cost of the second phase is at most $t \cdot \mu$. This completes the second step and the proof of Theorem 2.1.

3 Random Permutations and Secretary Problems

The goal of this section is to introduce a new semi-random model, more challenging than the unknown i.i.d. models that we've been discussing, that is particularly well suited to online problems. It is called the *random permutation model.*

3.1 The Random Permutation Model

In the random permutation model, an adversary chooses an arbitrary (worst-case) input. Then, nature presents the input one piece at a time, in an ordering that is chosen uniformly at random. The goal is to design algorithms that work well no matter what input the adversary chooses (provided it is randomly ordered). This is in the spirit of other semi-random models we've seen, like smoothed analysis and pseudorandom data, where an adversary chooses a worst-case input distribution subject to a "sufficient randomness" constraint. Like other semi-random models, this model is appealingly agnostic about the details of the input distribution. Since the values of n i.i.d. random samples from a distribution are randomly ordered (conditioned on the unordered set of values), positive results in the random permutation model apply directly to the unknown i.i.d model.

3.2 The Secretary Problem

The canonical problem in the random permutation model is the *secretary problem.* Probably you've heard of it: you are presented with n numbers one-by-one (e.g., the quality of a candidate you just interviewed), and at each time step you can either accept the current number (hire the current interviewee) or continue to the next one. The goal is to accept the highest number in the sequence, even though you have no idea what the overall set of numbers is. When the numbers are adversarially chosen but then randomly ordered, it is easy to stop on the largest number with 25% probability: watch the first $n/2$ numbers go by, and stop on the first subsequent number that exceeds them all (if any). As long as the highest number is in the second half of the sequence and the second-highest number is in the first half of the sequence ($\geq 25\%$ probability), you are guaranteed to stop on the highest number. An optimized version of this algorithm and analysis yields a success probability of $\approx \frac{1}{e} \approx 36.7\%$; see Homework #10.

3.3 The Matroid Secretary Problem

It is fun to muse about generalizations of the secretary problem, for example where the algorithm can accept more than a single number of the input sequence. An abstraction that captures many interesting generalizations is the *matroid secretary problem* [1]. There is a universe U of secretaries, each with a value, and a set $\mathcal{F} \subseteq 2^U$ of *feasible sets*. Both U and $\mathcal{F}$ are known to the algorithm in advance. The values $\{v_i\}_{i \in U}$ of the secretaries are unknown a priori, and are presented to the algorithm in random order. Each time an algorithm sees a new secretary, it must decide irrevocably whether or not to accept it. The algorithm must terminate with a set S of secretaries that is a feasible set — i.e., with $S \in \mathcal{F}$. The goal is to design an algorithm that, for every set of values $\{v_i\}_{i \in U}$, outputs a set that is feasible (with probability 1 over the random order of U), and has expected total value as close as possible to — ideally, within a constant factor of — the maximum total value of a feasible set.

For example, the standard secretary problem can be captured by this general framework by taking $\mathcal{F}$ to be the singleton sets plus the empty set. In the natural generalization where the algorithm can hire up to k secretaries, $\mathcal{F}$ consists of all subsets of U that have cardinality at most k. It is not too difficult to extend the guarantee of $1/e$ for the secretary problem to the k-secretary problem for all k; see Homework #10 for details.

The matroid secretary problem imposes two restrictions on the feasible sets $\mathcal{F}$. First, $\mathcal{F}$ should be *downward-closed*, meaning that $S \in \mathcal{F}$ and $T \subseteq S$ implies that $T \in \mathcal{F}$ as well. This assumption obviously holds in most natural generalizations of the secretary problem, including the k-secretary problem. The second restriction is called the *exchange property*, and it insists that whenever $S, T \in \mathcal{F}$ with $|T| < |S|$, there exists $i \in S \setminus T$ such that $T \cup \{i\} \in \mathcal{F}$. This assumption also clearly holds for the k-secretary problem: if S includes more secretaries than T and both contain at most k secretaries, then adding any secretary of $S \setminus T$ to T increases the size of T by one without violating feasibility. Thus, the k-secretary problem is a special case of the matroid secretary problem.

There are several other natural special cases of the matroid secretary problem. In the *graphical secretary problem*, U is the edge set of an undirected graph $G = (V, U)$, and a subset $F \subseteq U$ is feasible (i.e., in $\mathcal{F}$) if and only if (V, F) is an acyclic graph. It is clear that such an $\mathcal{F}$ is downward closed; a simple argument (see Homework #10) shows that the exchange property also holds. See Homework #10 for an algorithm for the graphical secretary problem that guarantees expected value at least a constant fraction of the maximum possible. Another special case is the *transversal secretary problem*, where the secretaries are the left-hand-side vertices of a bipartite graph $G = (U, V, E)$, and $F \subseteq U$ belongs to $\mathcal{F}$ if and only if there exists a matching in G such that every vertex in F is matched. (Think, for instance, of V denoting locations, and edges denoting the locations that a secretary can work, with only one worker permitted per location.) Like the previous example, it is clear that $\mathcal{F}$ is downward-closed, and the exchange property follows from a combinatorial argument. There is also an online algorithm for this problem that achieves a constant-factor approximation guarantee [2, 8]. For a final example, the *representable matroid problem*, let U denote a set of vectors over a field $\mathbb{F}$, with $F \subseteq U$ in $\mathcal{F}$ if and only if the vectors F are linearly independent over $\mathbb{F}$. It is again obvious that $\mathcal{F}$ is downward-closed, and elementary linear algebra verifies the exchange property. While online constant-factor approximation algorithms are known for special cases of this problem [3], it is an open question whether or not such an algorithm exists for all representable matroids. The *matroid secretary conjecture* [1] states that every matroid secretary problem admits an online algorithm that achieves a constant approximation. Currently, the best approximation factor known for the general matroid secretary problem is $O(\log \log k)$, there k is the rank of the matroid [9, 4].[2]

3.4 Online Network Design in the Random Permutation Model

Online network design problems, as in Section 2, are natural candidates for analysis in the random permutation model. While the model does not tell us anything new about the specific problem of online Steiner tree — as mentioned earlier, the worst-case lower bounds carry over even to the easier model of an unknown i.i.d. distribution [5] — it does yield interesting insights into some generalizations of the problem.

For example, in the *online k-connectivity problem*, an algorithm must maintain k edge-disjoint paths between each pair of vertices in $\{r, s_1, \ldots, s_t\}$, where $s_1, \ldots, s_t$ are terminals that arrive online. The online Steiner tree problem is precisely the special case in which $k = 1$.

For every $k \geq 2$, every online algorithm has a worst-case approximation factor of $\Omega(t)$, where t is the number of terminals. In the random permutation model, however, the natural greedy algorithm is guaranteed to produce a solution with expected cost $O(\log t)$ times that of an optimal solution.[3] See [6] for details on these results.

References

[1] M. Babaioff, N. Immorlica, and R. D. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 2007.

[2]The exchange property implies that every maximal independent set of a matroid has the same cardinality. This common size is called the *rank* of the matroid. For example, in a graphical matroid with graph $G = (V, U)$, the rank is precisely $|V| - c(G)$, where $c(G)$ denotes the number of connected components of G.

[3]When a new terminal s_i arrives, the graph-so-far is augmented in the cheapest way possible subject to containing k edge-disjoint paths between s_i and $\{r, s_1, \ldots, s_{i-1}\}$. This can be implemented in polynomial time via a minimum-cost flow computation (exercise). To prove that the final solution is feasible, use induction and the max-flow min-cut theorem (exercise).

[2] N. B. Dimitrov and C. G. Plaxton. Competitive weighted matching in transversal matroids. *Algorithmica*, 62(1-2):333–348, 2012.

[3] M. Dinitz and G. Kortsarz. Matroid secretary for regular and decomposable matroids. *SIAM Journal on Computing*, 43(5):1807–1830, 2014.

[4] M. Feldman, O. Svensson, and R. Zenklusen. A simple $O(loglogRank)$-competitive algorithm for the matroid secretary problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015. To appear.

[5] N. Garg, A. Gupta, S. Leonardi, and P. Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 942–951, 2008.

[6] A. Gupta, R. Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. *SIAM Journal on Computing*, 41(6):1649–1672, 2012.

[7] M. Imase and B. M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[8] N. Korula and M. Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Proceedings of the 36th Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 508–520, 2009.

[9] O. Lachish. $O(loglogRank)$ competitive ratio for the matroid secretary problem. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 326–335, 2014.