Homework 5

Sample Solution

Due Date: Thursday, June 21, 11:59 pm

Directions: Your solutions should be typed and submitted as a single pdf on Gradescope by the due date. LATEX is preferred but not required. If you use another editor for your solutions (such as Microsoft Word), you should convert the final document to a pdf, confirm that the symbolic math from the equation editor is properly formatted, and submit the pdf. Answers should be clear and concise; show your work but do not write paragraphs where a sentence suffices. Algorithms should be written in pseudocode or precise sentences that clarify details. Partial credit will be awarded for incorrect answers that are close and demonstrate understanding of the problem.

Discussion among students is permitted, but students must write up solutions independently on their own. In particular, you should understand and be able to explain everything that you write in your own words. No materials or sources from the Internet (except the course textbook) can be consulted at any point. Failure to adhere to these policies will result in referral to the Office of Student Conduct.

Problem 1: Partition Problem [15 points]. In the partition problem, we are given a set \( X = \{x_1, x_2, \ldots, x_n\} \) of integers and asked to determine whether there exists a partition \( P \subseteq X \) such that \( \sum_{x_i \in P} x_i = \sum_{x_i \not\in P} x_i \). That is, we are asked to determine whether there is a partition of \( X \) into two sets such that the sets have exactly the same sum. Show that the partition problem is NP-complete. [hint: you should reduce from one of the problems shown in Figure 8.7 of DPV on page 247]

Solution 1. First note that the partition problem is in the class \( NP \). To see this, observe that a proposed solution \( P \) can be verified in polynomial time by simply computing \( \sum_{x_i \in P} x_i \) and \( \sum_{x_i \not\in P} x_i \) in \( O(n) \) time.

To prove NP-hardness, we reduce from the Subset Sum problem. In the subset sum problem, we are given a set \( Y = \{y_1, y_2, \ldots, y_m\} \) of non-negative integers and a target \( t \), and are asked to determine whether there is a subset \( S \subseteq Y \) such that \( \sum_{y_i \in S} y_i = t \). Given such an instance of the subset sum problem, we reduce it to an instance of the partition problem as follows: Let \( X = Y \cup \{s + t, 2s - t\} \) where \( s = \sum_{i=1}^{n} y_i \), i.e., the sum of all the integers in \( Y \).

First, suppose that \( S \) is a solution to the subset sum problem on \( Y \) with target \( t \). Then note that there is a solution \( P \) to the partition problem on \( X \) by taking \( S \cup \{2s - t\} \), since the total value of this set must be \( 2s \), and the total value of \( X \setminus (S \cup \{2s - t\}) \) is \((s - t) + (s + t) = 2s\).

Second, suppose that \( P \) is a solution to the partition problem on \( X \). Then we can find a solution to the subset sum problem on \( Y \) with target \( t \). To see this, note that the sum of all the elements of \( X \) is \( s + (s + t) + (2s - t) = 4s \), so the sum of elements in \( P \) must equal \( 2s \) (and the same for the elements not in \( P \)). Since the two elements we added sum to \((s + t) + (2s - t) = 3s\), exactly one of them must be in \( P \), and one of them not in \( P \). Without loss of generality, suppose \( 2s - t \in P \). Then the remaining elements in \( P \) must sum to \( t \), and thus constitute a solution to the subset sum problem on \( X \).

Finally, note that all of the reduction is polynomial time: we just need to sum over elements in \( Y \), and then check which side of the partition contains \( 2s - t \).
Problem 2: Maximum Common Subgraph (Taken from DPV 8.15) [15 points]. In the maximum common subgraph problem, we are given two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, and a budget $b \in \mathbb{N}$. We want to find two sets of vertices $V'_1 \subseteq V_1$ and $V'_2 \subseteq V_2$ such that deleting $V'_1$ in $G_1$ and $V'_2$ in $G_2$ leaves at least $b$ vertices in each graph, and makes the two graphs identical. Show that the maximum common subgraph problem is NP-complete. [hint: you should reduce from one of the problems shown in Figure 8.7 of DPV on page 247]

Solution 2. First note that the maximum common subgraph problem (MCSP hereafter) is in the class $NP$. To verify a given solution $V'_1$ and $V'_2$ along with a proposed isomorphism mapping $V'_1 \setminus V'_1'$ to $V'_2 \setminus V'_2'$, we simply need to delete these sets of vertices and their incident edges from the graphs $G_1$ and $G_2$ and check whether the proposed isomorphism is indeed bijective, which can be done in $O(n^2)$ time. This is an example of a problem with a less trivial step to prove that it is in NP, and you will not be penalized if your solution is left somewhat informal.

To prove NP-hardness, we reduce from the clique problem. In the clique problem, we are given a graph $G = (V, E)$ and a goal $g$, and are asked to find a clique of size $g$ in $G$. A clique of size $g$ is a set of $g$ vertices such that there is an edge between every pair of vertices in the set. Given an instance of the clique problem, we reduce to an instance of the MCSP as follows: create a complete graph $G'$ which has the same vertices as $G$ but has an edge between all pairs of vertices. The two input graphs to our maximum common subgraph problem are $G$ and $G'$.

Suppose that $V'_1 \subseteq G$ and $V'_2 \subseteq G'$ is a solution to the MCSP on $G$ and $G'$. If $|V| - |V'_1| \geq g$, then there is a clique of size $g$ in $G$. Since $G'$ is a complete graph, any subgraph must also be a complete graph, and thus a clique, so if there is a common subgraph of $G$ and $G'$ of size at least $g$, then this must be a clique of size at least $g$. In particular, $V \setminus V'_1$ is a clique of size at least $g$ (and one can trivially throw away vertices to get a clique of size exactly $g$). Otherwise, there is no clique of size $g$ in $G$. Conversely, suppose that there is a clique $V'$ of size $g$ in $G$. Then clearly it forms a common subgraph of size $g$ with any subgraph of $G'$ of size $g$. The reduction is in polynomial time because it just requires creating a complete graph with a polynomial number of edges and vertices with respect to $G$.

Problem 3: Greedy Knapsack [20 points]. In the knapsack problem, we are given $n$ items labeled $1, \ldots, n$, with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$. We also have an overall weight constraint $W$. We want to find a subset of the items that maximizes total value subject to the constraint that the total weight of the items is no more than $W$. We gave a dynamic programming solution to the knapsack problem that finds an optimal solution in $O(nW)$ time, but this is not quite polynomial running time since $W$ can be represented with $\log_2(W)$ bits. In this problem, we consider developing an $O(n \log(n))$ running time approximation algorithm for knapsack.

(a) First, consider the following greedy scheme: sort the items by decreasing value to weight ratios ($v_i/w_i$). Now, simply iterate through the items in this order, adding an item to the knapsack if doing so does not violate the total weight constraint. Prove that this algorithm can have an arbitrarily bad approximation ratio: that is, for any $\alpha > 1$, give an instance of the knapsack problem where this algorithm has approximation ratio $\alpha$ [5 points].

(b) A simple modification to the above algorithm yields a 2-approximation. In particular, this modification should take the best of two possible solutions: the one computed by the above greedy algorithm, and some other very simple solution. State what this other solution is, and then prove that the approximation ratio of the resulting algorithm (taking the best of the two solutions) has an approximation ratio of 2 [15 points]. [Hint: you should choose the other simple solution to fix the bad example from part a].

Solution 3.

(a) Consider an instance of the knapsack problem with total weight constraint $W = 1$ and two goods. One item has weight and value of one ($w_1 = v_1 = 1$). Let $\alpha > 1$. The other item has $v_2 = 1/\alpha$ with weight $w_2 = 1/2\alpha$. Then the second item has value to weight ratio of 2, so our greedy algorithm packs it first,
and subsequently cannot pack item 1. So we get total value of $1/\alpha$, when just packing the first item would have given a total value of 1. Therefore, for any $\alpha > 1$, this shows an instance where the algorithm has approximation ratio at least $\alpha$. In other words, the approximation ratio is unbounded.

(b) Compute two solutions. Consider the items ordered by decreasing value to weight ratio so that $v_i/w_i \geq v_j/w_j$ if and only if the second of its endpoints we consider is placed in the opposite set as its first endpoint. Every graph $G$ problem a, with total value $V(X) = \sum_{i=1}^k v_i$. Let $Y = \{k+1\}$ be the solution that takes only the first item in the decreasing $(v_i/w_i)$ order that was not chosen by the greedy algorithm (and thus is not in $X$), because it would have violated the total weight constraint, with total value $V(Y) = v_{k+1}$. We choose whichever of $X$ or $Y$ has the larger total value.

Let $OPT$ be the maximum total value of a feasible solution to the given knapsack problem with total weight constraint $W$. We want to prove that $\max (V(X), V(Y)) \geq OPT/2$. To do this, we will argue that $OPT \leq V(X) + V(Y)$.

Let $OPT'$ be the optimal total value of a feasible solution to the same knapsack problem but with larger weight constraint $W' = \sum_{i=1}^{k+1} w_i$. $W' > W$, since otherwise the greedy algorithm would have included item $k+1$ in $X$. Therefore, $OPT' \geq OPT$ (since the same problem with larger weight constraint can only be easier). But note that $X \cup Y$ is optimal for the instance with total weight $W'$, since it uses all of the weight $W'$ and has the optimal value to weight ratio. So

$$V(X \cup Y) = V(X) + V(Y) = OPT' \geq OPT$$

By the pigeonhole principle, since $V(X) + V(Y) \geq OPT$, at least one of $V(X)$ or $V(Y)$ must be at least $OPT/2$. So in particular, $\max (V(X), V(Y)) \geq OPT/2$.

**Problem 4: Approximation Algorithm for Max-Cut [20 points].** Given an undirected unweighted graph $G = (V, E)$, the maximum cut problem asks for a partition of $V$ into sets $A$ and $B$ (partition meaning $A \cap B = \emptyset$ and $A \cup B = V$) so that the number of edges between these sets is maximized. Give an efficient approximation algorithm that achieves an approximation ratio of 2, that is, the algorithm should compute a partition with at least 1/2 times as many crossing edges as the optimal partition. Show that your algorithm achieves this approximation ratio, and state the running time of your algorithm. [Hint: Consider a greedy or local search procedure. You may find the handshaking lemma useful in your analysis: $\sum_{v \in V} d(v) = 2|E|$ where $d(v)$ is the degree of $v$].

**Solution 4.** Let $edges(A, B)$ count the number of edges between vertices in $A$ and those in $B$. Here, I give a greedy procedure that builds the cut from the ground up. It is also possible to solve the problem with a local search procedure using the handshaking lemma.

```
1: procedure MAXCUT(G)
2:   A, B ← ∅
3:   for v ∈ V do
4:     if $edges(A \cup \{v\}, B) \geq edges(A, B \cup \{v\})$ then
5:       A ← A \cup v
6:     else
7:       B ← B \cup v
8:   end if
9: end for
10: Return A, B
11: end procedure
```

We want to show an approximation ratio of 2. For every edge $e \in E$, $e$ is a cut edge in the final partition if and only if the second of its endpoints we consider is placed in the opposite set as its first endpoint. Every
time we add a vertex $v$ to one of the partitions, let $|e_v|$ be the number of edges for which $v$ is the second endpoint considered. Since every edge has exactly 1 vertex which will be the second endpoint we consider, note that $\sum_{v \in V} |e_v| = |E|.$

Consider the point at which we encounter a vertex $v$ in the algorithm. There are $|e_v|$ edges for which the placement of $v$ decides whether they are cut edges so so there must be a set into which we can add $v$ to add at least $|e_v|/2$ cut edges by the pigeonhole principle. So our greedy choice contributes at least $|e_v|/2$ edges to the cut, otherwise we would have placed $v$ in the other set of the partition. But then the size of our final cut is at least $\sum_{v \in V} |e_v|/2 = |E|/2.$ Since any optimal solution can contain no more than all $|E|$ of the edges, this means that the greedy algorithm finds at least half as many cut edges as the optimal solution.