Institute of Theoretical Computer Science
Bernd Gärtner, Mohsen Ghaffari, Rasmus Kyng, Angelika Steger, David Steurer

Solution 1A

The modified algorithm performs two recursive calls. The first recursive call is on a graph with at most $n/4$ vertices (as one Borůvka step reduces the number of vertices by at least a factor of 2) and it contains at most $p \cdot m$ edges in expectation.

The second recursive call is also on a graph with at most $n/4$ vertices and we will show below that it contains at most $\frac{n/4}{p}$ edges in expectation.

Hence, following the argumentation on page 5 of the script, in order to show that for a given $p \in [0,1]$ the expected runtime of the algorithm is $O(n+m)$, it suffices to prove that for given constants $C_B, C_{FH} \geq 0$ there exists a constant $C \geq 0$ such that

$$C_B(n+m) + m + C(n/4 + p \cdot m) + C_{FH}(n/4+m) + C\left(n/4 + \frac{n/4}{p}\right) \leq C(n+m)$$

or equivalently

$$C_B(n+m) + C_{FH}(n/4+m) + C\left(n\left(\frac{1}{2} + \frac{1}{4p}\right) + p \cdot m\right) \leq C(n+m).$$

One can always find such a constant C as long as $\frac{1}{2} + \frac{1}{4p} < 1$ and $p < 1$. Hence, any $p \in (0.5, 1)$ works.

It remains to prove that the input to the second recursive call is a graph which contains at most $\frac{n/4}{p}$ edges in expectation.

We show this by modifying the analysis on page 6 and 7 of the script, replacing each $1/2$ by p and instead of flipping a fair coin we now flip a coin that comes up heads with probability p.

In particular, we consider the following procedure:

---

(1) Let $E' = T = F = \emptyset$.
(2) **for** $i = 1, \ldots, m$ **do**:
(3) **if** e_i connects two components of T **then**
(4) Flip a coin that comes up heads with probability p to decide whether e_i belongs to G'.
(5) If so, let $E' := E' + e_i$ and $T := T + e_i$, otherwise let $F := F + e_i$.
(6) **else**
(7) Flip a biased coin that comes up heads with probability p. If so, add e_i to E'.

Similar as in the script, we assume that the input graph G has n vertices (Later we apply the analysis to a graph with at most $n/4$ vertices). Let $G'' = (V'', E'')$ be the graph one obtains from G by removing all the T-heavy edges in G. We are interested in the expected size of E''. Note that $E'' = T \cup F$ and therefore the expected size of E'' is upper bounded by the expected number of times line 4 is executed. At most $n-1$ edges can be added to T during the execution. Hence, a similar argumentation as in the script together with a variant of the result of Exercise 1.1 implies that the expected size of F can be upper bounded by $\frac{n-1}{p} - (n-1)$. The variant of Exercise 1.1 assumes that $\Pr[X_i = 1] = 1 - p$ and requires to show that $\mathbb{E}[Z] = \frac{K}{p} - K$. Hence,

$$\mathbb{E}[|E''|] = \mathbb{E}[|T|] + \mathbb{E}[|F|] = n - 1 + \frac{n-1}{p} - (n-1) = \frac{n-1}{p}.$$

Now, replacing n with the number of nodes of the graph we obtain after two Borůvka steps (which is at most $n/4$) leads to the desired upper bound of $\frac{n/4}{p}$.

Solution 1B

Before stating the whole algorithm, we first consider the following procedure, which will be called twice by our final algorithm, and prove two lemmas about it.

Algorithm 1 ReduceEdges

Require: Input graph $G_{in} = (V_{in}, E_{in})$ is a weighted graph with pairwise distinct edge weights

$p \leftarrow \min\left(1, \frac{|V_{in}|^{4/3}}{|E_{in}|}\right)$
$G' \leftarrow \mathcal{A}_{sample}(G_{in}, p)$
$T \leftarrow \mathcal{A}_{MSF}(G')$
$G_{out} \leftarrow \mathcal{A}_{T-heavy}(G_{in}, T)$
return G_{out}

Lemma 1. *The graph G' has at most $100|V_{in}|^{4/3}$ edges with probability $1 - O\left(\frac{1}{|V_{in}|}\right)$.*

Proof. There is nothing to show in case $|E_{in}| \leq 100|V_{in}|^{4/3}$. Hence, consider the case that $|E_{in}| > 100|V_{in}|^{4/3}$. For each $e \in E_{in}$, we denote with X_e the indicator variable for the event that e gets sampled. We are interested in the random variable $X = \sum_{e \in E_{in}} X_e$. By linearity of expectation,

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{e \in E_{in}} X_e\right] = \sum_{e \in E_{in}} \mathbb{E}[X_e] = |E_{in}| \cdot \frac{|V_{in}|^{4/3}}{|E_{in}|} = |V_{in}|^{4/3}.$$

As X is a sum of independent Bernoulli-distributed random variables, we can apply a Chernoff bound to conclude that

$$\Pr[X > 100|V_{in}|^{4/3}] \leq \Pr[X \geq (1+1)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\mathbb{E}[X]} = O\left(\frac{1}{|V_{in}|}\right).$$

Therefore, $\Pr[X \leq 100|V_{in}|^{4/3}] \geq 1 - O\left(\frac{1}{|V_{in}|}\right)$, as desired. □

Lemma 2. *The graph* G_{out} *has at most* $\max(|V_{in}|^{4/3}, \frac{2|E_{in}|}{|V_{in}|^{1/3}})$ *edges with probability* $1-O\left(\frac{1}{|V_{in}|}\right)$.

Proof. We first show that in the procedure we analyzed to solve exercise 1A we have $\Pr\left[|E''| \geq \frac{2n}{p}\right] = O(1/n)$. Note that $\Pr\left[|E''| \geq \frac{2n}{p}\right]$ is upper bounded by the probability that among $\frac{2n}{p}$ independent biased coin throws, where each coin comes up heads with probability p, less than $n-1$ of them come up heads (as T contains at most $n-1$ edges at the end). Let X_i be the indicator for the event that the i-th coin comes up heads and $X = \sum_{i=1}^{\frac{2n}{p}} X_i$. We have $\mathbb{E}[X] = 2n$. Hence, a Chernoff Bound implies

$$\Pr[X \leq n-1] \leq \Pr[X \leq (1-0.5)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\mathbb{E}[X]0.5^2} = O(1/n)$$

and therefore $\Pr\left[|E''| \geq \frac{2n}{p}\right] = O(1/n)$.

With this result in hand, the remaining part of the proof is trivial. First, if $|E_{in}| \leq |V_{in}|^{4/3}$, then there is nothing to show. Hence, it remains to consider the case that $|E_{in}| > |V_{in}|^{4/3}$. As $\frac{2|V_{in}|}{p} = \frac{2|E_{in}|}{|V_{in}|^{1/3}}$, the discussion above directly implies that G_{out} has at most $\frac{2|E_{in}|}{|V_{in}|^{1/3}}$ edges with probability $1 - O\left(\frac{1}{|V_{in}|}\right)$. □

We are now ready to state the final algorithm.

Algorithm 2 MST

Require: Input graph G is a weighted graph with pairwise distinct edge weights
  $G_1 \leftarrow \text{ReduceEdges}(G)$
  $G_2 \leftarrow \text{ReduceEdges}(G_1)$
  $T \leftarrow \mathcal{A}_{MSF}(G_2)$
  return T

Note that Algorithm 1 only removes edges that are not contained in the unique MST of G. Hence, Algorithm 2 is correct, assuming that $\mathcal{A}_{MSF}$ only gets graphs with at most $100n^{4/3}$ edges as input. We show that this is the case with probability $1-O(1/n)$. We say that an invocation of Algorithm 1 is successful in case the two events of Lemma 1 and Lemma 2, which both hold with probability $1-O(1/n)$, are satisfied. By a Union Bound, the two invocations of Algorithm 1 in Algorithm 2 are successful with probability $1-O(1/n)$. It therefore only remains to show that having two successful invocations implies that G_2 contains at most $100n^{4/3}$ edges. Lemma 2 together with the fact that a graph contains at most $\binom{n}{2} \leq n^2/2$ edges implies that G_1 has at most $\frac{2n^2/2}{n^{1/3}} = n^{5/3}$ edges and therefore also that G_2 contains at most $\frac{2n^{5/3}}{n^{1/3}} \leq 100n^{4/3}$ edges, as desired.

Solution 2

1. First, the algorithm repeatedly contracts an edge chosen uniformly at random until the graph has $\lceil 2\alpha \rceil$ vertices. Second, the algorithm chooses a cut in the resulting graph uniformly at random. The chosen cut naturally corresponds to a cut in the original graph, which the algorithm outputs. For a given multigraph G and cut $(S, V \setminus S)$ of G, let $p_{G,(S,V\setminus S)}$ denote the probability that the algorithm outputs the cut $(S, V \setminus S)$ when the input graph is G. We define

$$p_n := \inf_{G \text{ is an } n\text{-node multigraph, } (S, V \setminus S) \text{ is an } \alpha\text{-approximate cut in } G} p_{G,(S,V\setminus S)}.$$

For $n \leq \lceil 2\alpha \rceil$, we have $p_n \geq \frac{1}{2^{\lceil 2\alpha \rceil}}$. Now, consider a multigraph with $n > \lceil 2\alpha \rceil$ vertices and let $(S, V \setminus S)$ be an arbitrary α-approximate minimum cut. The probability that during the first contraction no edge with one endpoint in S and one endpoint in $V \setminus S$ gets contracted is at least $1 - \frac{2\alpha}{n}$. In that case, there exists an α-approximate cut in the contracted graph that corresponds to the cut $(S, V \setminus S)$. Hence, for $n > \lceil 2\alpha \rceil$, we have

$$\begin{aligned}
p_n &\geq \left(1 - \frac{2\alpha}{n}\right) p_{n-1} \\
&\geq \frac{n-2\alpha}{n} \cdot \frac{n-1-2\alpha}{n-1} \cdot \ldots \cdot \frac{\lceil 2\alpha \rceil - 2\alpha + 1}{\lceil 2\alpha \rceil + 1} \cdot p_{\lceil 2\alpha \rceil} \\
&\geq \frac{n-\lceil 2\alpha \rceil}{n} \cdot \frac{n-1-\lceil 2\alpha \rceil}{n-1} \cdot \ldots \cdot \frac{1}{\lceil 2\alpha \rceil + 1} \cdot p_{\lceil 2\alpha \rceil} \\
&= \frac{\prod_{i=1}^{\lceil 2\alpha \rceil} i}{\prod_{i=1}^{\lceil 2\alpha \rceil} (n-i+1)} p_{\lceil 2\alpha \rceil} \\
&\geq \frac{1}{(2n)^{\lceil 2\alpha \rceil}},
\end{aligned}$$

as desired.

2. Let $\mathcal{C}$ denote the set containing all the α-approximate minimum cuts. For each $(S, V \setminus S) \in \mathcal{C}$, let $E_{(S,V\setminus S)}$ denote the event that the algorithm from a) outputs $(S, V \setminus S)$. We have $\Pr[E_{(S,V\setminus S)}] \geq \frac{1}{(2n)^{\lceil 2\alpha \rceil}}$. Moreover, the events are pairwise disjoint. Hence, we have

$$1 \geq \Pr\left[\bigcup_{(S,V\setminus S)\in\mathcal{C}} E_{(S,V\setminus S)}\right] = \sum_{(S,V\setminus S)\in\mathcal{C}} \Pr[E_{(S,V\setminus S)}] \geq \frac{|\mathcal{C}|}{(2n)^{\lceil 2\alpha \rceil}}$$

and therefore $|\mathcal{C}| \leq (2n)^{\lceil 2\alpha \rceil}$.

3. First, there is a small mistake in the exercise. It should be "Consider an arbitrary integer $\alpha \geq 1$" instead of "Consider an arbitrary integer $2\alpha \geq 2$". Consider an arbitrary n and an even k. We let H be a cycle where every two neighboring nodes are connected by $k/2$ multi-edges. Clearly, $\mu(H) = k$. To show that H has $\Omega(n^{\lceil 2\alpha \rceil})$ α-approximate minimum cuts, let V' be an arbitrary set of $\lfloor n/2 \rfloor$ pairwise non-adjacent vertices of H, i.e., V' is an independent set. Now, consider an arbitrary set of 2α vertices in V'. These 2α vertices define α non-adjacent intervals in H. Let S be the set containing each node that is contained in one of the α intervals. $(S, V \setminus S)$ is an α-approximate cut. Moreover, as V' is an independent set, any 2α vertices in V' define a unique α-approximate cut. Hence, H contains at least

$$\binom{\lfloor n/2 \rfloor}{2\alpha} \geq \left(\frac{\lfloor n/2 \rfloor}{2\alpha}\right)^{2\alpha}$$

α-approximate minimum cuts, as desired.

Solution 3

1. Consider an arbitrary $i \in \{r+1, r+2, \ldots, n\}$. We denote with B_i the event that i comes first among the keys 1 up to i. Moreover, we denote with C_i the event that one of the first r keys comes first among the keys 1 up to $i-1$. We have $\Pr[B_i] = 1/i$ and $\Pr[C_i] = r/(i-1)$. Moreover, the two events B_i and C_i are independent and $A_i = B_i \cap C_i$. Therefore,

$$\Pr[A_i] = \Pr[B_i \cap C_i] = \Pr[B_i] \cdot \Pr[C_i] = \frac{r}{i(i-1)},$$

as desired.

2. Assume that the event A_i occured. This implies that the first $i-1$ keys all end up in the left subtree of i. However, this does not imply that the left subtree of i is a random binary search tree on the first $i-1$ keys (conditioned on A_i). The reason for this is that A_i implies that one of the first r keys will be the left child of i. For $j \in [r]$, let $A_{i,j}$ denote the event that A_i occured and j is the left child of i. Conditioned on $A_{i,j}$, the left subtree of j is a random binary search tree with $j-1$ nodes and the right subtree of j is a random binary search tree with $i-j-1$ nodes. The expected height of both of these random search trees is $O(\log i)$. Hence, the expected height of the subtree rooted at i is $O(\log i)$ and therefore $\mathbb{E}[X_{n,r}|A_i] = O(\log i)$.

3. We have

$$\mathbb{E}[X_{n,r}] = \frac{r}{n} O(\log n) + \sum_{i=r+1}^{n} \mathbb{E}[X_{n,r}|A_i]\Pr[A_i].$$

Note that $\frac{r}{n} O(\log n) = O(\log r)$. Hence, it suffices to show that $\sum_{i=r+1}^{n} \mathbb{E}[X_{n,r}|A_i]\Pr[A_i] = O(\log r)$.

We have

$$\begin{aligned}
\sum_{i=r+1}^{n} \mathbb{E}[X_{n,r}|A_i]\Pr[A_i] &\leq \sum_{i=r}^{\infty} \frac{r}{i^2} O(\log i) \\
&\leq \sum_{j=0}^{\infty} \sum_{i=r2^j}^{r2^{j+1}-1} \frac{r}{i^2} O(\log i) \\
&\leq \sum_{j=0}^{\infty} r2^j \frac{r}{(r2^j)^2} O(\log(r2^{j+1})) \\
&\leq \sum_{j=0}^{\infty} \frac{1}{2^j} (O(\log r) + (j+1)) \\
&= O(\log r) \sum_{j=0}^{\infty} 0.5^j + \sum_{j=0}^{\infty} \frac{j+1}{2^j} \\
&= O(\log r),
\end{aligned}$$

as desired.

Solution 4

Scanline approach: We assume that no two vertex points of P or Q have identical x-coordinates.

From the given input, it is easy to compute the respective upper and lower boundaries of P and Q in linear time. By the *upper boundary* of P, say, we mean an array (sorted by x-coordinates) of all vertices p of P with the property that a vertical ray shooting upwards from p intersects P only in p. An analogous definition can be given for the *lower boundary* of P (and Q).

Let $x_1, \ldots, x_n$ be a sorted list of the x-coordinates of all vertices of both P and Q. The idea is to iterate over all pairs (x_i, x_{i+1}) and to restrict our attention to the area between the two vertical lines through x_i and x_{i+1}, respectively. Restricted to that area, both P and Q are trapezoids T_P and T_Q with two vertical boundaries through x_i and x_{i+1}, respectively.[1]

Iterating over all pairs (x_i, x_{i+1}) can be done by maintaining four pointers into the respective upper and lower boundaries of P and Q, and by advancing them one at a time. Slightly more detailed, each of those four pointers points between two vertices (or, in other words, it points to an edge) and the considered interval (x_i, x_{i+1}) is so that x_i is the largest x-coordinate among all vertices that are directly to the left of any of these four pointers, and x_{i+1} is the smallest x-coordinate among all vertices that are directly to the right of these four pointers. Hence, in each iteration, it is easy to reconstruct in constant time the left and right boundaries of the trapezoids T_P and T_Q (or, in other words, the respective intersections of P and Q with the vertical lines through x_i and x_{i+1}).

Given that it is a constant size problem, for two such trapezoids we can clearly decide in constant time whether one of them contains the other, whether they intersect but neither trapezoid contains the other, or whether they are disjoint. If we want to be more careful than that, then we can observe that T_P contains T_Q if and only if the vertical boundaries of T_P contain the respective vertical boundaries of T_Q, and vice versa. Also, T_P and T_Q are disjoint if and only if the vertical boundaries of T_P are both strictly above (or both strictly below) the respective vertical boundaries of T_Q.

We note that case (i) holds globally if and only if in every iteration T_P contains T_Q. A similar statement can be made for the cases (ii) and (iv). By elimination, if we are neither in case (i), (ii) or (iv), then and only then case (iii) must hold.

The upper bound of $O(n)$ on the running time follows because the initialization can be done in linear time, and because each one of the $n-1$ iterations requires at most constant time.

Rotation approach: We give two algorithms, one for deciding whether we are in case (i), and one for deciding whether we are in case (iv). This automatically gives us an algorithm for case (ii), since it is symmetric to case (i). And it also gives us an algorithm for case (iii), which holds if and only if neither of the other three cases hold.

Throughout, we assume that no two edges of P and Q have the same slope.

We first note that case (i) can be characterized as follows: For each edge of P we consider its line extension, and then the halfplane h_P which has this line as its boundary and which completely contains P. We further consider h_Q which is the smallest (in terms of set inclusion) halfplane whose boundary has the same slope (and the same "orientation") as that of h_P and which contains Q. Then, h_P must contain h_Q for all edges of P.

[1] There are some exceptional cases, where instead of a trapezoid we might have a triangle, a single point, or even the empty set. But these special cases are not harder to deal with than the typical case, and so we will not go further into them.

Checking these conditions can now easily be done by iterating over all edges of P and by maintaining a pointer that points to the unique vertex of Q that is contained in the boundary of the respective halfplane h_Q (it is always the vertex where one of the incident edges has a slope that is smaller than the slope of h_P, and where the other edge has a slope that is larger[2]). Updating that pointer can be done by walking in the appropriate direction in the vertex array of Q. If we iterate over the edges of P in clockwise order, say, then it is not hard to see that this pointer will have visited each vertex of Q at most once (it may however stay at the same vertex for more than just one iteration). Hence the total number of iterations is linear in the number of edges (and thus vertices) of P, and the extra work required for maintaining the extra pointer is linear in the number of vertices of P and Q. The desired running time of $O(n)$ follows.

Case (iv) can be characterized in a similar way: We again consider the halfplanes h_P for all edges of P. The corresponding halfplanes h_Q are defined as before, with the only difference that we change their orientation (i.e., before, the intersection of h_P and h_Q was always a halfplane, whereas now it is either a strip or the empty set). Then, case (iv) holds if and only if we have $h_P \cap h_Q = \emptyset$ for at least on edge of P. The "if" is obvious since h_P and h_Q are supersets of the respective polygons. The "only if" is technically false, but it holds for at least one of the two cases if we interchange the roles of P and Q (therefore, the algorithm we are going to describe simply has to be invoked a second time with the roles of P and Q interchanged). This can be seen by considering a line that separates P from Q, by then blowing this line up to a strip so that it touches both P and Q, and then by rotating the two boundary edges around the points where they touch P and Q until one of those boundaries coincides with an edge of either P or Q.

As for the algorithm, we again iterate over all edges of P and consider the halfplane h_P, we maintain a pointer that allows us to reconstruct the corresponding halfplane h_Q, and we then check in constant time whether the intersection of h_P and h_Q is empty. The running time bound of $O(n)$ is analogous to what we did for case (i).

[2]We ignore the technical issue of wrapping around.