CS420+500: Advanced Algorithm Design and Analysis

Lectures: March 22nd + 24th, 2017

Prof. Will Evans *Scribe: Ben Chugg*

In this lecture we:

- Defined and discussed hardness of approximation with applications to the TSP;
- Discussed online algorithms with applications to page replacement;
- Introduced competitive analysis.

As an interesting addendum to the last lecture, we remark that it was in fact Christos Papadimitriou who showed that the Euclidean Travelling Salesman Problem (and Euclidean Tour TSP and Manhattan TSP) is NP-Complete [1].

1 Hardness of Approximation

There are hard problems, and then there are HARD problems. How does one capture the intuitive notion that even approximating certain problems is computationally difficult?

We may proceed along the same lines as when showing any problem is NP-hard or NP-complete. Consider the following example.

1.1 TSP is Hard to Approximate

First, recall that a Hamiltonian cycle in a graph is a cycle that visits each vertex exactly once. The Hamiltonian Cycle Problem (HCP) asks, given an unweighted graph G, does G contain a Hamiltonian cycle?

Lemma 1. *Unless $P = NP$, there is no polynomial-time c-approximation algorithm for* TSP*.*

Proof. We will reduce from HCP; we begin with the construction. Suppose that $\mathscr{A}$ is a polynomial time c-approximation algorithm for TSP. We will use $\mathscr{A}$ to solve HCP. Let an unweighted graph $G = (V, E)$ be given, and set $n = |V|$. Consider the complete graph on n vertices, K_n, which can obtained by filling in all non-existent edges in G. Write $K_n = (V, E \cup \overline{E})$ where $\overline{E}$ is the complement edge set to edges of E. Define a weight on the edge set of K_n as follows. Let $w : E \cup \overline{E} \to \mathbb{R}$ give the edge weights and set

$$w(u, v) = \begin{cases} 1, & \text{if } (u, v) \in E \\ cn + 1, & \text{if } (u, v) \in \overline{E} \end{cases}.$$

Now, run $\mathscr{A}$ on the graph K_n, and let the result be denoted $TSP_{\mathscr{A}}(K_n)$. If the size of the tour given by $\mathscr{A}$ is less than or equal to cn, that is, $|TSP_{\mathscr{A}}(K_n)| \leq cn$ then return yes. Otherwise return no.

To see correctness, suppose that $|TSP_{\mathscr{A}}(K_n)| \leq cn$. Then this tour must have used only original edges from the graph (other edges weigh too much) and, by definition of a tour, visited each vertex exactly once. Since vertices of K_n are vertices of G, it follows that G contains a Hamiltonian Cycle.

Conversely, suppose that G contains a Hamiltonian Cycle. We claim that this tour constitutes a solution to TSP in K_n. This tour has weight n since there are n edges on the tour and since all edges have weight ≥ 1, there cannot be a smaller tour. ◀

Remark 2. In the above proof, we have not used the fact that c is a constant. Indeed, c can be any polynomial time computable function with codomain $\mathbb{R}_{\geq 1}$. Note that it must be polynomial time computable because constructing the input graph to $\mathscr{A}$ must be a polynomial time construction.

2 Online Algorithms

Consider an optimization problem in which the values over which we're optimizing are not all known. An algorithm which must make decisions based only on the parameters seen so far is called an *Online Algorithm.* More formally, for an input sequence $p_1, \ldots, p_n$ (where n is typically large), an online algorithm must produce an output given $p_1, p_2, \ldots, p_i$ - without knowledge of $p_{i+1}, \ldots, p_n$ - for all $i \in \{1, \ldots, n\}$.

2.1 Page Replacement in Caches

To initiate our exploration of online algorithms, we will study online page replacement. We briefly recall some facts from the study of operating systems. A machine is said to have cache size $k \in \mathbb{N}$ if it can fit k pages in its cache. A page is some fixed length block of (virtual) memory. Programs access data by means of the cache, therefore the cache must update the pages it keeps based on the requirements of a program.

Let us formalize the problem of online page replacement.

Online Page Replacement. Let $p_1, \ldots, p_n$ be a sequence of page requests made by a program on a machine with cache size $k \in \mathbb{N}$. Before each page request, p_i, the cache contains some k pages which may or may not include p_i. If p_i is not in the cache (i.e., a page fault), then a page is evicted from the cache and p_i is added. We define the cost of a page replacement algorithm, $\mathscr{A}$, on a sequence $p_1, \ldots, p_n$ of requests to be

$$f_{\mathscr{A}}(p_1, \ldots, p_n) \equiv \text{ number of page faults on } p_1, \ldots, p_n.$$

Some examples of online replacement algorithms are as follows:

1. Least Recently Used (LRU). Here, the algorithm evicts the page whose most recent request occurred furthest in the past (i.e., the page was the *least recently used*).
2. Least Frequently Used (LFU). Here, the algorithm evicts the page that has been requested the least often.
3. Marking Algorithms (a class of algorithms). Here, each page is labeled as either marked or unmarked. Initially, all pages begin as unmarked. If a page is requested, it is marked. When

a page fault occurs, it evicts a random unmarked page. If all pages are marked, it clears the marks and keeps going.

4. First-In-First-Out (FIFO). Here, pages are evicted in the order in which they were received.

The optimal, yet impossible, algorithm is Belady's algorithm commonly known as Furthest in the Future. Here, we evict the cache page which is used the furthest in the future. For a proof that Belady's algorithm is optimal see [2].

2.2 Analyzing Online Algorithms

Typically, computer scientists tend to be pessimistic and are interested in the worst case performance of algorithms. Consider, however, attempting a worst case analysis for online page replacement. We may always adversarially pick a sequence of page requests that gives worst case performance (e.g., all distinct pages). Therefore, for any online page replacement algorithm $\mathscr{A}$,

$$\max_{p_1,\ldots,p_n} f_{\mathscr{A}}(p_1,\ldots,p_n) = n,$$

and this doesn't give us any insight into the differing performances of distinct algorithms.

Our next attempt might be to try an average case performance analysis. Let m be the total number of pages possibly requested, and assume that the pages are independent with respect to one another and are drawn from a uniform distribution. Recalling that k is the cache size, k/m is then the probability that an arbitrary page is in the cache. Therefore, $1 - k/m$ is the probability than a page is not in the cache, and

$$\mathbb{E}[f_{\mathscr{A}}(p_1,\ldots,p_n)] = \sum_{i=1}^{n} \Pr(\text{Page } p_i \text{ not in cache }) = \sum_{i=1}^{n}(1-k/m) = n\left(1-\frac{k}{m}\right),$$

for any algorithm $\mathscr{A}$. This may be a better approximation than the worst case analysis, however it is still not unique to the algorithm. We turn, therefore, to competitive analysis as a more discriminating method of online algorithm analysis.

2.2.1 Competitive Analysis

In the competitive analysis of an online algorithm, we ask how its performance compares to the best offline algorithm.

Definition 3. *An online algorithm $\mathscr{A}$ is c-competitive if there exists $\alpha \in \mathbb{R}$ such that for all sequences of requests $p_1, p_2, \ldots, p_n$,*

$$f_{\mathscr{A}}(p_1,\ldots,p_n) \leq c f_{OPT}(p_1,\ldots,p_n) + \alpha,$$

where OPT is the best offline algorithm which knows the future. The factor c is called the competitive ratio of $\mathscr{A}$.

Remark 4. What's the deal with the constant additive factor α? It represents cost that is not associated to the size of the input. Roughly speaking, if $\alpha \leq 0$, then $\mathscr{A}$ is "better" (with respect to cf_{OPT}) by a constant additive amount α on any input. Similarly, if $\alpha \geq 0$, then $\mathscr{A}$ is "worse" by α on any input.

We conclude by stating two theorems which will be proved next class.

Theorem 5. *LRU and FIFO are k-competitive online page replacement algorithms, where k is the cache size.*

Theorem 6. *If $\mathscr{A}$ is any deterministic c-competitive online algorithm for paging then $c \geq k$.*

Combining these two theorems imply that LRU and FIFO are optimally competitive online page replacement algorithms.

References

[1] Christos H. Papadimitriou, The Euclidean travelling salesman problem is NP-complete, *Theoretical Computer Science*, Volume 4, Issue 3, 1977, Pages 237-244.

[2] Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.