Homework NUMBER 6
Colorado CSCI 5454

Rithik Kumar Athiganur Senthil

October 5, 2022

People I studied with for this homework: Ashwin

Other external resources used:

- Lecture notes and videos

,

Problem 1 (4 points)

The *Weighted Max Cut* problem is as follows:

- **Instance**: an undirected weighted graph $G = (V, E)$ with positive edge weights $w_{u,v}$.
- **Feasible solution**: subset of items with total weight $\leq$ W .
- **Objective**: maximize the total weight crossing the cut, i.e. if g(u) $\in$ A,B is the assignment, then maximize $\sum \{w_{u,v}\} : \{u, v\} \in E, g(u) \neq$ g(v).

Give a randomized $\frac{1}{2}$-approximation algorithm and carefully prove its approximation factor.

Solution:

Let's take our instance as a graph $G = (V, E)$, lets take a randomized algorithm such that a vertex u $\in$ V can be either in set A or B with equal probability of $\frac{1}{2}$.
And the cut will pass through A and B.
Let's take a function $g(x)$ to check if a vertex is present in A or B.
Let X indicate if an edge would cross the cut, and if it did, it would be equal to the edge's weight $\{u, v\}$ $\in$ E. It is not possible for g(u) to equal g(v) or else X would be zero.

Let the randomized algorithm for edges {u,v} ∈ E be,

$$randAlgo = \sum_{\{u,v\}\in E} X_{u,v}$$

Let's analyze the performance of our randomized algorithm by,

$$\begin{aligned}\mathbb{E}\,[randAlgo] &= \mathbb{E}\,[\sum_{\{u,v\}\in E} X_{u,v}] \\ &= \sum_{\{u,v\}\in E} \mathbb{E}\,[X_{u,v}] \\ &= \sum_{\{u,v\}\in E} (0 * P[X_{u,v}=0]) + (w_{u,v} * P[X_{u,v}=w_{u,v}])\end{aligned}$$

By analyzing the above equation we can see that the probability of a vertex V being present in either A or B id $\frac{1}{2}$.
Let's check the probability of an edge {u,v} ∈ V be present in different sets.

i.e, (1 - (probability of u is in A)*(probability of v is in A) + (probability of u is in B)*(probability of v is in B))

$$=1 - ((\tfrac{1}{2} * \tfrac{1}{2}) + (\tfrac{1}{2} * \tfrac{1}{2})) = \tfrac{1}{2}$$

With the information provided above, we can infer that 50 percent of the graph's edges would cross the cut. Using this, we determine the expectation from the calculation above.

Let's take the following conventions:

W_e be the sum of weight of all the edges.
W_{opt} be the total weight crossing the cut in the optimal case.

$$\begin{aligned}\mathbb{E}\,[randAlgo] &= \sum_{\{u,v\}\in E} (0 * P[X_{u,v}=0]) + (w_{u,v} * P[X_{u,v}=w_{u,v}]) \\ &= \sum_{\{u,v\}\in E} (0 * \frac{1}{2}) + (w_{u,v} * \frac{1}{2}) \\ &= \frac{1}{2}|W_e|\end{aligned}$$

$$\begin{aligned} W_{opt} &\leq W_e \\ \frac{1}{2}W_{opt} &\leq \frac{1}{2}|W_e| \\ \frac{1}{2}W_{opt} &\leq randAlgo \end{aligned}$$

Hence, this gives the approximation factor of the randomized algorithm as $(\frac{1}{2})$.

Problem 2 (4 points)

The *Max-k-Partition* problem is identical to Max Cut, but we must partition the vertices of a graph into k sets $S_1, ..., S_k$ so that we maximize the number of edges that cross between two different sets, i.e. edges {u, v} where u and v are not in the same set. Give a randomized $\frac{k-1}{k}$ approximation for this problem and carefully prove the approximation ratio.

Hint: define an indicator variable $X_{u,v}$ for the event that edge {u,v} crosses between two sets.

Solution:

Let's take our instance as a graph $G = (V, E)$, lets take a randomized algorithm such that a vertex u ∈ V can be either in set A or B with equal probability of $\frac{1}{2}$.
And the cut will pass through A and B.
Let's take a function $g(x)$ to check if a vertex is present in A or B.
Let X indicate if an edge would cross the cut, and if it did, it would be equal to the edge's weight $\{u, v\}$ ∈ E. It is not possible for g(u) to equal g(v) or else X would be zero.

Let the randomized algorithm for edges {u,v} ∈ E be,

$$randAlgo = \sum_{\{u,v\}\in E} X_{u,v}$$

Let's analyze the performance of our randomized algorithm by,

$$\begin{aligned} \mathbb{E}\left[randAlgo\right] &= \mathbb{E}\,[\sum_{\{u,v\}\in E} X_{u,v}] \\ &= \sum_{\{u,v\}\in E} \mathbb{E}\left[X_{u,v}\right] \\ &= \sum_{\{u,v\}\in E} (0 * P[X_{u,v} = 0]) + (w_{u,v} * P[X_{u,v} = w_{u,v}]) \end{aligned}$$

In problem 1 we have the probability of a vertex V being present in either A or B. In this K=2, similarly,

Prob[g(u) = g(v)] = $\frac{1}{k}$ for an edge {u,v} $\in$ V

Prob[g(u) $\neq$ g(v)] = 1 - $\frac{1}{k}$ = $\frac{k-1}{k}$ for an edge {u,v} $\in$ V.

Now let's calculate the expectation of our randomized algorithm.

$$\begin{aligned}
\mathbb{E}\left[randAlgo\right] &= \sum_{\{u,v\}\in E} (0 * Prob[X_{u,v} = 0]) + (1 * Prob[X_{u,v} = 1]) \\
&= \sum_{\{u,v\}\in E} (0 * \frac{1}{k}) + (1 * \frac{k-1}{k}) \\
&= \frac{k-1}{k}|E|
\end{aligned}$$

Let's take the following conventions:

E denotes the total number of edges crossing the k cuts.
E_{opt} be total no.of edges crossing the k cuts in optimal case.

$$\begin{aligned}
E_{opt} &\leq E \\
\frac{k-1}{k}E_{opt} &\leq \frac{k-1}{k}|W_e| \\
\frac{k-1}{k}E_{opt} &\leq randAlgo
\end{aligned}$$

Hence, this gives the approximation factor of the randomized algorithm as $\frac{k-1}{k}$.

Problem 3 (10 points)

In the *weighted vertex cover* problem, the input is an undirected graph G = (V,E) and a list of positive vertex weights: w_v for each v $\in$ V.
The goal is to output a vertex cover while minimizing *total weight*.

Recall that a *vertex cover* is a set of vertices, S $\subseteq$ V , such that for all edges (u,v) $\in$ E, at least one of u,v is in S. The *total weight* of a vertex cover is w(S) := $\sum_{v\in S} w_v$.

Part a (4 points)

Consider this algorithm: find a maximal matching M in the graph, then let S equal the set of all endpoints of edges in M.

Does this algorithm give a constant-factor approximation to the *weighted* vertex cover problem? If so, prove it (and give the factor). If not, prove why not (e.g. using counterexamples).

Solution:

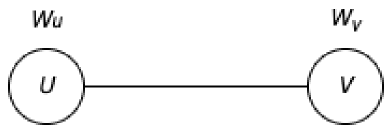

Figure 1: $G = (V, E)$

Let's take the above given graph $G = (V, E)$ as the example with weights of vertex U and V as W_u and W_v.
Now, the maximal matching on the given graph G has only the edge $\{U, V\}$ and the vertex cover would be $S_{alg} = [U, V]$.
now, if $W_U \leq$W$_V$.
then the approximation factor would be:

$$\left(\frac{|S_{alg}|}{|S_{Opt}|}\right) = \left(\frac{W_U + W_V}{W_U}\right)$$

Hence, the approximation factor would be 26 if $W_U = 1$ and $W_V = 25$ and would vary for edges U,V $\in$ E.
So we cannot have a constant approximation factor.

Part b (4 points)

Now consider this randomized algorithm: for each edge in the graph, if it is not yet covered, pick one of its endpoints uniformly at random and add it to S.
Does this algorithm give a constant-factor approximation? If so, prove it (and give the factor). If not, prove why not (e.g. using counterexamples).

Solution:
Let's take the same graph above $G = (V, E)$.
According to the algorithm,

probability of picking U is $(\frac{1}{2})$
probability of picking V is $(\frac{1}{2})$

. Let's compare the expectation of our randomized algorithm with the optimal solution.

$$
\begin{aligned}
randAlgo &= \sum_{\{u,v\}\in E} Prob(picking\ each\ vertex\ in\ set\ \{U,V\}) \\
\mathbb{E}\left[randAlgo\right] &= \sum_{u\in V} \mathbb{E}\left[Prob(picking\ each\ vertex\ in\ set\ \{U,V\})\right] \\
&= (\frac{1}{2}W_U + \frac{1}{2}W_V)
\end{aligned}
$$

Now, if $W_U \leq W_V$, the optimal vertex cover would be S_{Opt} =[U].
The approximation factor for our algorithm is:

$$
\begin{aligned}
(\frac{|S_{alg}|}{|S_{Opt}|}) &= (\frac{\frac{W_U+W_V}{2}}{W_U}) \\
&= (\frac{W_U+W_V}{2W_U})
\end{aligned}
$$

Hence, the approximation factor would be 13 if $W_u = 1$ and $W_v = 25$ and it could vary for edges u,v $\in$ E.
Since W_U and W_V is variable, we cannot have a constant approximation factor for our algorithm.

Part c (2 points)

Briefly compare your answers above to the randomized algorithm described in class. (For algorithms: which is better and why? For counterexamples: why does the algorithm from class succeed?)
Solution:
Let's take the same graph above in part (a) $G = (V, E)$.
According to the algorithm we have,

probability of picking U = $\frac{W_V}{W_U+W_V}$
probability of picking V = $\frac{W_U}{W_u+W_V}$

$$
\begin{aligned}
Alg &= \sum_{\{u,v\}\in E} P(picking\ each\ vertex\ in\ set\ \{u,v\}) \\
\mathbb{E}\left[Alg\right] &= \sum_{u\in V} \mathbb{E}\left[P(picking\ each\ vertex\ in\ set\ \{u,v\})\right] \\
&= (\frac{W_v}{W_u+W_v}W_u + \frac{W_u}{W_u+W_v}W_v) \\
&= (\frac{2W_uW_v}{W_u+W_v})
\end{aligned}
$$

Now, if $W_u \leq W_v$, the optimal vertex cover S_{Opt} =[u].
The approximation factor for the algorithm would be:

$$\begin{aligned}\frac{|S_{alg}|}{|S_{Opt}|} &= \frac{\frac{2W_U W_V}{W_U+W_V}}{W_U} \\ &= \frac{2W_V}{W_U + W_V} \leq 2 \\ |S_{alg}| &\leq 2|S_{Opt}|\end{aligned}$$

Here we can see that the approximation factor will be 2 at max(as the upper bound is 2).
Compared to the algorithms we analyzed in Part (a) and Part (b) they both have a variable approximation factor, the algorithm discussed in the class is better.
This technique is superior because we favor adding the smaller-weight vertex; for example, if W_V is significantly bigger than W_U, we are considerably more likely to add U.

Problem 4 (6 points)

Running Time of Karger's Min-Cut Algorithm. Recall that Karger's Min Cut algorithm succeeds with probability $\frac{2}{n(n1)} \geq \frac{2}{n^2}$.

The algorithm can be implemented in O(n^2) time on a graph with n vertices; let us take this as given because it involves some advanced data structures.

Part a (2 points)

Prove that if we repeat the algorithm n^2 times, for a total running time of O(n^4), the expected number of successes is at least 1.

Solution: Lets take a variable X as our success indicator.
with values 1 if our algorithm succeeded and 0 if our algorithm failed.

$$\text{X=0} \implies algorithm \;\; succeeded \tag{1}$$
$$\text{X=1} \implies algorithm \;\; failed \tag{2}$$

Value after n^2 iterations would be:

$$alg = \sum_{k=1}^{n^2} X_k$$

$$\begin{aligned}
\mathbb{E}\,[alg] &= \mathbb{E}\,[\sum_{k=1}^{n^2} X_k] \\
&= \sum_{k=1}^{n^2} \mathbb{E}\,[X_k] \\
&= \sum_{k=1}^{n^2} [0 * Prob(X_k = 0) + 1 * Prob(X_k = 1)] \\
&= \sum_{k=1}^{n^2} [0 * (1 - \frac{2}{n(n-1)}) + 1 * (\frac{2}{n(n-1)})] \\
&= \sum_{k=1}^{n^2} \frac{2}{n(n-1)} \\
&= n^2 * \frac{2}{n(n-1)} \\
&\geq n^2 * \frac{2}{n^2} \\
&\geq 2
\end{aligned}$$

Hence we can see that after n^2 iterations, we have the expected number of successes as atleast 1.

Part b (2 points)

Suppose we repeat the algorithm n_2 times and output the smallest cut found, for a total running time of O(n^4). Prove that the probability of success (i.e. outputting the min cut) is at least 1 - $\frac{1}{e^2}$.
Hint: To do so, prove that the probability of failure is at most $\frac{1}{e^2}$. Use that for all $x \in \mathbf{R}$, 1 + $x \leq e^x$
Solution:
For the given algorithm,

We know the following probabilities,

$$\begin{aligned}
\text{success is atleast } &\implies \tfrac{2}{n^2} \\
\text{failure is at most } &\implies 1 - \tfrac{2}{n^2}
\end{aligned}$$

probability of failures after n^2 iterations is,

$$\begin{aligned} Prob(failures\ after\ n^2\ iterations) &\leq (1 - \frac{2}{n^2})^{n^2} \\ &\leq (e^{\frac{-2}{n^2}})^{n^2} \\ &\leq (e^{\frac{-2n^2}{n^2}}) \\ &\leq \frac{1}{e^2} \end{aligned}$$

Prob(success after n^2 iterations) = 1 - Prob(failure after n^2 iterations) $\geq$ (1 - $\frac{1}{e^2}$).

hence, we have the probability of success after n^2 iterations is atleast (1 - $\frac{1}{e^2}$).

Part c (2 points)

Now suppose we want a success probability of 1 - δ, for some small number $\delta > 0$ such as δ = 0.00001. How many times should we repeat the algorithm before taking the smallest cut found? Prove your answer.
Solution:

Prob(success) = 1 - Prob(failure) = (1 - δ)
Prob(failure) $\leq$ (1 - $\frac{2}{n^2}$).
Solving for x we get,

$$(1 - \frac{2}{n^2})^x \geq \delta$$

$$(e^{-2x/n^2}) \geq \delta$$
$$\frac{-2x}{n^2} * \ln(e) \geq (\ln(\delta))$$

$$x \geq \frac{(-n^2) \cdot (\ln(\delta))}{2}$$

For δ = 0.00001,

$$\begin{aligned} x &\geq \frac{(-n^2) * (\ln(10^{-5}))}{2} \\ &\geq \frac{(-n^2) \cdot -5 \cdot (\ln(10))}{2} \\ &\geq \frac{(5n^2) \cdot (2).(3)}{2} \\ &\geq 5n^2 \end{aligned}$$

Hence we can conclude that the algorithm has to run for atleast $5n^2$ times in order to achieve $= 0.00001$.