CS180 Final

Siddharth Joshi

TOTAL POINTS

69 / 100

QUESTION 1

1 Problem 1 25 / 25

✓ **- 0 pts Correct. Used greedy algorithms and scan/remove from leafs.**

- 0 pts Correct DP solution

- 0 pts Correct. Modeled it as bipartite matching or network flow

- 5 pts Used Dynamic Programming and gave a recursion. However, the recursion cannot reflect how you make sure that the edges do not share nodes.

- 5 pts Did not indicate how do you construct two disjoint node sets and converted this problem to a bipartite graph matching problem. A simple approach is taking any node as the root and coloring nodes level by level.

- 10 pts Pointed out that we first solve the subproblem of the child node and merged the result. However, no correct DP recursion is given or you do not consider the two cases separately that the current root can be taken/not taken.

- 5 pts The way that you divide nodes into two sets is not right. A simple approach is taking any node as the root and coloring nodes level by level.

- 15 pts Misunderstood of the problem / gave not quite related solution. The correct solution is modeling the tree as a bipartite graph or using dynamic programming.

- 25 pts No answer

- 20 pts No clear algorithm

- 10 pts Used greedy algorithm but not started from leafs. The algorithms cannot always yield correct solution if you just start from a random node.

QUESTION 2

2 Problem 2 25 / 25

✓ **- 0 pts Correct**

- 25 pts It's not NP-complete

- 10 pts Only answered the feasibility question but did not give an algorithm that returns the assignment

- 8 pts Didn't consider the availability of classrooms

- 15 pts Modest attempt. Your algorithm may output no feasible assignment when there is indeed one

- 3 pts Did not describe how to assign edge capacity

- 3 pts Did not answer when there is or is not a feasible solution. (When max flow equals to what?)

- 3 pts Wrong edge capacity design

- 12 pts Correct direction but wrong algorithm. You need "perfect matching" but not a "matching"

- 20 pts No solution is given for (b) / No related is solution given for (b)

+ 5 pts Mentioned that we can use max flow/matching though no algorithm is given

- 5 pts It's not NP-complete. The algorithm is fine, though

- 0 pts Click here to replace this description.

QUESTION 3

3 Problem 3 5 / 25

- 0 pts Correct

- 0 pts (a) full credit

- 10 pts (a) wrong reduction

✓ **- 7 pts (a) Reducing from Hamiltonian cycle or tsp. but no discussion about the length of the cycle. no or wrong discussion about how to set the edge weights and no or wrong discussion about why the constructed edges weights satisfy the triangle inequality.**

✓ **- 5 pts (a) the edge weights are set improperly. The triangle inequality is broken.**

- 0 pts (b) full credit

- **3 pts** (b) reduction from euclidean TSP, but fail to explain the edge weight setting correctly.
- **5 pts** (b) wrong reduction or fail to explain how to set the edge weights
- **3 pts** (b) the reduction details are missing or wrong
- **0 pts** (c) full credit
- **5 pts** (c) Apply the MST idea with wrong MST details.

✓ **- 8 pts (c) The polynomial solution is missing or wrong. No explanation why it works.**

- **10 pts** (c) empty answer or wrong answer
- **5 pts** (c) confirm it as a NPC and give a MST-like solution
- **25 pts** Empty answer for three sub-questions
- **5 pts** (c) The polynomial solution is partially correct

QUESTION 4

4 Problem 4 14 / 25

- **0 pts** a. Correct

✓ **- 6 pts a. wrong recursion (formula)**

- **8 pts** a. no/irrelevant recursion (formula)
- **10 pts** a. No answer / irrelevant answer
- **0 pts** b1. Correct

✓ **- 0 pts b2. Correct**

- **3 pts** b1. Incorrect argument to avoid O(v^2 log v) total cost

✓ **- 5 pts b1. no answer to how to avoid O(v^2 log v) total cost**

- **5 pts** b2. no(wrong) explaination for log N rounds in total
- **6 pts** b2. algorithm lacks details
- **7 pts** b2. The algorithm lacks many details and no explanation for log N rounds in total
- **10 pts** b2. no answer

CS 180: Introduction to Algorithms and Complexity
Midterm Exam

Mar 19, 2019

Name	Siddhath Joshi
UID	105032378
Section	3D @ Rui Rui Li

1	2	3	4	Total

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

★ **Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.**

- The exam is a closed book exam. You can bring one page cheat sheet.
- There are 4 problems. Each problem is worth 25 points.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
- Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.

Name: UID: CS180 Final Exam

1. We have seen in class a polynomial time algorithm for maximum matching in bipartite undirected graphs. In general undirected graph, the problem is not *NP*-complete but the algorithm is quite involved. Suppose we take a tree and ask for maximum matching. Can you give a polynomial time algorithm? If you can, outline the algorithm. [25 pts]

maxMatch(Tree T):
→ Let T' be a copy of the tree
→ For every leaf (a node w no children ≡ a node w degree 1) in T that still exists in T':
- Add the edge connecting the leaf to its parent to matching
- Delete the parent and all its children (including the selected leaf) from T'

→ Recurse on T'

Time Complexity:
- Each recursive call has a for loop looping over the leaves that does O(1) work inside it ∴ for loop = O(|V|)
- determining the leaves of a tree requires running BFS from the root (can be chosen arbitrarily as a node w degree > 1) = O(|V|) for a connected tree

∴ every recursive call does O(|V| + |E|) work
- # of recursive calls = $\frac{1}{2}$ O(height of tree) as at every step 2 levels are matched.

in the worst case height = $O(|V|)$
$\therefore$ # of recursive calls = $O(|V|)$
hence total runtime = $O(|V|^2 + |V||E|)$

Proof of Correctness :-

→ Since every node in the tree has only one parent and some k children, it must be matched in the maximal matching to either its parent or its child.

→ However, every time a leaf is matched to its parent, also in the optimal maximum matching, every time an edge incident to a certain node is picked, no other edge incident on this node can be part of the maximum matching ⟹ the maximal matching in some sense must pair the levels of the graph (dist. from closest leaf) s.t. one parent matches to one in the next level just one above the leaves matches to just one leaf. But this is exactly the algorithm described earlier, hence the matching found by the aforementioned is maximal.

2. You are given a list of professors. Each professor P_i teaches C_{P_i} different classes, each of an hour, and submits C_{P_i} different hour intervals in which she wants to teach. She is indifferent to what class she teaches in each hour interval which she submitted. On the other hand we have a list of classrooms. H_{R_k} is the list of time intervals when each classroom R_k is available. We want to answer whether it is feasible for all professors' requests to be satisfied, and if it is, output the assignment. This problem is obviously in *NP* (why?).

   (a) Is the problem *NP*-complete? [5 pts]

   (b) If yes, prove it is *NP*-complete. If not, give a polynomial time algorithm to answer the feasibility question and output a feasible assignment if there is one. [20 pts]

a) No, as for it to be NP-complete it must be in NP AND also be in NP-hard (the set of problems atleast as hard as the hardest problems in NP)

b) Since I claimed that this problem is not NP-complete, it is possible for a deterministic polynomial time algorithm to exist. The idea of this algorithm is that the question above is after all just some bipartite matching from classes times (C_{P_i}'s) to hours that rooms are available, and thus we can use Ford Fulkerson's max flow to both find if this is possible (from the value of the max flow) and find the matching that makes this possible (the augmenting paths)

Algorithm (a directed graph)

→ Create a network G s.t. • there is a super-source vertex s and a super sink vertex t

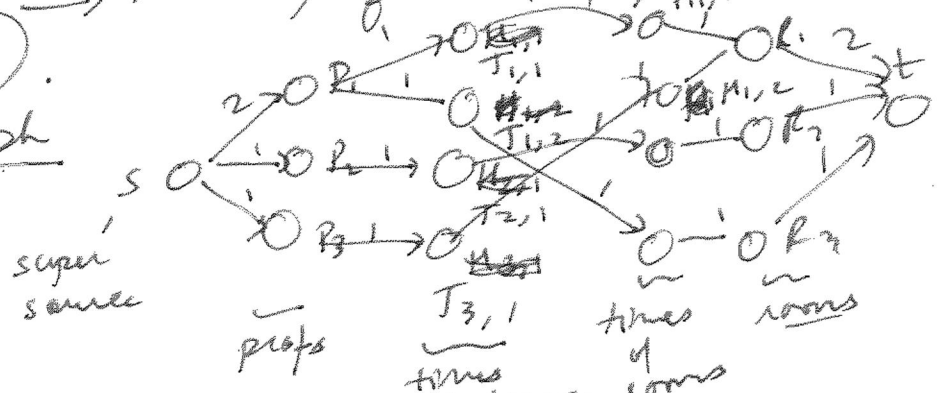
• create vertices representing each professor P_i

• Connect s to every professor P_i w an edge w capacity = C_{P_i} - the # of hour classes the professor teaches

• For each professor P_i's create vertices corresponding to the hour intervals she will teach during called $T_{i,j}$ and connect P_i to each its corresponding $T_{i,j}$'s w edges of capacity 1

- For each create vertices R_k for all k - representing the rooms and connect each R_k to the sink t w edges of capacity = cardinality of list H_{R_K} - the # of hour intervals the room could at most support is available for
- For every room R_k, create a vertex for every hour in its list H_{R_K} called labelled $H_{R_K,i}$, index in list and connect these vertices w an edge of capacity 1 to the corresponding room R_K
- Finally connect the $T_{i,j}$s - the hour slots that professors teach in to all corresponding $H_{R_K,i}$ s.t. the time they represent is the same (e.g. both indicate Monday 3-4pm) w edges of capacity = 1
- Run Ford Fulkerson's max flow algorithm on this, the augmenting paths indicate how to schedule the matching for the schedule. (if max flow < # total # of hours professors teach for ⇒ no perfect matching exists ⇒ no feasible schedule).

E.g. graph



Time complexity: $O(mn)$ where m = # of edges n = # of nodes (as Ford Fulkerson is bottlenecked).
since the construction of the

Correctness:
- the edges of capacity 1 into the $T_{i,j}$s ⇒ no class scheduled twice
- edges of capacity 1 from $H_{k,i}$s to Rooms ⇒ no time conflicts

Assumption is professors don't give conflicting times to teach classes at different times i.e. 1 prof can't teach 2 classes simultaneously

3. We have seen in class, by reduction from Hamiltonian cycle, that undirected TSP is *NP*-complete.

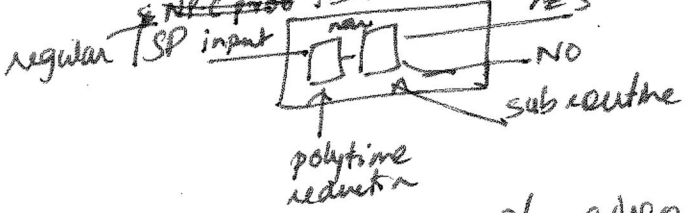
   (a) The Euclidean TSP (call in class mistakenly "planar") is a TSP problem where edge weights in the graph satisfy the triangle inequality ($\forall\ v_1, v_2, v_3,\ w\{v_1, v_2\} \leq w\{v_1, v_3\} + w\{v_3, v_2\}$). Prove that the Euclidean TSP is *NP*-complete. [10 pts]

   (b) We relax the condition on the (non-Euclidean) TSP that each city is to be visited only once. If the saleswoman goes to Chicago through Huston, she can fly back to Huston on the way to Miami (nevertheless, at the end she back to her city). Show that the relaxed-TSP is *NP*-complete (hint: do not ignore context). [5 pts]

   (c) We are now in the relaxed-TSP, and not only the weights do not satisfy the triangle inequality but also they are terribly skewed with respect to each other. Namely, the weights when ordered from low to high $w_1, w_2, \ldots$ satisfy that $w_i > \sum_{j=1}^{j=i-1} w_j$ for all i. Is the relaxed-skewed-TSP problem *NP*-complete? If not, give a polynomial time algorithm and argue the correctness of your algorithm. [10 pts]

a) → Proving that euclidean TSP ∈ NP is trivial: a simple NP algorithm for finding this would be a traversal of the graph by asking at every point which is the 'best' node to visit next since there are $O(n)$ nondeterministic decisions to be made and $O(n)$ choices the NP alg is in $O(n^2)$ ∴ euclidean TSP ∈ NP

Proving now that euclidean TSP ∈ NP-hard requires us to create a box for some NP-hard/NP-complete problem using euclidean TSP as a subroutine



By assumption that graph given to undirected TSP is complete (asked TA in exam), the reduction is as follows.

Reduction from undirected TSP → euclidean TSP

Inputs: a complete graph G and a max weight for the TSP i.e. k

→ Create G' = a copy of G and k' = k

→ for every edge (u,v) in G:

- min = +∞
- for every vertex w:
  - if (sum of weights of (u,w) and (w,v) < min)
    - → min = sum of weights of (u,w) and (w,v)
- if min < weight of (u,v):
  - set weight of (u,v) to min in G'
  - k' ← k' − (weight of (u,v) − min)

• Proof: • TSP $\leq k$ in $G \Rightarrow$ euclidean TSP $\leq k'$ in G'

if there exists a TSP $\leq k$ in G then the same cycle in G' ... can at worst have taken all edges w modified weight but $k' = k - \sum_{\text{all } e \in E}$ (weight of e in G - weight of e in G') ∴ the cycle can have at most weight k' in G' Hence.

• euclidean TSP $\leq k'$ in $G' \Rightarrow$ TSP $\leq k$ in G

same logic as above, The cycle found in G' has weight $\leq k$ in G

∴ time. undirected

Hence TSP $\leq_p$ euclidean TSP $\Rightarrow$ euclidean TSP $\in$ NP-H

∴ euclidean TSP $\in$ NP-Complete

---

b) relaxed-TSP $\in$ NP trivially in the same way shown in part a) w the condition of repeats being disallowed removed. ∴ R-TSP $\in$ NP.

∴ Proving relaxed-TSP $\in$ NP-Complete $\Rightarrow$ NP-Hard $\Rightarrow$ relaxed TSP $\in$ NP-Complete

Reducing euclidean TSP to relaxed TSP

For inputs G - a graph and k - a max size of euclidean TSP

Set $G' = G$ and $k' = k$ and ask relaxed TSP if there exists a relaxed TSP of size $\leq k'$

This trivial reduction works as the cycle found in G' using relaxed-TSP will never actually repeat a vertex as due to the triangle inequality it is always cheaper to go directly rather than repeat nodes in an optimal solution as $w(u,v) + w(v,w) \geq w(u,w)$ $\forall u,v,w$. Hence relaxed TSP $\Rightarrow$ euclidean TSP (as cycle doesn't repeat in relaxed) but euclidean tsp $\Rightarrow$ relaxed tsp - trivially as relaxed can always take a regular euclidean TSP cycle.

---

c) No, this problem is not NP-complete there is a poly-time

DP
1 to all

4. (a) In class we have seen the Bellman-Ford algorithm for one-to-all shortest paths with negative edges but no negative cycle. Write a recursion for shortest paths problem such that you can argue the recursion is amenable to dynamic-programming. And argue that the Bellman-Ford algorithm is in fact an iterative implementation of your recursion (Do not confuse Bellman-Ford with Floyd-Warshal which is all-to-all shortest paths algorithm). [10 pts]

(b) We said in class that the "idea" of an algorithm is manifested in its recursion.
Here's a recursion to solve MST: For each node v find the min weight edge adjacent to it. These chosen edges create a forest (why no cycle?). We take these edges to be in the MST. Now we "contract" all nodes incident to the same tree into a single "new node", which is connected to other "nodes" by original edges that connect a node in one tree to a node in *another* tree. All the intra-tree edges (edges aside from the tree edges that connect nodes in the same tree) are "gone." Notice that this might create "parallel edges" but that is ok.
We want to implement this recursion into an $O(|E|\log|V|)$ time algorithm. The implementation that Prof. Gafni knows requires that for each node the edges around it are ordered by weights. Alas, this looks like resulting in a cost of $O(|V|^2\log|V|)$ algorithm which is larger than $O(|E|\log|V|)$ for a sparse graph.

- Help get Prof. Gafni out of this conundrum. [5 pts]
- Outline an algorithm and argue it achieves the desired complexity (To find whether an edge is inter or intra tree its better be that all nodes in the same tree in the recursion are named the same. You want to argue that throughout the algorithm a node changes name at most $\log|V|$ time. Recall Union-find.) [10 pts]

same edge can be chosen from both sides..
is an edge intra or inter do find recursion a)

"commit"
union find
less body
- amortized time complexity argument

a) SP(u, v, G):
Return $\min_{edge(u,w)\in G}$ (wt of edge(u,w) + SP(w, v, G−(u,w)))

This is amenable to DP as there are a lot of subproblems being resolved — in total there are $O(n^2)$ subproblems

b) This solution is identical to bellman-ford that instead of directly finding the shortest intermediate path, does so eventually by all our updates.

Alg: → For each node v:
• pick min weight edge adjacent to it
• add these edges to MST
→ union all vertices adjacent to the tree rooted at v. for every tree in th
→ Recurse

6) → For every vertex v, find min edge adj. to it and ~~use~~ select it to be in the MST
→ Union every vertex with the root of trees ~~formed by steps~~ in the forest formed by step 1
→ Recurse on new Graph (combined vertices are no longer considered)

Time complexity: ~~$O(\log V)$~~
The amortized time complexity of this alg is $O(|E| \log |V|)$ as in total through the running of the algo (counting all recursions) ~~the~~ ~~over there can be at most~~ ~~$O(|E| \log V)$~~ a node can at most change names $O(\log V)$ times - as at every step ~~it must be combined to at least~~ every vertex is paired w/ at least 1 other vertex and ∴ in $O(\log |V|)$ steps ⇒ only 1 will remain. (as at every step $|V|$ is reduced by a factor 2 at least as $|V| \to \frac{|V|}{2}$ every recursive call). Similarly, ~~since~~ the amortized time of finding min edge adjacent to each node is $O(E)$ total runtime
$= O(|E| \times \log |V|)$