Exercise Sheet
COMSM0068 Advanced Topics in Theoretical Computer Science
2020/2021

1 Minimum Spanning Tree (MST)

We consider a weighted graph $G = (V, E, w)$, where $w : E \rightarrow \mathbb{N}$ is an edge weight function. A *minimum spanning tree* $F \subseteq E$ in G is a spanning tree in G of minimum weight, i.e., the sum of its edge weights is as small as possible.

We consider the streaming edge-arrival model where the edges arrive together with their weights. More specifically, the input stream consists of a sequence of tuples $(e_i, w(e_i))_i$, where $w(e_i)$ is the weight of edge e_i.

1. Give a 1-pass semi-streaming algorithm for computing an MST.

   Hint: Adapt the spanning tree algorithm from the lecture.

   Solution:

   $F \leftarrow \emptyset$
   While stream not empty:

   (a) Let e be the next edge in the stream

   (b) **if** $(F \cup \{e\})$ does not contain a cycle **then** $F \leftarrow F \cup \{e\}$

   (c) **else** ($(F \cup \{e\})$ does contain a cycle)

      i. Let C be the edge set of the (unique) cycle in $F \cup \{e\}$
      ii. Let f be an edge of maximum weight in $C \setminus \{e\}$
      iii. **if** $w(f) > w(e)$ **then** $F \leftarrow (F \setminus \{f\}) \cup \{e\}$

   return F

2. Let E_i be the first i edges in the stream, $G_i = (V, E_i, w|_{E_i})$ (where $w|_{E_i}$ denotes the weight function w restricted to the domain E_i), and let F_i denote the collection of edges stored by the algorithm given in the previous exercise after iteration i. Prove by induction that F_i is a MST in G_i.

   The following property may be useful:

   Lemma 1. *Let $T \subseteq E$ be a spanning tree in a weighted graph $G = (V, E, w)$. Then, if T is not a minimum spanning tree, then there exists an edge $e \in E \setminus T$ such that $w(e) < w(f)$, for at least one edge f different to e in the unique cycle in $T \cup \{e\}$.*

Solution:

Proof.
Base case. $F_0 = \emptyset$ and $E_0 = \emptyset$. Observe that F_0 is a MST of an empty graph.
Induction step. Let F_i be a MST in graph G_i. We will only consider the interesting case when $F_{i+1} = (F_i \setminus \{f_{i+1}\}) \cup \{e_{i+1}\}$, where f_{i+1} is the edge of the cycle C_{i+1} that was removed when inserting e_{i+1}. Observe that this implies that $w(e_{i+1}) < w(f_{i+1})$.
Assume for the sake of a contradiction that F_{i+1} is not a MST in G_{i+1}. Then, by Lemma 1, there exists an edge $e \in E_{i+1} \setminus F_{i+1}$ such that $F_{i+1} \cup \{e\}$ contains a unique cycle C with $w(e) < w(f)$ for some edge $f \in C \setminus \{e\}$. Since $e_{i+1} \in F_{i+1}$ and $e \notin F_{i+1}$, we have $e \neq e_{i+1}$ and therefore $e \in E_i$.
We will argue now that $F_i \cup \{e\}$ also contains a cycle C' such that e is not a heaviest edge in C'. This, however, contradicts then the fact that F_i is a MST, since we could swap in F_i the edge e with a heaviest edge in C' and create a spanning tree of less weight.
We consider two cases:

(a) First, suppose that $e_{i+1} \notin C$. Then, $C \subseteq E_i$ and C also constitutes a cycle in $F_i \cup \{e\}$ with the same property that e is not a heaviest edge in this cycle.

(b) Next, suppose that $e_{i+1} \in C$. Then, the symmetric difference $C' = C \oplus (C_{i+1} \setminus \{e_{i+1}\})$ (with $A \oplus B := (A \setminus B) \cup (B \setminus A)$) also forms a cycle that necessarily contains the edges f_{i+1} and e (see Figure 1). Two configurations are possible:

Suppose first that $f \in C'$ (top illustration in Figure 1) . Then we are done since $w(e) < w(f)$.

Next, suppose that $f \notin C'$ (bottom illustration in Figure 1). Then, we necessarily have that $f \in C_{i+1}$ and since the algorithm removed f_{i+1} from F_i instead of f, we have $w(f) \leq w(f_{i+1})$. Since $w(e) < w(f)$, we also have $w(e) < w_{f_{i+1}}$ and e is thus not the heaviest edge.

□

2 Matchings

2.1 Weighted Matching with Restricted Edge Weights

Let $G = (V, E, w)$ be a weighted graph with $w : E \to \{1, 2\}$. Consider the following two algorithms, which can be implemented as semi-streaming algorithms, for computing matchings:

$\mathbf{A}_1$: Ignore the edge weights and use the Greedy matching algorithm to compute a maximal matching M. Return M with its edge weights.

$\mathbf{A}_2$: Run Greedy on the subgraph of edges of weight 1, which produces a matching M_1. In parallel, run Greedy on the subgraph of edges of weight 2, which produces a matching M_2. The output matching M is obtained by inserting every edge of M_1 into M_2 if possible.

1. What is the approximation guarantee of $\mathbf{A}_1$?

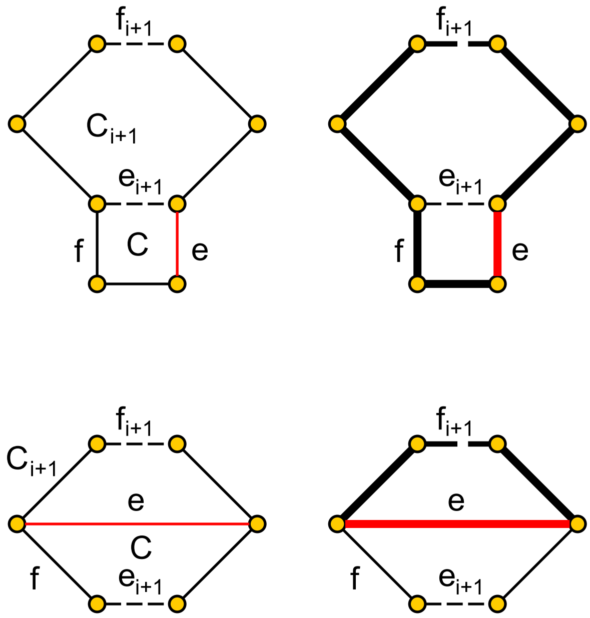

Figure 1: Solution to the MST exercise. Top: Case $f \in C'$. Bottom: Case $f \notin C'$.

Solution:

Let M^* be a maximum matching in the input graph and M be the matching returned by $\mathbf{A}_1$. We know that GREEDY has an approximation guarantee of $\frac{1}{2}$, so

$$|M| \geq \frac{1}{2}|M^*| \ .$$

Since each edge weight is in $\{1, 2\}$, we have:

$$\begin{aligned} w(M) &\geq |M| \ , \text{ and} \\ |M^*| &\geq \frac{1}{2}w(M^*) \ . \end{aligned}$$

Combining, we obtain:

$$w(M) \geq |M| \geq \frac{1}{2}|M^*| \geq \frac{1}{4}w(M^*) \ .$$

See Figure 2 for a worst case example.

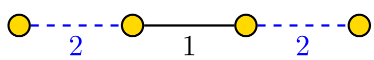

Figure 2: A worst case example of $\mathbf{A}_1$.

2. What is the approximation guarantee of $\mathbf{A}_2$?

   Solution:

Let M^* be a maximum matching in the input graph. Let $M_1^* \subseteq M^*$ denote the subset of edges of weight 1, and let M_2^* denote the subset of edges of weight 2. For each edge $m \in M_2$, let $C(m)$ denote the set of at most 2 edges from M_1 that are incident to m. In other words, m is responsible for these at most two edges for not being added to the final output matching M.
Next, for any $i \in \{1, 2\}$, observe that since M_i is maximal, every edge $m \in M_i^*$ is either adjacent to an edge from M_i or is itself contained in M_i.
We will now charge the weights of the edges M^* to the edges in M, as follows:

- Let $m \in M_2^*$: We charge $w(m)$ to every edge that is incident to m in M_2. If there is no such edge, then $m \in M_2$, and we charge m by $w(m)$.
- Let $m \in M_1^*$: Let $N_1(m)$ denote the edges of M_1 that are incident to m, or, if there are no such edges (which implies $m \in M_1$), let $N_1(m) = m$. We now charge the weight $w(m)$ to every edge in $N_1(m)$. Then, if an edge in $N_1(m)$ is not included in the output matching, then we transfer its charge to the edge in M_2 that prevent it from being inserted.

Observe first that we inject at least $w(M^*)$ charge to the edges of the output matching. It remains to bound the maximum charge of an edge in M:

- Consider an edge $m \in M_1 \cap M$, i.e., m is included in the the output matching. Then m is charged at most $2w(m)$.
- Consider an edge $m \in M_2$. Then, $m \cup C(m)$ forms a path of length at most 3 and thus covers at most 4 vertices. This implies that $m \cup C(m)$ is incident to at most 4 edges from M^*. Since $m \cup C(m)$ contains only one edge from M_2 (i.e., the edge m), at most two of these 4 edges are from M_2^*. Hence, m has a charge of at most $2 \cdot 2 + 2 \cdot 1 = 3w(m)$ (since $w(m) = 2$).

Overall, an edge in the output matching receives a charge at most three times its own weight. Hence, $w(M^*) \leq \frac{1}{3}w(M)$. See Figure 3 for a worst case example.

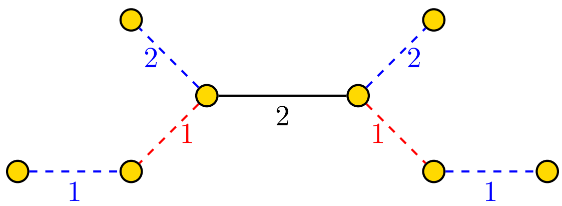

Figure 3: A worst case example of $\mathbf{A}_2$.

2.2 Weighted Matching Algorithm from the Lecture

Give an example of an input stream on which the algorithm for weighted matching discussed in the lecture produces an approximation ratio close to 1/6. Such an example input stream demostrates that our analysis is best possible.

Solution:

Figure 4 shows a hard instance that can easily be extended to yield an approximation ratio arbitrarily close to 6. In this example, a weight x^- means a value of $x - \epsilon$, for some arbitrarily small $\epsilon > 0$. Edges arrive in the following order: $1, 2^-, 2, 4^-, 4, 8^-, \dots, 64, 128^-, 128^-$. Observe that the algorithm outputs the edge with weight 64. The red edges form an optimal matching of weight 382. The algorithm therefore produces a $64/382 \approx 1/5.968$ approximation.

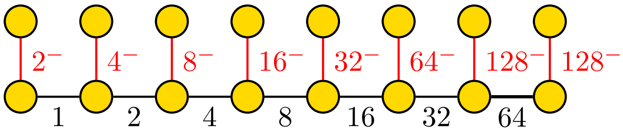

Figure 4: Hard Instance example for question 2.3.

3 Bounding the Error Probability of Randomized Algorithms

Randomized algorithms often invoke subroutines that are themselves randomized. Assume that our algorithm **A** executes the subroutines $R_1, R_2, \dots, R_k$ and each subroutine R_i has a failure probability of ϵ_i. Denote by E_i the event that subroutine i fails. Observe that E_i and E_j may be arbitrarily correlated. Our algorithm fails if at least one subroutine fails. We would therefore like to compute the probability:

$$Pr[E_1 \cup E_2 \cup \dots \cup E_k] \ .$$

Show by induction over k that

$$Pr[E_1 \cup E_2 \cup \dots \cup E_k] \leq \sum_{i=1}^{k} \epsilon_i \ .$$

Remark: The bound we ask you to prove is known as the union bound.

Solution:

Proof.
Base case. $Pr[E_1] = \epsilon_1 \leq \epsilon_1$
Induction step. Assume that $Pr[E_1 \cup E_2 \cup \dots \cup E_k] \leq \sum_{i=1}^{k} \epsilon_i$ holds and show that it holds for $k+1$. Using the fact that all probabilities are non-negative and $Pr[A \cup B] = Pr[A] + Pr[B] - Pr[A \cap B]$, we obtain:

$$\begin{aligned} Pr[\cup_{i=1}^{k} E_i \cup E_{k+1}] &= Pr[\cup_{i=1}^{k} E_i] + Pr[E_{k+1}] - Pr[\cup_{i=1}^{k} E_i \cap E_{k+1}] \\ &\leq Pr[\cup_{i=1}^{k} E_i] + Pr[E_{k+1}] \leq \sum_{i=1}^{k} \epsilon_i + \epsilon_{k+1} = \sum_{i=1}^{k+1} \epsilon_i \ . \end{aligned}$$

□

4 Sampling $\min\{k, \deg(v)\}$ Edges Incident to a Given Vertex (hard!)

The insertion-deletion matching algorithm discussed in Lecture 13 solves the following subproblem:

Let $v \in V$ be a vertex. Compute $\min\{k, \deg(v)\}$ arbitrary edges incident to v in an insertion-deletion graph stream.

This is achieved by running *enough* l_0-samplers on the substream of edges incident to v. Assuming that the l_0-samplers never fail (recall that the samplers themselves have a small error probability, but for simplicity, we assume that they do not fail here), how many l_0-samplers need to be run in parallel in order to solve this task?

Observe that this problem can be rephrased as follows: We have $\deg(v)$ bins. How many balls are needed so that, if we throw each ball into a random bin, at least $\min\{k, \deg(v)\}$ bins are non-empty?

Hint: Let t_i be the expected number of balls needed to hit an empty bin, conditioned on $i - 1$ bins being already non-empty. Using the t_i, what is the expected number of balls needed overall? Then, use the Chernoff bound to obtain a high probability result.