CS 354: Unfulfilled Algorithmic Fantasies April 1, 2019

Lecture 1: Query Complexity

Lecturer: Aviad Rubinstein *Scribe: Josh Brakensiek*

1 Why study hardness?

The overarching goal of this class is to study *hardness*, evidence or proof that certain algorithms do not exist. Why is this a worthwhile study? There are many such reasons.

- *Understand possibilities.* There is not much point looking for an algorithm if it cannot exist!
- *Design.* Knowing that certain algorithmic barriers exist can help inform the design of algorithms which circumvent these roadblocks.
- *Science.* In the study of computer *science*, knowing that an algorithm does not exist is just as fundamental as whether it does exist.
- *Art.* To prove a lower bound, one needs to just show that one family of instances is hard. The lack of constraints *a priori* gives the prover much artistic freedom in showing a lower bound.

2 Conditional vs unconditional hardness

Roughly each week, we will explore a new type of intractability. The first two weeks cover *unconditional* lower bounds for *restricted* models of computation. The last eight weeks will cover *conditional* hardness. Such results will rest on some unproven assumptions, like $\mathsf{P} \neq \mathsf{NP}$ and the Unique Games conjecture.

3 Query Complexity

In the *query complexity* model of computation, instead of directly being given the input, it is concealed behind an *oracle*. To access the input, an algorithm is allowed to *query* the oracle certain information about the input. The goal is then to solve the problem at hand by minimizing the number of queries need. Importantly, the amount of computation between queries is not constrained!

Example 1 (Sorting an array). Given as input an array A on n elements, one can ask an oracle for any two indices $i, j \in [n] =: \{1, 2, \ldots, n\}$ whether $A[i] > A[j]$. It is a well-known theorem (e.g., [4]) that the optimal number of queries is $\Theta(n \log n)$.

Example 2 (Gradient descent). Given as input a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, one can ask an oracle for its gradient $\nabla f(x)$ for any $x \in \mathbb{R}^n$. Finding the optimal number of such queries to approximate an extrema of f is a well-studied problem (see [1]).

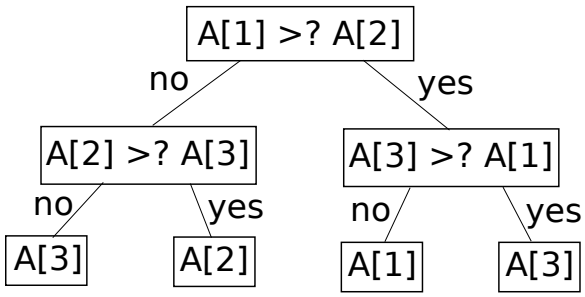

Figure 1: Illustration as to how a query algorithm can be viewed as a decision tree. In this case, the decision tree is computing the maximum element in a 3-element array.

3.1 Connection to decision tree complexity

Any query algorithm can be viewed as a *decision tree.* Have the internal nodes of the tree represent the queries and the leaves represent the solutions. The branching based on the response the oracle makes for that query (see Figure 1). Thus, the optimal query complexity for a given task is equal to the depth of the optimal decision tree which computes the given task.

3.2 Utility

The query model of computation does not perfectly correspond to the Turing machine model of computation. For example, the $\Theta(n \log n)$ query lower bound, it oblivious to known linear-time algorithms for sorting, like radix sort [4]. Even so, there are still many reasons query complexity is a fruitful area of study.

- Because the query model is simpler than the Turing machine model, it is much easier to prove tight lower bounds.
- For many natural problems, the query complexity is close approximation of the Turing machine complexity.
- In some models of computation, a certain operation can be much more expensive that others, such as computing the value of a submodular function (next section). Abstracting that operation as a query can assist in studying the complexity of that particular model of computation.

4 Optimizing submodular functions

The remainder of this week is concerned with the query complexity of *submodular optimization.* We start by defining what a submodular function is. We let $[n] := \{1, 2, \ldots, n\}$, $2^{[n]}$ be the set of all subsets of $[n]$, and $\mathbb{R}_+$ be the set of nonnegative reals.

Definition 4.1 (Submodular function)**.** A function $f : 2^{[n]} \to \mathbb{R}_+$ is *submodular* if for all $S \subset T \subset [n]$ and for all $i \in [n] \setminus T$,

$$f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T).$$

In other words, the marginal contribution of any particular decreases as the collection of elements gets larger.

Definition 4.2 (Monotone)**.** We say that a submodular function $f : 2^{[n]} \to \mathbb{R}_+$ is *monotone* if for all $S \subset T \subset [n]$, $f(T) \geq f(S)$.

4.1 Examples of submodular functions

There are many interesting examples of submodular functions.

Example 3 (Cut function). Let $G = (V, E)$ be an undirected graph on vertex set V and edge set E. We define the *cut function* $f : 2^V \to \mathbb{R}_+$ to be for all $S \subset V$,

$$f(S) := |E \cap S \times (V \setminus S)|.$$

Studying cut functions is known to have applications in computer vision.

Example 4 (Coverage function). Let $G = (V, E)$ be an undirected graph. We define the *coverage function* $f : 2^V \to \mathbb{R}_+$ to be for all $S \subset V$,

$$f(S) := \{v \in V \mid v \text{ is adjacent to some } w \in S\}$$

Such a function could be useful to study in advertising on social networks: find a small subset of people on the social network who could give exposure to a much larger fraction of the network.

Example 5 (Economic valuation). Let $[n]$ be a collection of goods, we let $f : 2^{[n]} \to \mathbb{R}_+$ be a buyers valuation of the collection of goods. A common assumption is that f has *diminishing returns*, which is equivalent to f being submodular.

Example 6 (Entropy). Let $X_1, \ldots, X_n$ be a joint distribution of random variables. Then, $f : 2^{[n]} \to \mathbb{R}_+$, defined by

$$f(S) := H(\{X_i\}_{i \in S} | \{X_i\}_{i \notin S})$$

is a submodular function. Studying such functions can help in machine learning by understanding which variables are the best to measure so as to minimize remaining entropy.

4.2 Optimization problems

There are many different optimization problems one can study for submodular functions. Often, the problems cannot be solved exactly efficiently, so *approximation algorithms* are studied. We say that an algorithm gives an α approximation, if the value output by the algorithm is within a factor of α of the true optimum.

The following a few of the most commonly studied optimization problems for a submodular function.

1. *Unconstrained non-monotone maximization:* given any submodular function, find its maximum value.
   - A 1/2-approximation using $\mathsf{poly}(n)$ queries is known. [2]
   - This is also tight. [6]

2. *Unconstrained non-monotone minimization.* given any submodular function, find its minimum value.
   - Known it can be solved exactly in $\tilde{O}(n^4)$ queries in the most general setting. [3]
   - Open: prove an $\Omega(n \log n)$ lower bound on the number of queries to f.
3. *Cardinality constraint monotone maximization:* given a monotone submodular function $f : 2^{[n]} \to \mathbb{R}_+$ and $k \in \{0, 1, \ldots, n\}$, find $S \subset [n]$ of size k which maximizes f. In other words, "pick the best k" coordinates of f.
   - A $1 - 1/e$ approximation using $\mathsf{poly}(n)$ queries is known. [5]
   - A $1 - 1/e$ lower bound is also known. [6]

4.3 Warm-up: the greedy algorithm

For the third problem on the list, we show that a simple greedy algorithm gives a $1 - 1/e$ approximation.

Algorithm 1 Greedy algorithm for monotone maximization with a cardinality constraint.

$S_0 \leftarrow \emptyset$
for $i = 0, 1, \ldots, k-1$
  $e_{i+1} \leftarrow \operatorname{argmax}_e f(S_i \cup \{e\}) - f(S_i)$
  $S_{i+1} \leftarrow S_i \cup \{e_{i+1}\}$
return S_k

To analyze this greedy algorithm, we prove the following claim by induction.

Claim 4.3. *For all $i = 0, 1, \ldots, k$,*

$$f(S_i) \geq \left(1 - \left(1 - \frac{1}{k}\right)^i\right) f(S^*), \tag{1}$$

where $S^* = \{x_1, \ldots, x_k\}$ maximizes f.

Proof. For our inductive base case, $i = 0$, the statement trivially holds, as

$$f(S_0) = 0 \geq (1 - 1) f(S^*).$$

Assume now that Eq. (1) holds for $i \in \{0, 1, \ldots, k-1\}$, we seek to show the equation holds for $i + 1$. Notice that

$$\begin{aligned}
f(S_{i+1}) - f(S_i) &\geq \frac{1}{k} \sum_{j=1}^{k} f(S_i \cup \{x_j\}) - f(S_i) && \text{(by definition of } e_{i+1}) \\
&\geq \frac{1}{k} \sum_{j=1}^{k} f(S_i \cup \{x_1, \ldots, x_j\}) - f(S_i \cup \{x_1, \ldots, x_{j-1}\}) && (f \text{ is submodular}) \\
&= \frac{1}{k} (f(S_i \cup S^*) - f(S_i)) && \text{(telescoping sum)} \\
&\geq \frac{1}{k} (f(S^*) - f(S_i)) && \text{(by monotonicity).}
\end{aligned}$$

Therefore,

$$\begin{aligned} f(S_{i+1}) &\geq \left(1-\frac{1}{k}\right) f(S_i) + \frac{1}{k} f(S^*) \\ &\geq \left(1-\frac{1}{k}\right)\left(1-\left(1-\frac{1}{k}\right)^i\right) f(S^*) + \frac{1}{k} f(S^*) \qquad \text{(induction hypothesis)} \\ &= \left(1-\left(1-\frac{1}{k}\right)^{i+1}\right) f(S^*). \end{aligned}$$

□

Thus, the S_k found by the greedy algorithm has the property that

$$f(S_k) \geq \left(1-\left(1-\frac{1}{k}\right)^k\right) f(S^*) \geq \left(1-\frac{1}{e}\right) f(S^*),$$

establishing that we always get a $1-1/e$ approximation. Can we hope to do better? As will be the theme of this class: nope!

4.4 Hardness via Symmetry Gap

In order to state the hardness result, we need to define some important concepts.

Given $y \in [0,1]^n$, we let $S \sim y$ denote the distribution of sets $S \subset [n]$ such that for all $i \in [n]$, we have $i \in S$ with probability y_i, independent for each i.

Definition 4.4 (Multi-linear relaxation)**.** Given $f : 2^{[n]} \to \mathbb{R}_+$, we define the *multi-linear relaxation* $F : [0,1]^n \to \mathbb{R}_+$ to be

$$F(y) := \mathop{\mathbb{E}}_{S \sim y}[f(S)].$$

We say that $f : 2^{[n]} \to \mathbb{R}_+$ is *symmetric* if for all $S, T \subset [n]$, if $|S| = |T|$, then $f(S) = f(T)$. That is, replacing

For any $x \in [0,1]^n$, we define $\bar{x} \in [0,1]^n$ to be such that $\bar{x}_i = \dfrac{1}{n}\sum_{j \in [n]} x_j$ for all $i \in [n]$.

Definition 4.5 (Symmetry Gap)**.** For a symmetric submodular function $f : 2^{[n]} \to \mathbb{R}_+$ and a symmetric constraint $C \subset \{0,1\}^n$, we define

$$\gamma(f, C) = \frac{\max_{x \in C} F(\bar{x})}{\max_{x \in C} F(x)}.$$

The symmetry gap was introduced by Vondrak [6] to show that it characterizes the complexity of many submodular optimization problems.

Theorem 4.6 (Informal version of Theorem 1.6 of [6])**.** *For a "natural" a family of submodular optimization problems, let γ be the minimum $\gamma(f, C)$ for all submodular functions $f : 2^{[n]} \to \mathbb{R}_+$ and constraints $C \subset \{0,1\}^n$ in this family. Then, the constrained maximization problem for f cannot be done in $2^{o(n)}$ queries.*

This theorem will be proved in the next lecture.

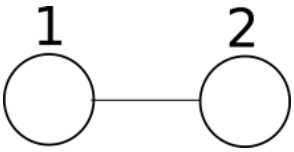

Figure 2: Graph whose cut function has optimal symmetry gap for unconstrained maximization.

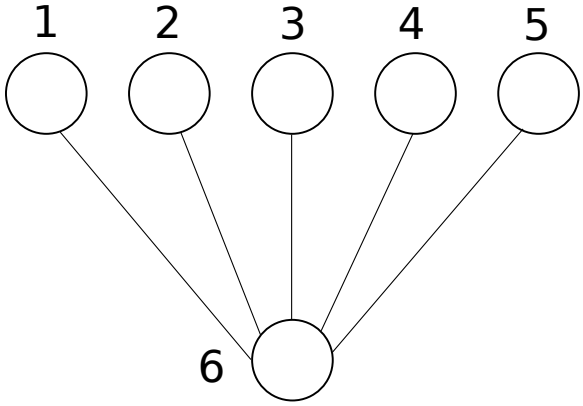

Figure 3: Graph whose cover function has optimal symmetry gap for constrained monotone maximization.

4.5 Examples of Symmetry Gap

We conclude that the examples introduced in Section 4.2 have tight lower bounds.

Unconstrained maximization. Consider the graph on two vertices connected by a single edge (see Figure 2). Let $f : 2^{[2]} \to \mathbb{R}_+$ be its cut function and $F : [0,1]^2 \to \mathbb{R}_+$ be its multi-linear extension. The maximum cut on this graph is $F(1,0) = 1$. For symmetrized inputs, the maximum value of F occurs at $(1/2, 1/2)$:

$$\begin{aligned} F(\tfrac{1}{2},\tfrac{1}{2}) &= \frac{1}{4}F(0,0) + \frac{1}{4}F(0,1) + \frac{1}{4}F(1,0) + \frac{1}{4}F(1,1) \\ &= \frac{1}{4}\cdot 0 + \frac{1}{4}\cdot 1 + \frac{1}{4}\cdot 1 + \frac{1}{4}\cdot 0 \\ &= \frac{1}{2}. \end{aligned}$$

Thus, the symmetry gap is $1/2$, showing that the stated $1/2$ approximation is optimal.

Cardinality constrained monotone maximization. Let $G = (V, E)$ be a graph on $n+1$ vertices with vertices $1, \ldots, n$ all connected to $n+1$ (see Figure 3). Let $f : 2^{[n]} \to \mathbb{R}_+$ be the coverage function of G restricted to subsets of $[n]$; that is, we do not allow $n+1$ in our covering set. Consider the cardinality constraint problem with $k = 1$. Then, the maximum value of f is clearly 1. To compute the symmetry gap, we compare this to $F(\frac{1}{n}, \ldots, \frac{1}{n})$. Since f is equal to 1 everywhere except at the empty set (whose value is 0), the value of F is equal to the probability of not picking

the empty set when sampling according to $(\frac{1}{n}, \ldots, \frac{1}{n})$:

$$F(\tfrac{1}{n}, \ldots, \tfrac{1}{n}) = 1 - \left(1 - \frac{1}{n}\right)^n.$$

Taking the limit as $n \to \infty$, shows that the symmetry gap at most $1 - \frac{1}{e}$, implying that the greedy algorithm is indeed optimal.

References

[1] Sébastien Bubeck. Convex Optimization: Algorithms and Complexity. May 2014.

[2] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015.

[3] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic Submodular Function Minimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1220–1231, New York, NY, USA, 2017. ACM.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

[5] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, December 1978.

[6] J. Vondrák. Symmetry and Approximability of Submodular Maximization Problems. *SIAM Journal on Computing*, 42(1):265–304, January 2013.