Homework 9

- Problems marked with **P** are graded on progress, which means that they are graded subjectively on the perceived progress toward a solution, rather than **solely** on correctness.
- Problems marked with a **G** are group problems. A group of 1-3 students may work on these together and turn in one assignment for the entire group. Each member listed should have made real contributions. The group problems will be turned in separately on gradescope.

For bonus and most extra credit questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. We strongly encourage you to typeset your solutions in LaTeX. If you collaborated with someone, you must state their name(s). You must write your own solution for all problems and may not look at any other student's write-up.

---

1. (20) A *linear system* $S_{M,n}$ consists a collection M of linear inequalities over variables $x_1, \dots, x_n$. Each *linear inequality* is of the form $a + b \geq 1$, where a and b are two variables chosen from $x_1, \dots, x_n$. For example, the following is a collection M of linear inequalities over variables x_1, x_2, x_3, which is denoted $S_{M,3}$:

$$x_1 + x_2 \geq 1$$
$$x_2 + x_3 \geq 1$$
$$x_2 + x_2 \geq 1$$

A *solution* to a linear system $S_{M,n}$ is an assignment of values in $\{0, 1\}$ to $x_1, x_2, \dots, x_n$ such that all inequalities are satisfied. We call $\sum_{i=1}^{n} x_i$ the *penalty* of the solution. Given a linear system $S_{M,n}$, the goal is to find a solution with the minimum penalty. For example, for the above system $S_{M,3}$, $(x_1, x_2, x_3) = (0, 1, 0)$ is a solution with the minimum penalty, 1.

We formalize the decision version of the problem as follows:

$$\text{LINEAR-INEQUALITY} = \{\langle S_{M,n}, k\rangle \mid S_{M,n} \text{ has a solution with penalty} \leq k\}.$$

Show that if LINEAR-INEQUALITY can be decided efficiently, then we can efficiently solve the related search problem, i.e., find a solution minimizing the penalty.

Solution: Given that Linear-Inequality can be decided efficiently, say we have some efficient decider D, which on input (S,k) returns whether or not S has a solution with a penalty less than or equal to k. We can solve the related search problem with an algorithm as follows:
Algorithm on input ($S_{M,n}$):
1. For k=0 till k=n run D on $(S_{M,n}, k)$ and if D accepts exit this for loop saving the value of k it accepted on.
2. For each $x_i \in S$ set all occurrences of x_i in S to 1 and then run D on $S_{M,n}, k - 1$. If D accepts then assign a value of 1 to x_i and otherwise, if D rejects, assign a value of 0 to x_i.
3.Return the assigned values of x_i as a solution to the minimum penalty of a satisfying arrangement for this system.

The first item invokes D at most n times and since D is efficient the loop completes in polynomial time. The second item invokes D n times and so therefore completes in polynomial times since D is efficient. Assigning values to each variable in S takes linear time and returning the values of the solution is linear so overall the solution is efficient with respect to the size of S. For any linear system S the algorithm will determine the minimum penalty, k, by testing each k value from 0 to n since n is the maximum penalty as k would equal n in the case every variable had to be assigned a 1. Once we have found the minimum penalty, k, we then know that the system is satisfied with k variables being assigned a value of 1. For each $x_i \in S$ we set all occurrences of the variable in the linear inequalities to 1 and then run D on that system and k-1. If in the solution of minimum penalty x_i has a value of 1 then D will accept because x_i was set to 1 in the minimum penalty solution and so therefore there were k-1 other variable that were set to 1, so in the case x_i is already set to 1 D will accept when k=k-1. If x_i was set to 0 in the solution of minimum penalty then k other vertices need to be set to 1, so if we set all occurrences of x_i to 0 and run D with k=k-1, D will reject because the minimum penalty solution had k other vertices set to 1 so the minimum penalty is k and D will not accept when k=k-1. Therefore we set all variables x_i to 1 when D accepts in the second loop and to 0 when D rejects in the second loop.

2. (20) In this question, we prove that the "Smart-Greedy" algorithm from lecture is a 1/2-approximation algorithm for the 0-1 Knapsack search problem.

   (a) Define the "fractional knapsack problem" as a variant of the original knapsack problem that allows one to take any *partial* amount of an item. That is, for an item of value v and weight w, one may take a fractional amount $t \in [0, 1]$ of the item and increase the value and weight of the backpack by $t \cdot v$ and $t \cdot w$, respectively. Prove that for this fractional variant, the optimal value we can achieve is at least as large as that for the original 0-1 variant (for the same weights, values, and knapsack capacity).

   Solution: Define OPT_{0-1} and OPT_{frac} to be the optimal value achieved by the 0-1 Knapsack problem and the fractional Knapsack problem respectively. If OPT_{0-1} has a total weight equal to the knapsack capacity then the fractional knapsack problem can replicate this solution by multiplying each item in the 0-1 solution by a factor of 1 in order to get a value of $OPT_{frac} = OPT_{0-1}$. In the case that OPT_{0-1} has a total weight less than the knapsack capacity, the fractional problem could use each element in this solution multiplied by a factor of 1 and then use the next best value to weight ratio element and multiply by a fractional amount a where weight*a = knapsack capacity - weight of OPT_{0-1}. Therefore the value of OPT_{frac} would be equal to $OPT_{0-1} + a$ * value of next best element. Therefore in either situation OPT_{frac} is either equal to or bigger than OPT_{0-1}.

   (b) In lecture, we defined the "Smart-Greedy" algorithm for the 0-1 Knapsack problem as follows:

      i. run the "Relatively-Greedy" algorithm,
      ii. run the "Dumb-Greedy" algorithm,
      iii. output the selection that achieves the most total value (breaking a tie arbitrarily).

Prove that the *sum* of the values obtained by the two sub-algorithms is at least the optimal value for the *fractional* knapsack problem. As in class, you are to assume that no object has a weight greater than the knapsack capacity.

Solution: The Relatively greedy algorithm will select as many items as possible with the largest value/weight ratio, starting with the highest value/weight ratios first. The fractional knapsack problem will begin operating the same way but then multiply the next highest value/weight ratio (some element s) by some constant a so that weight of s *a = knapsack capacity - weight of relatively-greedy solution. We then get that the fractional knapsack value will be greater than the relatively-greedy value by an amount $p =$ value of$s * a$. We know that p is not equal to the entire value of s (i.e. a=1 and therefore p=1 * value of s) because that would mean the entire weight of the item would of fit within the knapsack capacity and in that case it would of already been in the relatively greater solution. Since we now know that the fractional knapsack problem will be greater than the relatively greedy problem by an amount = p, we need to show that the dumb-greedy algorithm will obtain a value greater than or equal to p to show that the sum of the two algorithms is at least as big as the value for the fractional knapsack problem. In the case that the Dumb-Greedy algorithm selects s as its one element, the value of the Dumb-Greedy algorithm will be greater than p because we know that $p = a *$ value of s where a is greater than 0 and less than 1 so it will be a fractional amount of the value of s while the dumb-greedy value is the full value of s. In the case that the dumb-greedy algorithm selects an element that is not s we know that the value of this selected element is greater than the value of s since the dumb-greedy algorithm only selects the element with the highest value. In this case the value of the highest element is guaranteed to be greater than p as p is a fractional amount of a lesser value where as the dumb-greedy value is the full amount of the highest value. Therefore in every case the sum of the values obtained by the two sub-algorithms will be at least as big as the optimal value for the fractional knapsack problem.

(c) Using the previous parts, prove that "Smart-Greedy" is a 1/2-approximation algorithm for the 0-1 knapsack problem.

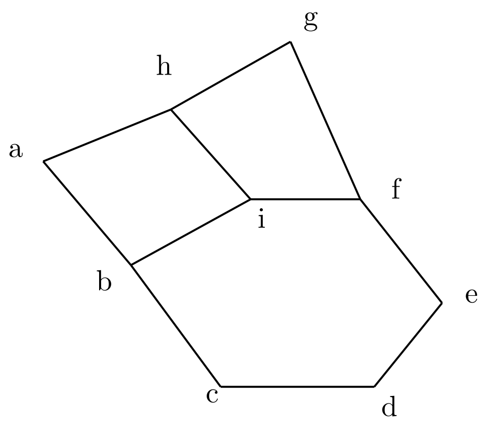
Solution: We need to show that $\frac{1}{2}OPT_{0-1} \leq value(\text{Smart-Greedy}) \leq OPT_{0-1}$. From part a we know that $OPT_{0-1} \leq OPT_{frac}$. From part B we know that $OPT_{frac} \leq$ the sum of the relatively greedy and dumb greedy algorithms. Since the sum of the two algorithms is at least as big as the value from fractional knapsack we know that the value of one of the algorithms is at least half the value of OPT_{frac} because since it is the sum of two items either both algorithms can have the same value where both must be equal to at least half of OPT_{frac} or one of the algorithms can be a smaller value than half of OPT_{frac} but then the other algorithm would have to have a value greater than half of OPT_{frac} in order for the sum of the two algorithms to be at least as big as OPT_{frac}. Since Smart-Greedy returns the larger of the two algorithm values we then can conclude that $\frac{1}{2}OPT_{frac} \leq value(\text{Smart-Greedy})$ Combining this with what we proved in part A we can then write $\frac{1}{2}OPT_{0-1} \leq \frac{1}{2}OPT_{frac} \leq value(\text{Smart-Greedy}) \leq OPT_{0-1}$

which reduces to $\frac{1}{2}OPT_{0-1} \leq value(\text{Smart-Greedy}) \leq OPT_{0-1}$ which proves that the smart-greedy algorithm is a 1/2 approximation algorithm for the 0-1 knapsack problem.

G 3. (20) Recall the "Double-Cover" algorithm discussed in lecture for 2-approximating a minimum vertex cover of a graph G:

```
1: $S \leftarrow \emptyset$
2: while $G$ still has edges do
3:     Pick an edge $e = \{u, v\} \in G$
4:     Remove all edges covered by $u$ and $v$ from $G$
5:     $S \leftarrow S \cup \{u, v\}$
```

(a) What's the smallest approximation ratio achievable when running the algorithm on the graph below? Explain with proof. Note: this problem is going to require some *ad hoc* (creative) reasoning. One thing that might be useful is to note that no vertex has a degree greater than 3.



Solution: The smallest vertex cover is 5 for this graph, this is because there is 11 edges total and suppose you pick the vertex that covers three edges, the remaining vertexes each cover two edges so for the remaining 8 edges that need to be covered one must pick at least 4 other vertices to give a total of 5 vertices in the vertex cover. The best case cover for the Double-Cover algorithm is 6 vertices which is achieved by selecting an edge that contains the vertex with the highest degree first. We can prove this is the best case cover for the Double-Cover algorithm by explaining why each lower value is impossible. For a double cover of 1,3, and 5 it is impossible because the algorithm only adds vertices to the cover two at a time. For a cover of 2 it is clearly impossible as the highest degree of a vertex is 3 and all other vertices have a degree of 2 and there are 11 edges to be covered. Therefore if we pick the edge containing two highest degree vertices of degree 3 the most edges we could cover is 5 so therefore a cover of 2 is impossible. A double cover of 4 is impossible because the highest degree for a vertex is 3 and even if it was possible to pick two edges containing 2 3-degree vertexes the most edges each pair of vertexes could cover is 5, for a total of 10 edges covered which is still less than the number of edges in the graph that need to be covered. Therefore 6 is the best case for the Double-Cover algorithm and the smallest approximation ratio is $\frac{6}{5}$

(b) Notice that, when there are multiple edges in G, we may choose any such edge on line 3 of the algorithm. Is it possible for two sequence of choices to result in vertex covers of different sizes? Prove your statement.

Solution: Yes this is possible and can be seen with the two following sequences:
Sequence 1:
Pick edge a-b then h-i then g-f then e-d to result in a cover of size 8.
Sequence 2:
Pick edge h-i then b-c then f-e to result in a cover of size 6.

(c) Give an example of a graph on which the algorithm *always* outputs a cover that's at least twice as large as the optimal one.

Solution: A graph where the algorithm will always output a cover that is at least twice as large as the optimal one is a graph with only two vertices and only 1 edge between them (essentially a straight line graph). The optimal cover for this kind of graph is always 1 because there is only one edge to be covered so it could be done by choosing one vertex on that edge. The algorithm would always output a cover of 2 because the algorithm only adds vertexes to the cover in pairs so both vertices in the graph would be added to the cover. Therefore the algorithm outputs a cover that is at least twice as large as the optimal one.

4. (20) Suppose there are n workers with skill-sets $S_1, \dots, S_n$, respectively. Let $U = S_1 \cup \cdots \cup S_n$ be the set of all skills possessed by the workers and, for each $u \in U$, let $I(u) = \{i : u \in S_i\}$ be the set of workers with skill u.

   Given that *each skill is possessed by at most f workers*, i.e., $|I(u)| \leq f$ for every $u \in U$, the goal is find a smallest team of workers that "covers" all the skills, i.e., a set of workers $C \subseteq \{1, \dots, n\}$ of minimum size such that $\bigcup_{i \in C} S_i = U$.

   Consider the following approximation algorithm for doing so.

```
1: $C = \emptyset$
2: while $U \neq \bigcup_{i \in C} S_i$ do          ▷ the selected workers do not yet cover $U$
3:     Pick a skill $u \in U \setminus (\bigcup_{i \in C} S_i)$      ▷ skill $u$ is not yet covered
4:     $C = C \cup I(u)$                  ▷ add all workers with skill $u$ to the cover
5: return $C$
```

Fix some arbitrary instance, and let C^* denote an optimal set cover for it. Let E denote the set of elements u chosen in Step 3 during a run of the algorithm, and let C denote the algorithm's final output.

(a) Prove that $I(u) \cap I(u') = \emptyset$ for any two distinct $u, u' \in E$.

Solution: We prove this with proof of contradiction. We know $u, u' \in E$ so we know that u and u' were chosen during step 3 of the algorithm and so therefore both are skills that are not yet covered. When u is selected in step 3 we then add all workers with that skill to the cover. For proof of contradiction we assume $I(u) \cap I(u') \neq \emptyset$ and so therefore there is some worker in I(u') that is also in I(u) however this itself is a contradiciton because if the worker was in I(u) then this set of workers would of already been added to the cover and since the worker is in I(u') the worker has the skill u' and therefore the skill is already covered and so therefore u' would never of been picked in step 3 and therefore $u' \notin E$. This is a contradiction because we know $u' \in E$ so therefore $I(u) \cap I(u') = \emptyset$ for any two distinct $u, u' \in E$

(b) Using part (a), prove that $|E| \leq |C^*|$.

Solution: $|E|$ is equal to the number of skills that have no workers in common with any other skills in E as shown in part a. Therefore each of these skills would need at least one worker to be in C* for a complete cover. We then know that C* must be at least as big as $|E|$ and further could be larger as there might be two workers with the same skills in which case if one was selected for the cover both of them would be selected for the cover since the algorithm works by adding all workers with a specific skill u to the cover. Therefore $|E| \leq |C^*|$

(c) Prove that $|C| \leq f \cdot |E|$, and conclude that the algorithm is an f-approximation algorithm for the problem.

Solution: The algorithm clearly adds all workers with each skill $u \in E$ to C. Since each skill is possessed by at most f workers, and since $|E|$ sets of workers with a specific skill are added to C the most workers that could be added to C is $f \cdot |E|$ and therefore $|C| \leq f \cdot |E|$. We can then use part B to get $f \cdot |E| \leq f \cdot |C^*|$ and then we can conclude that $|C*| \leq |C| \leq f \cdot |C^*|$ and since the algorithm runs in polynomial time we can conclude that the algorithm is an f-approximation algorithm.

(d) Prove or disprove: for every positive integer f, there is an input for which the algorithm necessarily outputs a cover that is exactly f times larger than an optimal one.

Solution: Disprove: Say $f = 2$ and we have three skills, skill 1, 2, 3. Say we have 3 workers each with following skills: worker A: 1, worker B: 2, worker C: 2,3. In this case the optimal solution is 2 and there is no input that could make the algorithm output a cover that is two times larger (or equal to 4) because there is only 3 workers, so the maximum cover size is 3.

5. (20) Define the languages:

$$\text{3-COLORABLE} = \left\{ \langle G = (V,E) \rangle : \begin{array}{c} \text{all the vertices in graph } G \text{ can be colored} \\ \text{with 3 colors } s.t. \text{ no two adjacent vertices} \\ \text{have the same color} \end{array} \right\}.$$

$$\text{SAT} = \{\langle \phi \rangle : \phi \text{ is a satisfiable Boolean formula}\}.$$

In this problem you are to show show that that 3-COLORABLE $\leq_p$ SAT without using the Cook-Levin Theorem. It has been broken into parts to make it easier for you to see the steps (parts a-c) involved. Part d is about the ramification of the result.

(a) Construct a mapping function f such that

$$\forall \langle G = (V,E) \rangle \in \text{3-COLORABLE}, f(x) \in \text{SAT}$$

$$\forall \langle G = (V,E) \rangle \notin \text{3-COLORABLE}, f(x) \notin \text{SAT}$$

which is computable in polynomial time. Describe the mapping f is pseudocode and provide justification that f is efficient.

Solution: Assume we have some efficient decider D for SAT.
mapping function f on input (G=(V,E)):
For all $(u, v) \in E$:
1.Construct 3 variables $p_{u,1}, p_{u,2}, p_{u,3}$ and $p_{v,1}, p_{v,2}, p_{v,3}$ for each vertex respectively that correspond to the three possible colors 1, 2, 3.
2. Arrange the variables in the formation $\phi_t = \neg(p_{u,1} \wedge p_{v,1}) \wedge \neg\ (p_{u,2} \wedge p_{v,2}) \wedge \neg(p_{u,3} \wedge p_{v,3})$
3. For each vertex u and v create $\phi_v = (p_{v,1} \vee p_{v,2} \vee p_{v,3}) \wedge \neg\ (p_{v,1} \wedge p_{v,2}) \wedge \neg(p_{v,2} \wedge p_{v,3}) \wedge \neg(p_{v,1} \wedge p_{v,3})$
4. Do $\phi = \phi \wedge \phi_t \wedge \phi_v \wedge \phi_u$
Run D on input (ϕ) and return as it does. F is efficient because the loop runs $|E|$ times and inside of it does linear work creating variables for the boolean formula so overall the loop runs efficiently. Additionally, we assumed that D runs efficiently so the call to D on ϕ should run efficiently and therefore the overall function runs in polynomial time and therefore is efficient.

(b) Show that $\forall \langle G = (V,E) \rangle \in$ 3-COLORABLE, $f(x) \in$ SAT.

Solution: $\forall \langle G = (V,E) \rangle \in$ 3-COLORABLE, $f(x) \in$ SAT because step 3 of the algorithm ensures that every vertex in graph G is colored with one and only one color and step 2 ensures that for every edge in graph G the two vertexes on it will not be the same color. Therefore every graph in 3-COLORABLE will have a satisfying arrangement to every step 3 clause added to the overall clause as every vertex will have one and only one color assigned to it, while also having a satisfying arrangement to every step 2 clause added to the overall clause because every edge will not connect two vertices of the same color. Since these clauses are all connected with and symbols, since every single one will be true SAT will accept.

(c) Show that $\forall \langle G = (V, E) \rangle \notin$ 3-COLORABLE, $f(x) \notin$ SAT.

Solution: $\forall \langle G = (V, E) \rangle \notin$ 3-COLORABLE, $f(x) \notin$ SAT because step 3 of the algorithm ensures that every vertex in graph G is colored with one and only one color and step 2 ensures that for every edge in graph G the two vertexes on it will not be the same color. Therefore every graph not in 3-COLORABLE will have at least one clause from step 2 or 3 that does not have a satisfying arrangement since for the graph not to be in 3-COLORABLE it must either not have every vertex be assigned 1 and only 1 color or it must be impossible to assign colors in a way that every edge connects two different colored vertexes. Since these clauses are all connected with and symbols and since at least one will be false SAT will reject.

(d) Suppose there is an efficient decider for SAT, called D_{SAT}. Show that, in this case, we can efficiently solve the 3-COLORABLE search problem. In other words, if the graph is 3-colorable, we can give a valid coloring using only 3 colors. Otherwise, we output "The graph is not 3-colorable".

Solution: We can construct an algorithm to solve the 3-COLORABLE problem using the decider D_{SAT} as follows:
On input (G=(V,E)):
1. Use the mapping function f(x) to map the graph to a boolean formula ϕ and then run D_{SAT} on ϕ If D_{SAT} rejects then print "The graph is not 3-colorable" and quit
2. For each $v \in V$: run D_{SAT} on ϕ but set all instances of $p_{v,1}$ to 1. If D_{SAT} accepts then vertex v is color 1 and continue with next loop iteration. ELSE change all instances of $p_{v,1}$ back to 0 run D_{SAT} on ϕ but set all instances of $p_{v,2}$ to 1. If D_{SAT} accepts then vertex v is color 2 and continue with next loop iteration. ELSE change all instances of $p_{v,2}$ back to 0 run D_{SAT} on ϕ but set all instances of $p_{v,3}$ to 1.If D_{SAT} accepts then vertex v is color 3 and continue with next loop iteration. This algorithm is efficient because we already know the mapping function is efficient and we are assuming that D_{SAT} is also efficient and the loop in this algorithm calls D_{SAT} at most $|V| \cdot 3$ times so therefore the entire algorithm runs in polynomial time and is therefore efficient. This algorithm is correct because if the graph is not colorable then D_{SAT} will reject and so we print the message and then exit the algorithm. For any graph that is colorable then the boolean formula will be satisfiable (i.e. D_{SAT} will accept) for every vertex when it is set to at least one of the colors.

6. **Bonus:** Recall the language:

$$\text{CLIQUE} = \left\{ \langle G, k \rangle : \begin{array}{c} G \text{ is an undirected graph that has a } k\text{-clique} \\ \text{subgraph} \end{array} \right\}.$$

Suppose that there exists a "black box" D that efficiently decides CLIQUE. Describe (with proof) an efficient algorithm that, given an undirected graph G, uses D to output a clique in G of maximum size.

Solution: