1 Overview

Last time we saw the problem of Combinatorial Auctions and framed it as a submodular maximization problem under a partition constraint. We showed that a greedy algorithm is a 1/2-approximation algorithm for this problem.

In this lecture we will present other applications of submodular maximization. In particular we will study the Max-Cut problem which is an example of non-monotone submodular maximization.

2 Applications of submodular maximization

Diversity maximization.

- **Input.** Set of documents on multiple topics.
- **Goal.** Find K documents that cover as many topics as possible.

Example. For a query like *Obama* to a search engine, the search engine computes a list of documents which match this query (for example containing the query term). Furthermore, the search engine knows a list of topics associated with each document. The same topic can be associated to multiple documents.

This problem is a direct application of the Max-Cover problem where each document is a set covering topics.

Clustering. We are given a set of points. For any pair of point (x, y), we know the distance $d(x, y)$ between x and y. For a point x and a set of points T, we define $d(x, T) = \min_{y \in T} d(x, y)$.

- **Input.** Set of points X
- **Goal.** find $S \in \text{argmax}_{|S| \leq k} \sum_{x \in X} d(x, S)$

The intuition is that each point $x \in X$ is associated to its closest point in S and the cost of point x is the distance to this closest point. In effect, this clusters X in k parts: for each $y \in S$ there is a part consisting of all the points associated with y.

As it is, this is not a submodular problem. We can however turn it into a submodular optimization problem. First add a point x_0 such that $d(x_0, x) \geq d(y, x)$ for any x, y (think of x_0 as being very far away). Then define:

$$f(S) = \sum_{x \in X} (d(x, \{x_0\}) - d(x, S \cup \{x_0\}))$$

Entropy and Information Gain. For a random variable X, we define the *entropy* of X:

$$H(X) = \sum_x \mathbb{P}[X = x] \log \frac{1}{\log \mathbb{P}[X = x]}$$

For two random variables we have:

$$H(X, Y) = \sum_{x,y} \mathbb{P}[X = x, Y = y] \log \frac{1}{\log \mathbb{P}[X = x, Y = y]}$$

And the conditional entropy:

$$\begin{aligned} H(Y|X) &= \sum_x \mathbb{P}[X = x] H(Y|X = x) \\ &= \sum_x \mathbb{P}[X = x] \left(\sum_y \mathbb{P}[Y = y|X = x] \log \frac{1}{\mathbb{P}[Y = y|X = x]} \right) \end{aligned}$$

A well-known principle in information theory is the *Information never hurts principle*: for any two variables, $H(Y|X) \leq H(Y)$. This principle leads to subdmodular functions in many information-theoretic problems (for example feature selection).

3 The Max-Cut problem

Definition 1. *Given an undirected graph $G = (V, E)$, the **cut induced by** $S \subseteq V$ in G is the set of edges "leaving" S:*

$$C(S) = |\{(u, v) \,|\, u \in S, v \notin S, (u, v) \in E\}|$$

The Max-Cut problem is then the following:

- **Input.** Undirected graph $G = (V, E)$.
- **Goal.** Find S, such that $C(S)$ is maximal.

Max-Cut is known to be NP-hard so we will focus on designing approximation algorithms.

3.1 Local search algorithm

Consider the algorithm described in Algorithm 1. An illustration of this algorithm is given in Figure 1.

It is not clear a priori that this algorithm terminates because the algorithm could keep finding improvements forever. This is in contrast to the greedy algorithm where the budget was limiting the number of steps. However this is not the case as we now prove.

Algorithm 1 Local search algorithm for Max-Cut

1: $S \leftarrow$ arbitrary set of nodes
2: **if** $\exists v \in S,\ c(S \setminus v) > C(S)$ **then**
3: $\quad S \leftarrow S \setminus v$
4: **end if**
5: **if** $\exists v \notin S,\ c(S \cup v) > C(S)$ **then**
6: $\quad S \leftarrow S \cup v$
7: **end if**
8: repeat lines 2 to 7 until there is no improvement.
9: **return** S

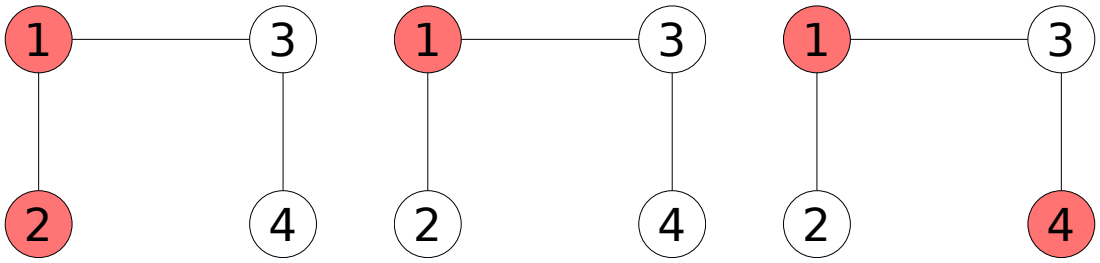

Figure 1: An illustration of the local search algorithm for Max-Cut. The algorithm starts with set $S = \{1, 2\}$ which has value $C(S) = 1$. Then removing 2 improves the value $C(S)$ to 2. Finally adding 4 improves the value to $C(S) = 3$ which is optimal for this graph.

Proposition 2. *Algorithm 1 terminates in at most $m = |E|$ steps.*

Proof. At every step of this algorithm the value of $C(S)$ either increases by one or not at all. When $C(S)$ no longer increases, this means that the algorithm cannot find an improvement and terminates. Since $|E|$ is an obvious upper-bound on the optimal value of $C(S)$ this implies that the number of steps is at most m. □

Theorem 3. *Algorithm 1 is a $1/2$-approximation algorithm for the Max-Cut problem.*

Proof. Let us define $e(v, T)$ the number of edges from $v \in V$ to $T \subseteq V$:

$$e(v, T) = |\{(v, u) \,|\, u \in T, (u, v) \in E\}|$$

Using this definition, we can rewrite $C(S)$:

$$C(S) = \sum_{v \in S} e(v, V \setminus S)$$

we could equivalently write:

$$C(S) = \sum_{u \notin S} e(u, S)$$

or even better, we can combine both expressions:

$$C(S) = \frac{1}{2} \left(\sum_{u \notin S} e(u, S) + \sum_{v \in S} e(v, V \setminus S) \right)$$

Let us write $d(v)$ the degree of $v \in V$. The first observation is that:

$$\forall v \in S,\ e(v, V \setminus S) \geq \frac{d(v)}{2}$$

Indeed, if this was not the case, removing v from S would increase the value of the cut. Similarly we have:

$$\forall u \notin S,\ e(u, S) \geq \frac{d(u)}{2}$$

otherwise adding u to S would increase the value of the cut. Plugging this in the above expression gives:

$$\begin{aligned} C(S) &\geq \frac{1}{2}\left(\sum_{u \notin S} \frac{d(u)}{2} + \sum_{v \in S} \frac{d(v)}{2}\right) \\ &= \frac{1}{4}\sum_{v \in V} d(v) = \frac{2m}{4} = \frac{m}{2} \end{aligned}$$

We can now conclude using that the optimal cut has value upper-bounded by $|E| \leq m$. □

Connection to coordinate descent. Remember the coordinate descent algorithm for convex optimization.

Algorithm 2 Coordinate descent algorithm

1: Guess $\mathbf{x}^{(0)}$, $k \leftarrow 0$
2: **while** $\|\nabla f(\mathbf{x}^{(k)})\| \geq \varepsilon$ **do**
3: $\quad i \leftarrow \text{argmax}_j \left|\frac{\partial f(\mathbf{x}^{(k)})}{\partial \mathbf{x}_j}\right|$
4: $\quad \mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + t_k.\text{sign}(e_i)$
5: $\quad k \leftarrow k + 1$
6: **end while**
7: **return** $\mathbf{x}^{(k)}$

In this algorithm, $\text{sign}(e_i)$ is $+e_i$ if $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_i} > 0$ and $-e_i$ otherwise.

If we encode a solution S to the Max-Cut problem as a binary vector $\mathbf{x} \in \{0,1\}^n$, then we can rewrite the local search algorithm

In this form, the analogy to coordinate descent is clear: the direction of improvement (adding or removing) plays the same role as the sign of the partial derivative in coordinate descent.

3.2 Randomized Algorithm

We will now consider a simple randomized algorithm.

Theorem 4. *Let S be the set returned by Algorithm 4. Then $\mathbb{E}[C(S)] = \frac{m}{2}$.*

This theorem again implies that Algorithm 4 is a 1/2-approximation algorithm.

Algorithm 3 Local search algorithm for Max-Cut, second version

```
Guess x
while there is an improvement do
    consider coordinate i
    if adding i to S improves C(S) then
        x ← x + e_i
    else if removing i from S improves C(S) then
        x ← x − e_i
    end if
end while
return x
```

Algorithm 4 Randomized algorithm for Max-Cut

```
S ← ∅
for u ∈ S do
    add u to S with probability 1/2
end for
return S
```

Proof. Define for each edge $e \in E$ the following random variable:

$$X_e = \begin{cases} 1 & \text{if } e \text{ is a cut edge} \\ 0 & \text{otherwise} \end{cases}$$

We have $\mathbb{P}[X_e = 1] = \frac{1}{2}$. Indeed, writing $e = (u, v)$, e is an edge cut if $u \in S$ and $v \notin S$ which occurs with probability $\frac{1}{4}$ or if $u \notin S$ and $v \in S$ which also occurs with probability $\frac{1}{4}$. By linearity of expectation:

$$\mathbb{E}[C(S)] = \mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e] = \sum_{e \in E} \mathbb{P}[X_e = 1] = \frac{m}{2} \qquad \square$$

3.3 Generalization to non-monotone submodular maximization

Max-Cut is in fact an example of non-monotone submodular maximization problem. Recall that a function f is said to be *monotone* iff $S \subseteq T \Rightarrow f(S) \leq f(T)$.

Definition 5. *A function f is called **symmetric** if $f(S) = f(N \setminus S)$.*

Observe that the cut function in the Max-Cut problem is symmetric.

Theorem 6. *For any non-monotone subdmodular function, the algorithm which selects each element independently with probability $1/2$ is a $1/4$ approximation algorithm in expectation. If the function is symmetric then the approximation ratio is $1/2$.*

Historical note. It was believed for a long time that $\frac{1}{2}$ was the best achievable ratio for symmetric submodular functions until the famous result by Goemans and Williamson in 1994 which gives an SDP-based algorithm for Max-Cut achieving an approximation ratio of 0.878.