University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithms Theory
Sample Solution Exercise Sheet 8

Due: Tuesday, 12th of January 2020, 4 pm

Exercise 1: Property of Augmenting Paths *(10+5* Points)*

Let $G = (V, E)$ be a *weighted, directed* graph with $n = |V|$ nodes and $m = |E|$ edges, and let $s, t \in V$.

(a) Let f be a valid s-t-flow in G and assume that $m_f \leq m$ is the number of edges $e \in E$ for which $f(e) > 0$. Show that f can be "decomposed" into flows along at most m_f s-t paths P_i (i.e., $1 \leq i \leq m_f$). More precisely, show that there exist at most m_f paths P_i each with a flow value f_i, such that for each $e \in E$ we have $\sum_{i:e \in P_i} f_i \leq f(e)$ and such that $\sum_i f_i = |f|$. *(5 Points)*

Hint: You can give a constructive proof using a greedy algorithm similar to the one in the lecture.

(b) The residual graph G_f can also be interpreted as a flow network with source s and sink t. Show that the minimum s-t-cut in the residual graph G_f has capacity exactly $|f^*| - |f|$, where f^* is a maximum flow of G. *(5 Points)*

(c) Let $|f| < |f^*|$. Show that there is always an augmenting path P for G with respect to a given flow f such that bottleneck$(P, f) \geq (|f^*| - |f|)/m$. *(5 Points)*

Sample Solution

(a) First some intuition: We construct these paths iteratively. First we find a flow along a s-t-path P_1 that consists only of edges that have positive flow with respect to f. We make the flow f_1 over P_1 as large as possible, so that all edges $e \in P_1$ have flow $f(e) \geq f_1$. Then we decrease the flow f along the path P_1 by a value f_1 to obtain a new, smaller flow. We rinse and repeat, until the remaining flow has value 0. In each repetition the number of edges with positive flow decreases by at least one. Thus we have split off at most m_f paths P_i. Now the formalism.

Let $f^{(i)}$ with $|f^{(i)}| > 0$ be the current flow that is left (initially $f^{(1)} := f$). Consider the graph $G_i = (V, E_i)$, where $E_i := \{e \in E \mid f^{(i)}(e) > 0\}$. Let P_i be an arbitrary s-t path in G_1. Such a path must exist given that $|f^{(i)}| > 0$. Let $f_i := \min_{e \in P_i} f^{(i)}(e)$ be the smallest flow over an edge of P_i. We "subtract" the flow along P_i from $f^{(i)}$ to obtain a new flow $f^{(i+1)}$ as follows.

$$f^{(i+1)}(e) := \begin{cases} f^{(i)}(e) - f_i, & \text{if } e \in P_i \\ f^{(i)}(e), & \text{else} \end{cases}$$

Then $f^{(i+1)}$ is again a valid flow given that $f^{(i)}$ was valid. For that, one can easily verify that the sum of incoming flows on each node $V \ni v \neq s, t$ still equals the sum of outgoing flows from v after subtracting f_i on all edges of P_i; for each edge in P_i that enters v there must be an edge of P_i that leaves v since P_i is an s-t-path. The value of the new flow is $|f^{(i+1)}| = |f^{(i)}| - f_i$.

After that, at least one edge e that had flow $f^{(i)}(e) > 0$ now has no flow left: $f^{(i+1)}(e) = 0$. Simply choose $e_i := \arg\min_{e \in P_i} f^{(i)}(e)$, then $f^{(i)}(e_i) = f_i$, hence $f^{(i+1)}(e_i) = f_i - f_i = 0$ by construction.

But this implies that in E_{i+1} we lost *at least one edge* compared to E_i (consider the definition of E_{i+1} from above).

Therefore, after at most m_f steps, there are no edges left anymore, i.e., the remaining flow value $|f^{(j)}|$ for some $j \leq m_f$ must be 0, and no additional path P_{j+1} can be split off anymore. In total we get at most m_f paths P_i, for $i \leq m_f$. The sum of flows along the paths P_i equals the flow f by construction, which implies that $\sum_{i:e\in P_i} f_i \leq f(e)$ and $\sum_i f_i = |f|$.

(b) Some intuition first. Roughly speaking, we can start out by considering a maximum flow f' of G_f. Then we prove (by dissassembling into augmenting paths) that a maximum flow f^* of G fulfills $|f'| = |f^*| - |f|$. Since f' is a max flow in G_f, according the the max-flow min-cut theorem, G_f must have a min-cut of capacity $|f'| = |f^*| - |f|$. Now more formally.

First of all, the statement is clear for $|f| = |f^*|$ (i.e., f is already optimal) because then there must be a cut with capacity 0 in G_f (c.f., lecture). Assume $|f| < |f^*|$. Again, we give a constructive argument. We start off by computing a maximum flow f' of G_f. We compute this flow f' step by step with a series of augmenting paths P_i in G_f.

Initially, let $f_1 := f$. Iteratively, let P_i be an augmenting path in G_{f_i} and let $f_{i+1} := \text{augment}(f_i, P_i)$ (note that the augment function "adds" an augmenting path to a flow and is given more formally in the lecture). Finally, in some step k, there will be no augmenting path in G_{f_k} anymore. Then there must be a cut in G_{f_k} that has capacity 0. Therefore $|f_k| = |f^*|$ is a maximum flow in G.

Then we consider the flow f' given by the augmenting paths $P_1, \ldots, P_k$ in G_f. More formally, we can define f' in iterative fashion as before, but starting with no flow and then adding the P_i: Let f'_1 be the largest possible flow along P_1 in G_f(!). Iteratively, let $f'_{i+1} := \text{augment}_{G_f}(f'_i, P_i)$ (whereas the augment_{G_f} function observes the capacities of G_f). Finally, we set $f' := f'_k$.

Note the symmetry of augmenting f in G along the paths P_i and augmenting the "zero-flow" in G_f along the same paths P_i. Due to this symmetry, f'_k is a max. flow of G_f because f_k is a max. flow of G. Therefore $|f'| = |f'_k| = |f_k| - |f| = |f^*| - |f|$. The final argument comes from the max-flow min-cut theorem. Since f' is a maximum flow in G_f, there must be a s-t-cut with capacity $|f'| = |f^*| - |f|$ in G_f.

(c) According to part (a) the maximum flow f' in G_f can be decomposed into at most $m_f \leq m$ paths P_i with respective flow values f_i, $1 \leq i \leq m$ such that these sum up to $|f'| = \sum_{i=1}^{m_f} f_i$. Consider a minimum s-t-cut in G_f with capacity $|f^*| - |f|$ (part b). Then f' has flow $|f'| = |f^*| - |f| = \sum_{i=1}^{m_f} f_i$ due to the max-flow min-cut theorem. Therefore the *average* size of the flow values f_i must be at least $(|f^*| - |f|)/m_f \geq (|f^*| - |f|)/m$. But then at least one f_j must be larger than average, i.e., $f_j \geq (|f^*| - |f|)/m$. This is the path P_j we were looking for.

One technicality remains: W.l.o.g. at most m edges of G_f must have a positive flow with respect to f. Note that G_f can have $2m$ edges, but we can make it so that only one direction of each edge has positive flow w.r.t. f (otherwise we can easily balance flows over forward and backward edge of a given edge out with one another so that at least one of the two flows becomes 0).

Exercise 2: Vertex Connectivity Refined *(10+5* Points)*

In the lecture we saw that a *minimum vertex cut* (MVC) can be computed in time $O(m \cdot n^3)$. Given that some parameters of the *undirected, unweighted* input graph $G = (V, E)$ are fairly small, we want to improve that runtime.

(a) Let κ be the vertex connectivity of G, i.e., the number of vertices in an MVC. Show that in order to compute a *global* MVC, it suffices to compute minimum s-t vertex cuts for $O(n \cdot \kappa)$ different source-target pairs (s, t). *(3 Points)*

Hint: Show that picking an arbitrary set of $\kappa + 1$ sources suffices.

(b) Give an algorithm to find an MVC in time $\mathrm{O}(m \cdot n \cdot \kappa^2)$. *(2 Points)*

Remark: You can assume for now that κ is already known. If you did not succeed in part (a), you might still be able to show a running time of $O(m \cdot n^2 \cdot \kappa)$ here.

(c) How can the same asymptotic running time be achieved without knowing κ in advance? *(3 Points)*

(d) Let δ be the minimum degree of G (i.e., the minimum number of incident edges of some node in G). Show that it even suffices to compute minimum s-t vertex cuts for $\mathrm{O}(n + \delta^2)$ pairs $s, t \in V$. Argue why this is the case and how you can find these $\mathrm{O}(n + \delta^2)$ s,t-pairs. *(6 Points)*

Hint: Let v be some node of degree δ and make a case distinction depending on if there exists an MVC that does not contain v or if v is contained in all MVCs.

(e) Explain how you get an overall running time of $\mathrm{O}\big(m\kappa \cdot \min(n+\delta^2, \kappa n)\big)$ for computing an MVC in a graph G with vertex connectivity κ and minimum degree δ. *(1 Point)*

Sample Solution

(a) Let X be some arbitrary MVC and let $s \notin X$. Let G' be the (unconnected) graph induced by $V \setminus X$. Let t be in a different connected component than s in G'. Then, in particular, X is also the (a) minimum s-t-vertex cut. Therefore, when computing a minimum s-t-vertex cut we will find X or another MVC that is equally small.

So we just have to make sure that we pick two nodes $s, t \in V$ that fulfill the requirements above. As in the hint, we pick $\kappa + 1$ different sources $s \in V$. But since $|X| = \kappa$, we must have picked at least one source $s \notin X$. Each time we pick a source s we compute the minimum s-t-vertex cut for *all* $t \in V$ that are not neighbors of s. That way we guarantee that we pick at least one t that is in a different connected component in G' than the source $s \notin X$. Overall we have to compute the minimum s-t-vertex cut for $(\kappa + 1) \cdot n = O(\kappa n)$ different pairs.

(b) The next observation is that by transforming the minimum s-t-vertex problem into a maximum flow problem (c.f., lecture Chapter 6 Part IV slide 6), the maximum flow is exactly κ provided that s, t are a "right pick", i.e., they are not in the MVC and in different connected components.

This means that we can stop the search after $\kappa+1$ augmentations at the latest, because if the flow is $\kappa + 1$, we would not have the right pair s, t. And if the flow is maximum (no augmenting path anymore) after at most κ augmentations, we have the right MVC. This means that computing a minimum s-t vertex cut takes $O(\kappa \cdot m)$ (instead of $O(n \cdot m)$ in general, c.f., lecture).

If we combine this with the result from part (a), where we showed that we have to make just $O(\kappa n)$ minimum s-t vertex cut computations, then the overall running time is $\mathrm{O}(m \cdot n \cdot \kappa^2)$.

(c) We employ a trick that is sometimes called exponential guessing. Let κ be the true size of a MVC. First we observe that if we guess a value for $\kappa' < \kappa$, then the algorithm above does not necessarily give a result that is correct or optimal (i.e., minimal) after doing the steps given in part (b) with κ'. We determine that $\kappa' < \kappa$ by trying to find another augmenting path after the we achieved a flow of κ' and if in fact $\kappa' < \kappa$ then we have to find one and the result was not optimal. The runtime for just one additional augmentation step is *asymptotically* the same. [1]

On the other hand, if we guess a value for $\kappa' \geq \kappa$ then this is no problem in terms of correctness, since we just try more possible pairs s, t and try to augment for longer. To sum it up: We know how to falsify a result for guessed values $\kappa' < \kappa$ and guessing $\kappa' \geq \kappa$ does not cause any issues.

[1] A bit more detailed (not expected from students) there are two issues from assuming a too small value $\kappa' < \kappa$. First, we will not be guaranteed to get a right pair s, t which fulfills the requirements stated in part (a) so that a minimum s-t vertex cut will not necessarily be a MVC. That means that the minimum s-t vertex cut could be larger than the actual MVC. Second, we will stop the augmentation process too early (c.f. part (b)). In both cases there would still be an augmenting path left which shows the incorrectness or non-optimality of the result.

Now all that is left is to guess increasing values for κ' and do the procedure as described above and then verify (or falsify) that we have the correct result. We start with $\kappa' = 1$ and then repeat by doubling $\kappa' = 2\kappa'$ as long as the result is still incorrect. Given that κ is the true size of a MVC, let $\kappa \leq 2^k < 2\kappa$, i.e., 2^k is the smallest power of 2 that is at least as big as κ. We have the correct result when we arrive at $\kappa' = 2^k$. Given that one run for a specific choice of κ' takes $\mathrm{O}(m \cdot n \cdot \kappa'^2)$ the overall runtime is (roughly, avoiding $\mathrm{O}(\cdot)$ in the calculation as long as possible)

$$\begin{aligned}
& m \cdot n \cdot 1^2 + m \cdot n \cdot 2^2 + \cdots + m \cdot n \cdot 2^{2k} \\
= \quad & m \cdot n \cdot \textstyle\sum_{i=0}^{k} 2^{2i} \leq m \cdot n \cdot \sum_{i=0}^{2k} 2^i = m \cdot n \cdot (2^{2k+1} - 1) \\
\leq \quad & m \cdot n \cdot 2^{2k+1} = m \cdot n \cdot 2 \cdot 2^{2k} = m \cdot n \cdot 2 \cdot (2^k)^2 \\
\leq \quad & m \cdot n \cdot 2 \cdot (2\kappa)^2 = m \cdot n \cdot 8 \cdot \kappa^2 = \mathrm{O}(m \cdot n \cdot \kappa^2).
\end{aligned}$$

(d) Let X be a MVC. Let G' be the unconnected graph induced by $V \setminus X$. We pick a node $v \in V$ that has degree δ (actually computing such a node is cheap by comparison, c.f., part (e)) and make a case distinction for $v \in X$ and $v \notin X$ (as suggested by the hint).

If $v \notin X$, then for a t that is in a different component of G' than v the minimum v-t vertex cut corresponds to the MVC X or to another MVC that is equally small. In this case we can simply compute minimum v-t vertex cuts for all possible $t \in V$ that are not neighbors of v, so we have $\mathrm{O}(n)$ pairs to check.

Now assume $v \in X$. We claim that at least two neighbors of v are in different connected components of G'. For a contradiction assume all neighbors of v are in X or in some connected component C of G'. But then removing $X' := X \setminus \{v\}$ from G already disconnects G into more than one connected component.

This is because, given a node $w \in C'$ where $C' \neq C$ is another connected component of G', then w will still be disconnected from C after removing only the smaller set X' from G as w did not share an edge with v (c.f., Figure 1). We have $|X'| = |X| - 1$, thus X was not an MVC, a contradiction.

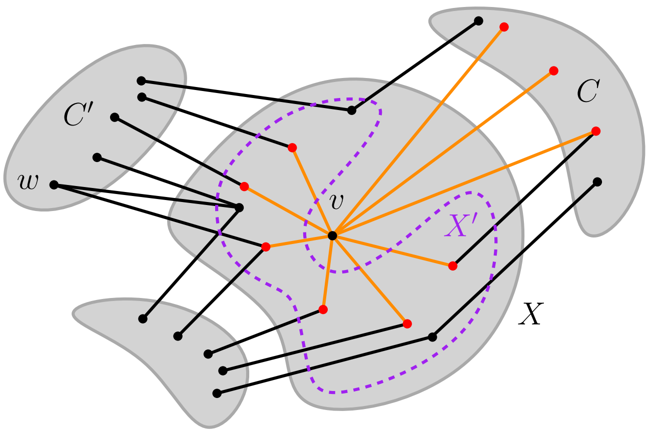

Figure 1: Set $X' := X \setminus \{v\}$ (purple) is a "smaller MVC" than X, a contradiction.

Consequentially, v must have two neighbors in different connected components of G'. Therefore we just have to compute minimum s-t vertex cuts for all possible pairs of neighbors of v in order to obtain a MVC. There are $\mathrm{O}(\delta^2)$ such pairs.

Since we do not know in advance which case is actually true, we run both proposed algorithms (i.e., we compute minimum v-t vertex cuts for the proposed pairs of nodes) and take the better result, i.e., the smaller vertex cut. Since one of the two cases must actually be true, we are guaranteed to obtain a MVC. In total we compute minimum v-t vertex cuts for $\mathrm{O}(n + \delta^2)$ pairs of nodes.

(e) The final runtime consists of bringing all the previous results together. A run of computing a minimum s-t vertex cut for just one pair $s, t \in V$ takes $\mathrm{O}(m\kappa)$ as outlined in part (b)+(c). Then we have to decide whether to go for the "pair-reduction" in (a) or the one in (d). We compare $n + \delta^2$ and κn and use the variant with fewer pairs. The overall running time is therefore

$$O(m\kappa \cdot \text{number of executions}) = O(m\kappa \cdot \min(n + \delta^2, n\kappa)).$$

Remark: Note that in case we use the variant from part (*d*)*, computing a node* v *with degree* δ *can be done in* $O(n + m)$ additional *time steps, which does not change the asymptotic runtime.*