HW10B

Section II

Yanhao Wang 175002614

Zheng Qi 174003950

Mohan Xiao 174005865

9.6

Assume that the T_s is the minimun steiner tree, So we can just handle the tree like the approximation algorithm for the TSP,the meaning is that use each edge for twice so we can just have a loop. Now we can just start from one of the point of this loop and when the next vertex is not belong to V', we can just ignore them. So the final loop is a loop which only contains the vertices that belong to V'.

In this way, we can use triangle inequality rules that we have $d_{ik} < d_{ij} + djk$. If the T_s as the minimun striner tree have cost C.Then as follow the path above we can obtain a path which have cost 2C. So that means the cost of MST which only contain V' can still not bigger than the twice of the cost of minimun striner tree.

9.7

(1) When k = 2, this problem is can be soleved with min-cut using the max-flow method, which can be solved in polynomial time. (2) When k = 3, first find a min-cut between S_1 and S_2. Then remove the min-cuts, leaving S_1 and S_2 unconnected. At this time, S_3 can only be connected with on of the other two vertices, we apply the same method to find the min-cut between these two vertices and again remove the min-cuts. Then S_1, S_2, S_3 will be unconnected. This can be solved in polynomial time as well. (3) We can use a greedy algorithm for multiway cut. Find the minimum cut that splits G into two components by running minimum flow from each $s_i \in V$ to each $s_{-i} \in V$. Denote the two resulting components by G'_1, G'_2. Find the minimum cut dividing G'_1 or G'_2, yielding G'_3 (and a modified G'_1 or G'_2). Continue this process until all k terminal nodes are in their own component.

9.8

a. If max-SAT can be solved in poly time, through ALG of the max-SAT we can get a solution of a SAT instance, Where the max is the size of clauses in the SAT instance. If the max is not the size of the SAT, the SAT got a no answer.

b. For any clause j, the probability of this problem is satisified is 1 minus the probability of all the literals is false which is $1 - \frac{1}{2^{kj}}$. where k_j is the number of literals in this clause j.

So, if there are m clauses,

$$E = \sum_{i=0} m * P = \sum_{i=0} m * (1 - \frac{1}{2^{k_j}})$$

Which is obviously bigger than $\frac{m}{2}$, because every clauses have bigger or equal to 2 literals.

c. Because when we assign value to a literal we choose a value that can maximize satisfied number of the remaining clauses. Because there are only two values for a literal, follow this algorithm, we can assume that every time we get satisified number is bigger or equal to the unsatisified number of clauses. So the probability of this algorithm is bigger than 1/2.

9.9

a. We first show that any solution S that the algorithm outputs must have the property that $w(S) >= 1/2 * \sum_{c \in E} w(e)$. Because that the maximum size of the maximun cut can be $w(S) >= \sum_{c \in E} w(e)$, this means that an approximation ratio is 2

So, To prove that the $w(S) >= 1/2 * \sum_{c \in E} w(e)$ we just need to find the contridiction of $w(S) < 1/2 * \sum_{c \in E} w(e)$, For each vertex belong to V, we can simply define that c(v) is the total weight of edges connected to V which is cross the cut and W(v) is the sum of weights of all edge linked to v. So we can have

$$\sum_{v \in V} c(v) = 2 * w(S) < \sum_{c \in E} w(e) = 1/2 * \sum_{v \in V} w(v) \tag{1}$$

Because that we use the algorithm that which count each edge twice(from each point of the same edge).Thus,there must be at least one vertex v such that $c(v) < 1/2 * w(v)$. If v belongs to S, so we can just simply define $S' \ as \ S \cup (V)$, hence we can just get

$$w(S') >= w(S) - c(v) + (w(v) - c(v)) = w(S) + w(v) - 2 * c(v) > w(S) \tag{2}$$

which can simply make the algorthm not stop at S, which will make a contradiction to the assumeption.

b. At each iteration, we take linear time to find a S' exsits. And after each step the value of cut increases. The total number of iteration must smaller than $O(\sum_{e \in E} w(e))$. Thus the running time is $O(n \sum_{e \in E} w(e))$. It's not poly time. When all edges have weight 1, it's poly time.