CS 354: Unfulfilled Algorithmic Fantasies April 3, 2019

Lecture 2: Symmetry Gap Theorem & Hardness of Finding Fixed Point

Lecturer: Aviad Rubinstein *Scribe: Junyao Zhao*

1 Symmetry Gap Theorem for Submodular Optimization

1.1 Recap from last lecture

In the last lecture, we stated a powerful theorem which provides a lower bound for the query complexity of maximizing submodular functions. Recall the following basic definitions.

Definition 1.1 (Submodular function)**.** A function $f : 2^{[n]} \to \mathbb{R}_+$ is submodular if for all $S \subseteq T \subseteq [n]$ and $i \in [n] \setminus T$, $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$.

Definition 1.2 (Multi-linear relaxation)**.** Given a function $f : 2^{[n]} \to \mathbb{R}_+$, we define the multi-linear relaxation $F : [0,1]^n \to \mathbb{R}_+$ to be $F(x) = \mathop{\mathbb{E}}_{S \sim x}[f(S)]$, where $i \in S$ with probability x_i.

We say that f is symmetric if $f(S) = f(T)$ for all $S, T \subseteq 2^{[n]}$ of the same cardinality. Similarly, a constraint $\mathcal{C} \subseteq 2^{[n]}$ is symmetric if for all $S \in \mathcal{C}$, any $T \subseteq 2^{[n]}$ with $|T| = |S|$ is also an element in $\mathcal{C}$. (In general, one can define the symmetry with respect to any symmetry group, but we use this specific definition for simplicity.) Furthermore, given $x \in [0,1]^n$, we define its symmetrization $\bar{x}$ to be such that $\bar{x}_i = \sum_{j \in [n]} x_j / n$ for all $i \in [n]$.

Definition 1.3 (Symmetry gap)**.** For a symmetric submodular function $f : 2^{[n]} \to \mathbb{R}_+$ and a symmetric constraint $\mathcal{C} \subseteq 2^{[n]}$, the symmetry gap is defined to be $\gamma(f, \mathcal{C}) = \max_{x \in \mathcal{C}} F(\bar{x}) / \max_{x \in \mathcal{C}} F(x)$.

As a running example, we let f be the cut function of K_2 (Figure 1) and $\mathcal{C}$ is simply the $2^{[2]}$, *i.e.*, unconstrained max-cut. It is easy to check that $\gamma(f, \mathcal{C}) = 1/2$.

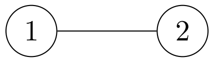

Figure 1: Graph whose cut function has unconstrained symmetry gap 1/2.

The symmetry gap was introduced by Vondrák [4] to show the following theorem, which characterizes the query complexity of many submodular maximization problems.

Theorem 1.4 (Informal version of Theorem 1.6 of [4])**.** *There is a "natural" family of submodular maximization problems, such that for any problem $\mathcal{P}$ in this family, suppose γ is the minimum $\gamma(f, \mathcal{C})$ for all instances $(f, \mathcal{C}) \in \mathcal{P}$, then there exists an instance $(f^*, \mathcal{C}^*)$ of $\mathcal{P}$ with $f^* : 2^{[n]} \to \mathbb{R}_+$, which requires $2^{\Omega(n)}$ queries of f^* to find a $(1+\epsilon)\gamma$-approximate solution, for any $\epsilon > 0$.*

1.2 Proof of the Symmetry Gap Theorem

Proof sketch. The idea of the proof is that for any algorithm $\mathcal{A}$, given an instance $(f_0, \mathcal{C}_0)$ of symmetry gap γ and the multi-linear relaxation F_0, we can "grow" this instance into a larger instance $(f, \mathcal{C})$. Specifically, we use random permutations to "symmetrize" the input to f, feed the symmetrized input to F_0, and then take the output of F_0 as the output of f. By this random construction of f, with high probability, the first query to f in $\mathcal{A}$ turns into a query, that is close to some symmetric $\bar{x}$, to F_0. Hence, the first query returns a poor value of F_0 at $\bar{x}$ because of the symmetry gap of f_0. $F_0(\bar{x})$ does not depend on the random permutations we used, so it gives no information about these permutations. Hence, f still looks like the same random construction to $\mathcal{A}$ after its first query, so by induction, $\mathcal{A}$ keeps getting stuck in the symmetry during its further queries, and the symmetry gap serves as a lower bound of the approximation guarantee of $\mathcal{A}$.

However, this argument has two fallacies. First, we did not explain how to define $\mathcal{C}$, but we want it to be of the same constraint type as $\mathcal{C}_0$. There is a natural operation which "grows" the instance and preserves a wide array of constraint types (Definition 1.5 of [4]). We will not discuss this operation for simplicity, and one may assume that we are dealing with unconstrained submodular maximization. Second, the definition of the symmetry gap does not take care of those vectors that are close to $\bar{x}$. We will "smooth" the multi-linear relaxation F_0 of f_0, so that the neighborhood near $\bar{x}$ has the same value as $F(\bar{x})$, and we need to ensure that the function is still submodular and optionally monotone. Now, we sketch the proof in three steps with some more details.

Step 1: "Blow up" the instance. Given a submodular function $f_0 : 2^{[k]} \to \mathbb{R}_+$, we construct $f : 2^{[n]\times[k]} \to \mathbb{R}_+$ as follows. For all $i \in [n]$, we pick a random permutation $\pi_i : [k] \to [k]$, and we define f to be

$$f(z_{1,1}, \ldots, z_{1,k}, \ldots, z_{n,1}, \ldots, z_{n,k}) = F_0\left(\frac{1}{n}\sum_{i\in[n]} z_{i,\pi_i(1)}, \ldots, \frac{1}{n}\sum_{i\in[n]} z_{i,\pi_i(k)}\right). \tag{1}$$

The construction of f for the case where f_0 is the cut function of K_2 and $n = 3$ is illustrated in Figure 2. We create 3 copies of the vertex set of K_2, and for each copy i, we permute the two vertices using π_i. Then we take the fraction of the nonzero vertices on each side as the input to F_0.

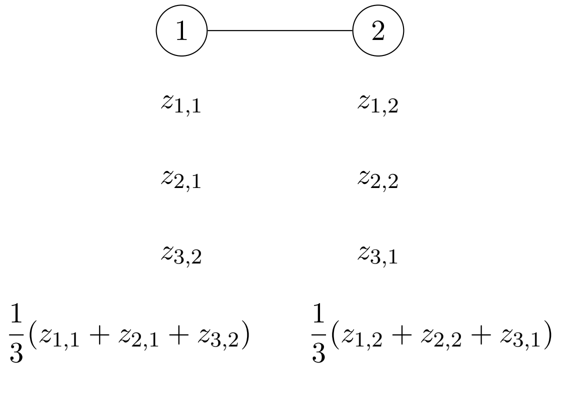
1
2
$z_{1,1}$
$z_{1,2}$
$z_{2,1}$
$z_{2,2}$
$z_{3,2}$
$z_{3,1}$
$\frac{1}{3}(z_{1,1} + z_{2,1} + z_{3,2})$
$\frac{1}{3}(z_{1,2} + z_{2,2} + z_{3,1})$

Figure 2: Illustration of how to "blow up" the cut function of K_2.

Intuitively, it is not hard to see that under random permutations, it holds with high probability that

$$\frac{1}{n}\sum_{i\in[n]} z_{i,\pi_i(1)} \approx \frac{1}{n}\sum_{i\in[n]} z_{i,\pi_i(2)} \approx \cdots \approx \frac{1}{n}\sum_{i\in[n]} z_{i,\pi_i(k)}, \tag{2}$$

i.e., the input to F_0 is close to be symmetric. If the input to F_0 is exactly symmetric, it will result into poor output value, due to the symmetry gap. In step 2, we will smooth F_0 so that those nearly symmetric vectors also have poor values. But before that, we have not yet proved f is submodular or monotone (optionally) if f_0 is. This follows from Lemma 3.1 in [4] and the observation that the π_i's are simply a relabeling of the elements in $[n] \times [k]$.

Step 2: "Smooth" the multi-linear relaxation. Given F_0 as before, we want to construct a function $H : [0,1]^k \to \mathbb{R}_+$ that satisfies the following conditions.

- If $\|x - \bar{x}\|_2 \le \delta$, then $H(x) = F_0(\bar{x})$.
- For x with large $\|x - \bar{x}\|_2$, $H(x) \approx F_0(x)$.
- $h : 2^{[k]} \to \mathbb{R}_+$ with $h(z) = H(z)$ is submodular and optionally monotone.

H is given as follows

$$H(x) = \phi(\|x - \bar{x}\|_2) \cdot F_0(\bar{x}) + (1 - \phi(\|x - \bar{x}\|_2)) \cdot F_0(x). \tag{3}$$

Instead of defining the function ϕ, we illustrate its plot in Figure 3. Observe that $\phi(x) = 1$ when $x \le \delta$. Hence the first condition is satisfied. When x is sufficiently large, $\phi(x)$ is close to 0, and hence the second condition is also satisfied.

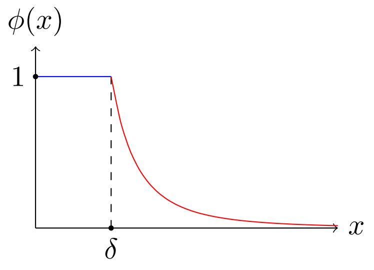

Figure 3: Plot of $\phi(x)$.

The third condition is actually not satisfied by this construction. However, it turns out that H has well-bounded first and second partial derivatives, and therefore, we can add a regularizer to H, to make h submodular and optionally monotone while preserving the desired properties of H. Now we can replace F_0 with H in the step 1.

Step 3: Complete the proof. Assume that after the first t queries, the algorithm did not learn anything about the permutations. By Equation 2, the step 1 guarantees that with high probability the first query z of $\mathcal{A}$ to f turns into a nearly symmetric input x to H, in the sense that $\|x - \bar{x}\|_2 \le \delta$. Hence the result of the first query $H(x) = F_0(\bar{x})$ is a bad approximation because of the symmetry gap of F_0. Moreover, it is straightforward to see that $\bar{x} = \bar{z}$, hence the query result

$F_0(\bar{x})$ does not depend on the π_i's we used in the step 1. Therefore, after the first $t+1$ queries, with high probability, f is still looks like a random construction that can fool $\mathcal{A}$. The proof finishes by induction on t. $\square$

2 Query complexity of finding fixed point

Brouwer's fixed point theorem is well-known in topology and very useful in economics. However, in order to approximately find the fixed point, exponential number of queries are necessary.

Theorem 2.1 (Brouwer's fixed point theorem). *If $f : [0,1]^n \to [0,1]^n$ is continuous and Lipschitz, then there exists some $x^* \in [0,1]^n$ such that $f(x^*) = x^*$.*

Theorem 2.2 (Query complexity of finding fixed point [2, 3]). *There exists $\epsilon > 0$, such that finding x that satisfies $\|f(x) - x\|_2^2 < \epsilon n$ requires $2^{\Omega(n)}$ queries.*

One can think of the condition $\|f(x) - x\|_2^2 < \epsilon n$ as ϵ-fraction of their coordinates are close.

Proof sketch of Theorem 2.2. We first introduce the end-of-a-line (EoaL) problem and its query complexity. Then, we reduce an instance of the EoaL to an instance of finding the fixed point.

Step 1: End-of-a-line.
Input:

- Base graph $G = (V, E)$.
- Successor function $S : V \to V$.
- Predecessor function $P : V \to V$.
- Start vertex $v_0 = P(v_0) \neq S(v_0)$.

Output: End of a line $y = S(y) \neq P(y)$.

Figure 4 shows an instance of the EoaL on the $N \times N$ directed-monotone grid (with self-loops). It is not hard to see that the query complexity of this problem is $\Omega(N)$. Intuitively, this is because any algorithm can only try to find a "random" point on the line, or sequentially follow the line.

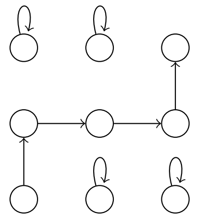

Figure 4: An end-of-a-line instance on an $N \times N$ grid ($N = 3$).

Step 2: EoaL $\longrightarrow$ Brouwer.
Goal: Construct f such that the following conditions are satisfied.

- f is continous and Lipschitz.

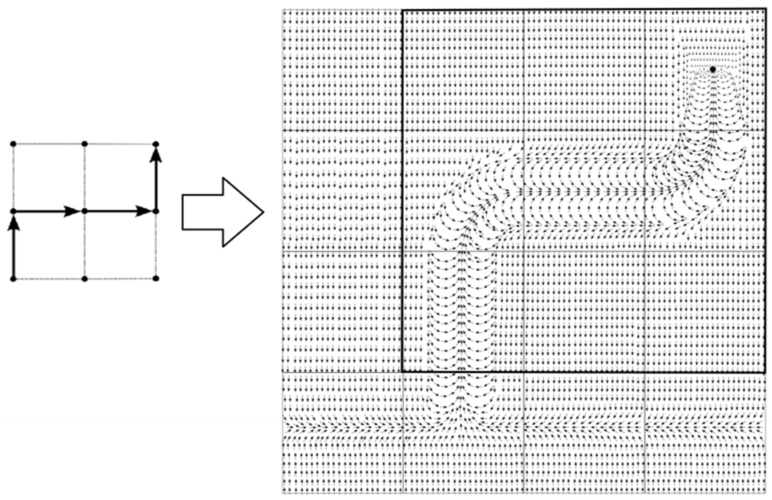

Figure 5: A 2-dimensional example of "EoaL $\longrightarrow$ Brouwer" (figure from [1]).

- The fixed point of f is corresponding to the end of a line.

Now we construct such f from an EoaL instance, by designing a vector field which characterizes the f. That is, for an arbitrary point x, $f(x)$ is defined to be the endpoint of the arrow attached to x. We will embed the EoaL instance into this vector field, and it follows that the end of the line is a fixed point. Figure 5 illustrates the construction in the 2-dimensional case.

Attempt 1: 2-dimensional construction

- On the line, the arrows follow the direction of the line.
- Close to the line, the arrows go towards the line.
- At mid-distance to the line, the arrows follows the opposite direction of the line.
- Far away from the line, the arrows go towards the start vertex of the line (default).

Notice that for any point at the border of any two consecutive layers, the two different arrows at this point, corresponding to the two layers, have a non-straight (sometimes orthogonal) angle. Hence there is no bad fixed point anywhere else in the vector field, besides the end of the line. Moreover, we can interpolate the arrows at the borders to make f Lipschitz. Therefore, the query complexity lower bound of the EoaL turns into the query complexity lower bound of finding the fixed point.

Attempt 2: n-dimensional construction

However, we are not done yet, because we only showed how to construct f in the 2-dimension, but we actually want an n-dimensional f. In particular, in constant dimensions, only finding exponentially-good approximations can be hard.

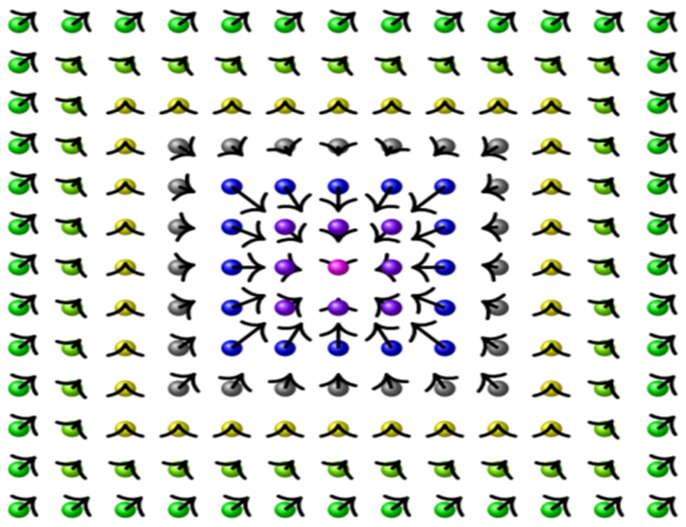

Figure 6: A facet of n-dimensional example of "EoaL $\longrightarrow$ Brouwer" (path going into the paper).

The idea is that we can embed the End-of-a-Line instance in an n-dimensional vector field, following the same principles we used for the 2-dimensions. Figure 6 illustrates this approach. Imagine that Figure 6 is a facet of the n-dimensional vector field. At the center point, we let the lower-dimensional vector sub-field, that is orthogonal to this facet, embed the lower-dimensional finding-fixed-point instance that we constructed from the EoaL. Furthermore, we arrange the directions of the arrows according to the distance to the center lower-dimensional vector sub-field on this facet, such that there is no extra bad fixed point.

Final construction: $O(n)$ dimensions

[We did not cover this construction in class - it is included here just FYI.]

Our n-dimensional construction suffices to show that it is hard to find x such that $\|f(x)-x\|_\infty^2 < \epsilon$. I.e. $f(x)$ is far from x on at least one dimension. In particular, most arrows in Figure 6 only point in one dimension. To get the stronger statement that $f(x)$ is far from x on a *constant fraction* of dimensions (i.e. $\|f(x)-x\|_2^2 < \epsilon n$ as in our theorem statement) we modify our construction. We again follow the same principles for embedding the End-of-a-Line instance, but we use *error correcting codes* to ensure that the embedding of any two vertices differs on a constant fraction of the coordinates. Unfortunately, we don't have another figure because we can't really plot vector fields in $O(n)$ dimensions, even for $n = 1$...

$\square$

References

[1] Yakov Babichenko. Query complexity of approximate nash equilibria. *J. ACM*, 63(4):36:1–36:24, 2016.

[2] Michael D Hirsch, Christos H Papadimitriou, and Stephen A Vavasis. Exponential lower bounds for finding brouwer fix points. *Journal of Complexity*, 5(4):379–416, 1989.

[3] Aviad Rubinstein. Settling the complexity of computing approximate two-player nash equilibria. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 258–265. IEEE, 2016.

[4] Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.