Assignment-3

Anannya Mathur 2023SIY7565

Question 1

Let C= k-Min-Cut = minumum number of edges whose removal will partition the graph G into k separate components.

The random edge contraction algorithm of Karger and Stein terminates when there are two vertices left. Instead, the algorithm can be made to stop when there are k vertices remaining. The success probability can be computed as follows:
Let $|C| = \lambda$. Let $X \in V$ such that $|X| = k-1$. We can observe that upon the removal of all those edges whose both the endpoints lie in X and the edges whose one endpoint is in X and the other in X^C, we create k-1 connected components consiting of just one vertex, which correspond to vertices that are a part of set X, while the vertices in X^C will have created ≥ 1 connected components. Thus, the set of edges we removed corresponds to a k-cut. For any set X, the size of the set that contains edges $e \in E$ such that none of the endpoints of e lie in X $\leq m - \lambda$. Number of ways we can create a set X= $\binom{n}{k-1}$.For and edge e $\in E$, the size of a set such that none of the endpoints of e lie in X = $\binom{n-2}{k-1}$.
$\therefore \binom{n-2}{k-1}m \leq \binom{n}{k-1}(m-\lambda)$
$\implies \frac{\binom{n-2}{k-1}}{\binom{n}{k-1}} \leq 1 - \frac{\lambda}{m}$ = Probability that a random edge selection is not in C.

$\therefore$ Prob(preserving k-min-cut until k vertices left) $\geq \frac{\binom{n-2}{k-1}}{\binom{n}{k-1}}\frac{\binom{n-3}{k-1}}{\binom{n-1}{k-1}}....\frac{\binom{k}{k-1}}{\binom{k+2}{k-1}}\frac{\binom{k-1}{k-1}}{\binom{k+1}{k-1}} = \frac{\binom{k}{k-1}}{\binom{n}{k-1}}\frac{\binom{k-1}{k-1}}{\binom{n-1}{k-1}} = \frac{\binom{k}{k-1}}{\binom{n}{k-1}}\frac{\binom{k-1}{k-1}}{\binom{n-1}{k-1}} = \frac{k}{\binom{n}{k-1}}\frac{1}{\binom{n-1}{k-1}} \geq \frac{2}{n^{2k-2}}$.

If we run the algorithm $4n^{2k-2}logn$ times, Prob(k-min-cut fails to survive n-k random edge contractions) $\leq (1 - \frac{2}{n^{2k-2}})^{4n^{2k-2}logn} \leq \frac{1}{n^2}$. Hence, with a success probability of $1 - \frac{1}{n^2}$, a global k-min-cut can be computed in $O((n-k)n^{2k-1}logn)$.

Therefore, a 3 min-cut can be computed by plugging in the value of k as 3. With a success probability of $1 - \frac{1}{n^2}$, a global 3-min-cut can be computed in $O(n^6 logn)$.

Question 2

Given G and a matching M, the Edmonds algorithm finds odd length cycles and shrinks them into a supernode. Therefore, if the algorithm finds an alternating path, beginning with an unmatched edge and ending with an unmatched edge, from a supernode to itself, a new supernode can be formed as the path forms an odd length cycle. Each supernode $\in V_{odd}(T)$ has size one while every supernode $\in V_{even}(T)$ has odd size. We define every supernode by its base vertex. For supernodes having just one vertex, the base vertex is the vertex itself. If an alternating path begins at supernode W_o and cycles back to itself, the new supernode W will have its base defined as the base of W_o. It can be noticed that if the base vertex of a supernode is matched, the supernode is matched too. Instead of contracting odd-length cycles, we store for every vertex the base vertex of supernode it belongs to and the edge e_i used to enter the supernode. In case multiple supernodes are merge into one supernode, S, we store a list of sub-supernodes, S1, S2,....Si, so that we can change the status of edges in the event of an augmenting path discovery. If we encounter an unmatched edge e=(u,v) provided u, v do not belong to the same supernode, such that u, part of a supernode say W_i can be reached from a free supernode via an alternating path P=(e1,e2,...,ei), where e1 is unmatched while ei is matched, we consider the following cases for v: 1. v is a part of free supernode W_j. 2. v is a part of a matched supernode. Since for every vertex, we have the information stored regarding the supernode it belongs to and the edge e used to enter the supernode, we can backtrack using those edges along the two paths, one beginning with the supernode W_j, which contains the vertex v and the other beginning with the supernode W_i, which contains the vertex u. If we find the two paths terminating at unmatched supernodes A $\neq$ B, then we have found an augmenting path. In case, the paths converge at the same supernode, say C, a new supernode should be formed, which contains the list of all the sub-supernodes contained within it, which can easily be found as they lie on the path connecting W_i and C and W_j and C.
During this traversal in search of a new supernode or an augmenting path, every edge e=(u,v) that we encounter along the way should be treated in a similar fashion given that it satifies the mentioned condition. In case u and v belong to the same supernode, we discard the edge. The algorithm terminates when it can no longer find a valid edge.
For every edge that fulfills the condition, it will take at most O(n) number of supernodes to traverse. It can be noticed that the supernodes are disjoint in nature. Since we are using Union-Find data structure for both concatenating multiple supernodes into one and finding the supernode a vertex is present in, it will take O(mlogn) time for m operations on a disjoint set of n objects. Number of augmentations $\leq \frac{n}{2}$. Therefore, the implementation will take O(mnlogn).

In an M-alternating tree T under some maximum matching M in G, a Tutte-Berge maximiser is the set of vertices in G lying in $V_{odd}(T_{final})$, where T_{final} is the final tree obtained post all valid cycle contractions. These vertices represent the vertices in T_{final} that can be visited by a free vertex via a path that starts with an unmatched edge and ends with an unmatched edge, that is, an odd-length path. Therefore, the above solution can be further extended to output the base vertices of supernodes in the final graph that fulfill the condition of being reachable from a free supernode via an odd-length path. Such supernodes are guaranteed to be singleton sets consisting of just one vertex.

Question 3

Let us consider a bipartite graph G= (X,Y,E). There can exist a path from X to Y or Y to X, therefore, if there does exist a cycle in G, it has to be of even length.
Let us consider that G has two edge disjoint perfect-matching, say M and M'. $M \cap M' = \emptyset$, which implies that for any vertex x $\in X$, there exists an edge x,a under M and x,b under M', where a and b $\in Y$. Therefore, G is 2-regular. If we pick any vertex in G and follow one of the two edges to reach another vertex and keep traversing one of the two outgoing edges of every vertex we encounter, we will eventually reach the original vertex, thus, discovering one of the cycles. We can do the same for all the unvisited vertices to expose the other cycles present in G. These cycles are disjoint from the rest and their union covers all the vertices in G. Therefore, if G has 2-edge disjoint perfect matching, it has a disjoint cycle cover.
If G has a disjoint cycle cover, every vertex must have 2 edges connecting it to the other vertex so that we can follow one of the two edges to reach the other vertex and keep traversing and we are bound to cycle back to the original vertex. Therefore, G is 2-regular $\implies$ G has two edge disjoint perfect-matching.

2-edge disjoint perfect matching can be found by a single max-flow computation as follows: We can introduce two vertices s and t. We connect s with all the vertices in X and assign the capacity of these edges as 2. Similarly, we connect all the vertices in Y to t with the capacity of these edges as 2. To all the edges connecting vertices in X and Y in G, we assign the capacity as 1. We compute the (s,t)-min cut for the updated graph and if the computed max flow value λ= 2|X|=2|Y|, the original bipartite graph G indeed has 2-edge disjoint perfect matching, otherwise, it does not.