# Data Mining - HW4

Nathan Geier
nathangeier@mail.tau.ac.il

1.

(a)

**Algorithm 1** $k$-mins sketch

**for** $1 \leq i \leq k$ **do**
    Iterate over nodes $u$ by increasing $h_i(u)$:
    Visit nodes $v$ reachable from $u$ (starting at $u$):
    **if** $s_v[i] = \emptyset$ **then**
        $s_v[i] \leftarrow s_v[i] \cup \{(h_i(u), u)\}$
        Continue the search in $outNeighbors(v)$
    **else**
        truncate search at $v$
    **end if**
**end for**

As learned in class, the complexity is $O(mk)$ since we processed each node at most 1 time in each iteration and there are $k$ iterations. (this is very similar to the algorithm we learned in class for min-hash with $k = 1$, just that now we do it for every hash function)

(b) This is a standard estimation using sketch problem, not related to graphs or how the data was generated (though we could try to use this information). As learned in lecture 2, we can merge the sketches of $Reach^{-1}(v) \cup Reach^{-1}(u)$ by taking the entry-wise min (per hash function) $s_{u,v}[i] \leftarrow \min\{s_v[i], s_u[i]\}$. After we have a $k$-mins sketch $s_{u,v}$, we use the MLE cardinality estimator for $k$-mins MinHash sktech:

$$\frac{k}{\sum_{i=1}^{k} s_{u,v}[i]}$$

where $h_i(x) \sim Exp(1)$. (As learned in lec. 2, it's easy to convert $h_i(x) \sim U[0,1]$ to $h_i(x) \sim Exp(1)$ so this isn't a problem)

2.

(a) Our estimator shall be

$$\widehat{C}_{\alpha,S}(v) = \sum_{u \in S} \widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])$$

Where $\widehat{I}_{v \rightsquigarrow u}$ is an "inverse probability" unbiased estimator of the "presence" of $u$ in $ADS(v)$, and $ADS(v)[u]$ is the distance $d_{vu}$ saved in $ADS(v)$, but if $u \notin ADS(v)$ we shall define it to be $+\infty$ so that $\alpha(ADS(v)[u]) = 0$. For the estimator $\widehat{I}_{v \rightsquigarrow u}$ we will use the "inverse probability": when $u \notin ADS(v)$ it will be 0, and otherwise it will be the inverse $(1/)$ of the probability for $h(u)$ having a "good" value that would let $u$ in, when fixing $h$ on all the other vertices. Namely, we look at all other vertices in $ADS(v)$, fix their hash values, and ask for what values of $h(u)$ we have that $u$ is included, and what is the probability for getting one of these values (we also need $k$ for this computation). Since the inclusion probability used is unbiased using the same arguments we learned in class when talking about "inverse probability", this estimator's expectation is 1. We have that

$$\mathbb{E}\left[\widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])\right] = \mathbb{E}\left[\widehat{I}_{v \rightsquigarrow u} \cdot \alpha(d_{vu})\right] = \alpha(d_{vu})\mathbb{E}\left[\widehat{I}_{v \rightsquigarrow u}\right] = \alpha(d_{vu})$$

Where in the first equality we used that if $ADS(v)[u] \neq d_{vu}$ then $u \notin ADS(v)$ so $\widehat{I}_{v \rightsquigarrow u} = 0$ and the multiplication is therefore 0 in both sides. Now, using the linearity of expectation, we get that

$$\mathbb{E}\left[\sum_{u \in S} \widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])\right] = \sum_{u \in S} \mathbb{E}\left[\widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])\right] = \sum_{u \in S} \alpha(d_{vu})$$

(b)

i. Since we assume $\{\widehat{I}_{v \rightsquigarrow u_i}\}_i$ are (pair-wise) independent, the variance of the sum is the sum of variances. Since variance is always non-negative:

$$Var\left[\widehat{C}_{\alpha,V}\right] - Var\left[\widehat{C}_{\alpha,S}\right] = \sum_{u \in V \setminus S} Var\left[\widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])\right] \geq 0$$

Where the $\geq$ turns to $>$ if there exists $u \in V \setminus S$ that isn't always in $ADS(v)$, so it cannot have zero variance. For example, this must happen when $|V \setminus S| > k$.

ii. No, since there could be vertices $u$ such that $\widehat{I}_{v \rightsquigarrow u} \cdot \alpha(ADS(v)[u])$ has expectation 1, say when $\alpha$ is some $T$ thresh-hold decay function with big enough $T$, and $u$ is a vertex with $d_{vu} < T$, but high variance (almost 1) because the inclusion probability could be really small making $\widehat{I}_{v \rightsquigarrow u} = 0$ most of the time, if there are many (say $100k^5$) vertices closer to $v$ than $u$. In that case, the $CV$ of $\widehat{C}_{\alpha,\{u\}}$ is almost 1.

(c)

i. When there are more vertices in $S$ the variance rises, but so does the expectation. So what matters when adding a node $u$ to $S$ is the relation between the $CV$ of $\widehat{C}_{\alpha,S}$ and that of $\widehat{C}_{\alpha,\{u\}}$. However, as we have seen, when $|S|$ is small, there is not much we can guarantee about the CV of $\widehat{C}_{\alpha,S}$ in the general case, and there are bad examples. When $|S|$ is large enough, there is more stability and lower dominance of bad examples, so we can guarantee more.

ii. When the distances of $S$ from $v$ are small relatively to distances of other nodes from $v$, the inclusion probability for nodes from $S$ is higher, and therefore the variance of our estimator $\widehat{C}_{\alpha,S}$ is lower, reducing its $CV$.

(d) We use the $ADS$ algorithm learned in class, with the difference that we will only save vertices from $S$ in our sketch. Namely, instead of iterating over all nodes $u$ by increasing $h(u)$, we will iterate over all $u \in S$ by increasing $h(u)$. The expected number of elements in $ADS(v)$ is now at most $k \ln |S|$, using the same argument we learned in class. Therefore, we process each vertex in expectation at most $k \ln |S|$ times so the expected total running time is upper bounded by $k(m+n) \ln |S|$. Since there are only $|S|$ iterations and each is linear, the running time is also upper bounded by $|S|\,(m+n)$, may be relevant for small $S$. To get the required sketch, let $k := \varepsilon^{-2}$. The $CV$ should be $\approx 1/\sqrt{k} = \varepsilon$, using the same $CV$ bound we learned in class for $\widehat{C}_{\alpha,V}(v)$, since the inclusion probabilities for elements from $S$ in our new sketch are exactly as if there are only $S \cup \{v\}$ in our graph using the classic $ADS$, with the same distances.

(e) Yes, again, the resulting sketch is as if we took $ADS$ from a graph with only $S \cup U$ (actually better since we don't waste space on vertices from $U \setminus S$ in our new sketch) and so its like having $\beta \equiv 1$ in that graph, the case in which the $CV$ is $O(1/\sqrt{k})$, as mentioned in class. The algorithm is the same one learned in class: we compute the "union" (as defined in class) $ADS(U)$ from the sets $ADS(v)$ for $v \in U$, and then we apply a centrality estimator for $ADS(U)$.

3.

(a) Our sketch will maintain the set $\{(h(t_i), t_i)\}_i$ of all timestamps with hash value in the bottom-$k$ hash values of timestamps seen before them. (very similar to what we did in the bottom-$k$ $ADS$, just that now we switched the roles of distance and hash value: we go over the timestamps with increasing "distance" instead of increasing hash value, and add to the sketch if our hash value is one of the smallest $k$, instead of distance)
How to update the sketch when a new time stamp arrives: If its hash value is one of the smallest $k$ hash values of timestamps in our sketch (or if there are less than $k$ timestamps in our sketch) add it to our sketch. Otherwise, ignore it. (we note that if the hash value is one of the smallest $k$ in the sketch it must also be one of the smallest $k$ seen so far, since any hash value smaller than it that was seen before it must have also been added)
The estimator: Just like in $ADS$, we use the estimator

$$\sum_{i|t_i<t} \widehat{I}_{t_i} \alpha(SK[t_i])$$

Where $\widehat{I}_{t_i}$ is the unbiased "presence" estimator of $t_i$ based on inverse-probability, and $SK[t_i]$ is $t_i$ if its in the sketch, and $t$ otherwise (doesn't really matter since the multiplication will be 0 anyway because $\widehat{I}_{t_i} = 0$ if $t_i$ isn't in the sketch)
The expected size of our sketch is $k \ln n$, our estimator is unbiased, and the $CV$ is at most $1/\sqrt{2k-2}$, all from the same arguments learned in class for bottom-$k$ $ADS$. All that's left is to choose $k = 1/\varepsilon^2$.

(b) This time, we want our sketch to contain the largest timestamps (closest to $t$) instead of the smallest. Our sketch will maintain the $k$ most recent timestamps, along with all previous timestamps whose hash value is one of the smallest $k$ hash values of timestamps that came after them.
How to update the sketch when a new time stamp arrives: We always add it to our sketch. Then, we throw out of the sketch all timestamps with hash value that is no longer in the bottom-$k$ hash values of later timestamps. This can be done in linear (with respect to sketch size) time by maintaining a counter for every sketch element of how many later timestamps have lower hash values, so when we add a new element we visit each sketch element once, if it has greater hash value we increase its counter, and if the counter went above $k$, we throw that element out.

The estimator: We use the estimator

$$\sum_{i|t_i<t} \widehat{I}_{t_i} \alpha(t - SK[t_i])$$

Since the sketch (and the estimator) we end up with are just like when receiving $t - t_i$ in an increasing order and using the sketch from $(a)$, we have that the estimator is unbiased and that the $CV$ is just like in the previous section, so we pick $k = 1/\varepsilon^2$ this time too. The only thing that might change is the sketch size and update time: We made sure that the update time stays linear in the sketch size. This time we have removals so bounding the final sketch size isn't all that matters (actually it is because we are talking about stream sketch so the "final" point could be anytime). From the same argument learned in class, after processing $d$ elements the expected sketch size is $k \ln d$.

4.

(a) $f$ is not monotone: We have that $f(\emptyset) = f(V) = 0$, so $f$ is monotone iff $f(S) = 0$ for every $S \subseteq V$, since it should hold that $f(\emptyset) \leq f(S) \leq f(V)$ for monotone functions. Since weights are positive (assumption in the question), we have that $f(S) > 0$ for every $S \neq \emptyset, V$ (so that the sum isn't trivial), therefore $f$ is not monotone.
$f$ is submodular: Let $S \subset T$ and $i \in V \setminus T$, then

$$f_S(i) = f(S \cup \{i\}) - f(S) = \sum_{(u,v)\in E|u\in S\cup\{i\}, v\in V\setminus(S\cup\{i\})} w_{uv} - \sum_{(u,v)\in E|u\in S, v\in V\setminus S} w_{uv} =$$

$$\sum_{(i,v)\in E|v\in V\setminus(S\cup\{i\})} w_{uv} - \sum_{(u,i)\in E|u\in S} w_{uv} \geq \sum_{(i,v)\in E|v\in V\setminus(T\cup\{i\})} w_{uv} - \sum_{(u,i)\in E|u\in T} w_{uv} = f_T(i)$$

Where we used that:

$$S \subset T \Rightarrow V \setminus (S \cup \{i\}) \supset V \setminus (T \cup \{i\}) \Rightarrow \sum_{(i,v)\in E|v\in V\setminus(S\cup\{i\})} w_{uv} \geq \sum_{(i,v)\in E|v\in V\setminus(T\cup\{i\})} w_{uv}$$

$$S \subset T \Rightarrow \sum_{(u,i)\in E|u\in S} w_{uv} \leq \sum_{(u,i)\in E|u\in T} w_{uv} \Rightarrow -\sum_{(u,i)\in E|u\in S} w_{uv} \geq -\sum_{(u,i)\in E|u\in T} w_{uv}$$

(b) For $S \subseteq V_1$, denote by $N_S(u)[k]$ the weight of the $k$'th largest edge adjacent to $u$ in the subgraph of $G$ induced by $S$ and $V_2$, and let it be zero if there are less than $k$ edges adjacent to $u$. It holds that $N_S(u)[k] \leq N_T(u)[k]$ for every $S \subset T \subset V_1, u \in V_2, k$ since all edges adjacent to $u$ in the graph induced by $S, V_2$ are also adjacent to $u$ in the graph induced by $T, V_2$, so we pick the $k$'th largest element from a better set and therefore the weight of the $k$'th largest edge can only be improved. If there are no $k$ adjacent vertices in $S, V_2$ and in $T, V_2$ there are, then the inequality still holds since edge weights are positive.
$f$ is monotone: Let $S \subset T \subset V_1$, then by using $N_S(u)[k] \leq N_T(u)[k]$, we have that

$$f(S) = \sum_{u\in V_2} N_S(u)[1] + N_S(u)[2] \leq \sum_{u\in V_2} N_T(u)[1] + N_T(u)[2] = f(T)$$

$f$ is submodular: Let $S \subset T \subset V_1$ and $i \in V \setminus T$, then

$$\big(N_{S\cup\{i\}}(u)[1] + N_{S\cup\{i\}}(u)[2]\big) - (N_S(u)[1] + N_S(u)[2]) =$$

$$\begin{cases} 0 & w(u,i) \leq N_S(u)[2] \\ w(u,i) - N_S(u)[2] & otherwise \end{cases} = \max(0, w(u,i) - N_S(u)[2])$$

Because if $w(u,i) \leq N_S(u)[2]$ nothing is changed,
but otherwise the previous 2nd edge is kicked out and $(u,i)$ joins.

$$f_S(i) = \sum_{u\in V_2} \max(0, w(u,i) - N_S(u)[2]) \geq \sum_{u\in V_2} \max(0, w(u,i) - N_T(u)[2]) = f_T(i)$$

Where we used that
$N_S(u)[2] \leq N_T(u)[2] \Rightarrow w(u,i) - N_S(u)[2] \geq w(u,i) - N_T(u)[2] \Rightarrow$
$\max(0, w(u,i) - N_S(u)[2]) \geq \max(0, w(u,i) - N_T(u)[2])$
(the function $\max(0,x)$ is non-decreasing)

5.

(a) It still holds that $f(G_i \cup Opt_k) \geq f(Opt_k)$ and that $f(G_i \cup Opt_k) \leq f(G_i) + k \max_u f_{G_i}(u)$ so we still have that $f(Opt_k) - f(G_{i+1}) \leq (f(Opt_k) - f(G_i)) \left(1 - \frac{1}{k}\right)$, all from the exact same arguments learned in class. The only change is that now we have

$$f(Opt_k) - f(G_{ck}) \leq \left(1 - \frac{1}{k}\right)^{ck} (f(Opt_k) - f(\emptyset))$$
$$f(G_{ck}) \geq f(Opt_k) \left(1 - \left(1 - \frac{1}{k}\right)^{ck}\right) > f(Opt_k) \left(1 - \frac{1}{e^c}\right)$$

(b) Same as before, with the change that now

$$f(G_{i+1}) \geq f(G_i) + (1 - \varepsilon) \max_u f_{G_i}(u) \geq f(G_i) + (1 - \varepsilon) \frac{f(Opt_k) - f(G_i)}{k}$$

So now

$$f(Opt_k) - f(G_{i+1}) \leq (f(Opt_k) - f(G_i)) \left(1 - \frac{1 - \varepsilon}{k}\right)$$

Giving that

$$f(Opt_k) - f(G_{ck}) \leq \left(1 - \frac{1 - \varepsilon}{k}\right)^{ck} (f(Opt_k) - f(\emptyset))$$
$$f(G_{ck}) \geq f(Opt_k) \left(1 - \left(1 - \frac{1 - \varepsilon}{k}\right)^{ck}\right) \approx> f(Opt_k) \left(1 - \frac{1}{e^{c(1-\varepsilon)}}\right)$$