CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani

# HW Solution 8

Due March 14, 11:59pm

**1. (20 pts.) Sugar Water Enterprises** Your parents have just left for Costa Rica and have, surprisingly, left you in charge of their beverage company, Sugar Water Enterprises. Unfortunately, they failed to decide on a production plan before they left, and so you must do so now. Your company uses sugar, which costs 10 cents per gram, caffeine, which costs 20 cents per milligram, and water, which costs 1 cent per ounce to produce its beverages. There are two beverages that your company produces, Zap Energy and the signature Sugar Water. One case of Zap Energy is made with 5 grams of sugar, 1 milligram of caffeine and 8 ounces of water. One case of Sugar Water is made with 12 grams of sugar and 16 ounces of water. Your Sugar Water machine can't produce more than 80 Sugar Water cases in a day, and your Zap Energy machine can't produce more than 50 Zap Energy cases in a day. You have a budget of $b$\$ that can be spent on materials. Zap Energy sells for 5\$ per case, and Sugar water sells for 2\$ per case. The goal is to maximize your profit in one day.

(a) Create a linear program to represent this problem.

(b) Find the dual of the linear program.

(c) Find the optimal solution as a function of $b$ (assume you can produce non-integer numbers of cases and can purchase items in non-integer quantities).

**Solution:**

a) Without loss of generality, we can assume that no excess items are purchased (only materials that are used are purchased, as this is optimal).

Cost of each Zap Energy case: $5 \cdot 0.1 + 1 \cdot 0.2 + 8 \cdot 0.01 = 0.78$; profit: $5 - 0.78 = 4.22$.
Cost of each Sugar Water case: $12 \cdot 0.1 + 16 \cdot 0.01 = 1.36$; profit: $2 - 1.36 = 0.64$.

LP:

$$\begin{aligned} \max \quad & 4.22z + 0.64s \\ \text{subject to:} \quad & z \leq 50 \\ & s \leq 80 \\ & 0.78z + 1.36s \leq b \\ & z, s \geq 0 \end{aligned}$$

b) Multiplying by $y_1, y_2, y_3$ yields:

$$\begin{aligned} \max \quad & 4.22z + 0.64s \\ \text{subject to:} \quad & (y_1)z \leq (y_1)50 \\ & (y_2)s \leq (y_2)80 \\ & 0.78(y_3)z + 1.36(y_3)s \leq (y_3)b \\ & z, s \geq 0 \end{aligned}$$

This yields the dual:

$$\begin{aligned} \min \quad & 50y_1 + 80y_2 + by_3 \\ \text{subject to:} \quad & y_1 + .78y_3 \geq 4.22 \\ & y_2 + 1.36y_3 \geq 0.64 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

(Additional explanation:
recall that the dual is trying to upper bound the primal's objective - the profit. The dual's objective, $50y_1 + 80y_2 + by_3$, replaces all the primal constraints by the combined constraint:

$$50y_1 + 80y_2 + by_3 \geq 4.22z + 0.64s. \tag{1}$$

What we know from multiplying and adding the primal constraints is only that:

$$50y_1 + 80y_2 + by_3 \geq \big(y_1 + .78y_3\big)z + \big(y_2 + 1.36y_3\big)s.$$

Therefore, in order to guarantee (1) we need to impose the dual constraints: $y_1 + .78y_3 \geq 4.22$ and $y_2 + 1.36y_3 \geq 0.64$.)

c) We note that all other things equal, it is better to produce Zap Energy if we can, as it costs less to make and earns a great profit per unit. Secondarily, we will want to make sugar water, as it does earn a profit, rather than making nothing. Finally, if our budget is so high that we can produce both Zap Energy and Sugar Water to the maximum, we will then use the budget for nothing else, and earn no extra profit. Thus, our optimal value is as follows:

$4.22\frac{b}{0.78}$ for $b \leq 39$
$211 + 0.64\frac{b-39}{1.36}$ for $39 < b \leq 147.8$
262.2 for $b > 147.8$

The corresponding solution, of form $(z, s)$, is:
$(\frac{b}{0.78}, 0)$ for $b \leq 39$
$(50, \frac{b-39}{1.36})$ for $39 < b \leq 147.8$
$(50, 80)$ for $b > 147.8$

**2. (20 pts.) Zigzag sequences**

(a) **(10 pts.)** A zig-zagging sequence is a sequence of numbers, $a_1, \ldots, a_k$ where $a_1 > a_2 < a_3 \cdots$ or $a_1 < a_2 > a_3 \cdots$: for example, $[3,2,4,3,5]$ is a zig-zagging sequence.

Give a subproblem definition, recurrence, and base cases for your subproblem for a dynamic programming algorithm for finding the length of the longest zig-zagging subsequence in a sequence. Also, state how you would output a final answer.

(Note that $[1,9,7,10,4]$ is a zig-zagging subsequence of $[1,5,9,8,7,10,4]$.)

The running time is not needed, but the associated dynamic programming algorithm should be polynomial.

(b) **(10 pts.)** Given two sequences, find the length of the longest common zig-zagging subsequence.

**Solution:**

(a) Notice that this problem is very similar to the longest increasing subsequence in the textbook. The key insight is that we need our subproblems to distinguish between two different possibilities for subsequences ending at the $i$-th element: "ends at $a_i$ and increasing on the next step" and "ends at $a_i$ and decreasing on the next step".

Let $Z(i,0)$ denote the longest zig-zagging subsequence using the prefix $a_1, \ldots, a_i$ which ends with $a_i$, and expects the next number to be greater than $a_i$. Similarly, let $Z(i,1)$ denote the longest zig-zagging subsequence using the prefix $a_1, \ldots, a_i$ which ends with $a_i$, and expects the next number to be less than $a_i$.

We define our base case to be $Z(0,0) = Z(0,1) = 0$, and we define our recurrence to be

$$Z(i,0) = \max_{k<i \text{ AND } a_k > a_i} Z(k,1) + 1$$
$$Z(i,1) = \max_{k<i \text{ AND } a_k < a_i} Z(k,0) + 1.$$

To return the length, we can simply take

$$\max_i(\max(Z(i,0), Z(i,1)))$$

as our solution.

Notice that even though the space complexity is $\Theta(n)$, the running time is $\Theta(n^2)$ because at each step of the recurrence we take a maximum over $\Theta(n)$ elements.

Bonus: there is also a greedy linear time algorithm - can you find it?

(b) Notice that this problem is very similar to the longest common increasing subsequence from last week's homework. The key insight again is that we need our subproblems to distinguish between two different possibilities for subsequences ending at $a_i = b_j$: "ends at $a_i = b_j$ and increasing on the next step" and "ends at $a_i = b_j$ and decreasing on the next step".

Whenever $a_i = b_j$, let $Z(i,j,0)$ denote the longest common zig-zagging subsequence using prefixes $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$, ending with $a_i = b_j$, and expecting the next number to be greater than $a_i$. Similarly, let $Z(i,j,1)$ denote the longest common zig-zagging subsequence using prefixes $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$, ending with $a_i = b_j$, and expecting the next number to be less than $a_i$. (If $a_i \neq b_j$, set $Z(i,j,r) = 0$.)

We define our base case to be $Z(0,0,0) = Z(0,0,1) = 0$, and we define our recurrence to be

$$Z(i,j,r) = \begin{cases} 0 & a_i \neq b_j \\ 1 & \max_{k<i,\ell<j} Z(k,\ell,1-r)+1, \end{cases}$$

where if $r = 1$, we take the maximum only over $k,\ell$ such that $a_k = b_\ell < a_i$, and if $r = 0$, only $a_k = b_\ell > a_i$ are considered (if no such $k,\ell$ exists, we just set $Z(i,j,r) = 1$). To return the length, we can simply take

$$\max_{i,j}(\max(Z(i,j,0),Z(i,j,1)))$$

as our solution.

**3. (20 pts.) Reconstructing evolutionary trees by maximum parsimony**

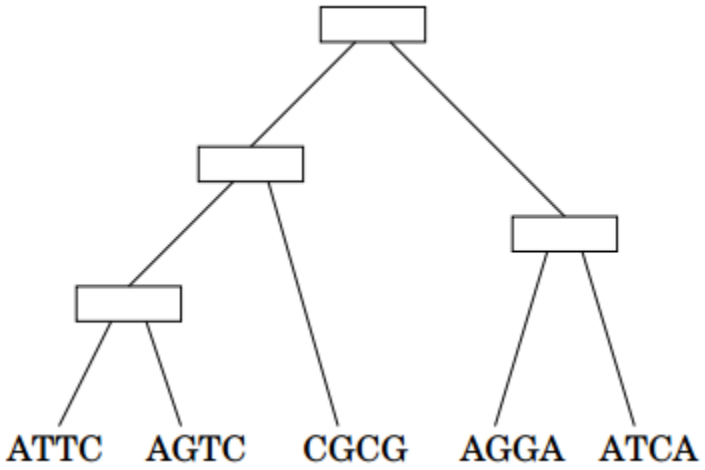
Suppose we manage to sequence a particular gene across a whole bunch of different species. For concreteness, say there are $n$ species, and the sequences are strings of length $k$ over alphabet $\Sigma = \{A,C,G,T\}$. How can we use this information to reconstruct the evolutionary history of these species? Evolutionary history is commonly represented by a tree whose leaves are the different species, whose root is their common ancestor, and whose internal branches represent speciation events (that is, moments when a new species broke off from an existing one). Thus we need to find the following:

- An evolutionary tree with the given species at the leaves.
- For each internal node, a string of length $k$: the gene sequence for that particular ancestor.

For each possible tree $T$, annotated with sequences $s(u) \in \Sigma^k$ at each of its nodes $u$, we can assign a score based on the principle of parsimony: fewer mutations are more likely.

$$score(T) = \sum_{(u,v)\in E} (\text{number of positions on which } s(u) \text{ and } s(v) \text{ disagree}).$$

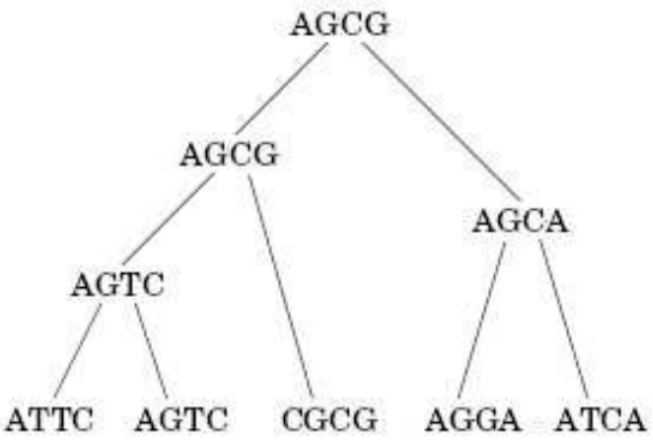
Finding the lowest-score tree is a difficult problem. Here we will consider just a small part of it: suppose we know the structure of the tree, and we want to fill in the sequences $s(u)$ of the internal nodes $u$. Heres an example with $k = 4$ and $n = 5$:



(a) In this particular example, there are several maximum parsimony reconstructions of the internal node sequences. Find one of them.

(b) Give an efficient (in terms of $n$ and $k$) algorithm for this task. (*Hint:* Even though the sequences might be long, you can do just one position at a time.)

**Solution:**




a)

b) The first observation is that letters in different positions don't interact. I.e. we can solve the problem separately for the first letter in each string, the second letter in each string, etc. and then add up the scores. So we henceforth consider the problem for sequences of length 1.

**Main ideas:**

1. We run a Dynamic programming algorithm.
2. The subproblem definition is, for each internal node $u$:
   $S(u)$ is the list of characters that could be optimal for $u$ given the subtree rooted at $u$ (i.e. ignoring $u$'s parent).
3. There is an easy formula for updating this list:
   Let $p$ be an internal node and let $c_1$ and $c_2$ be its children. If $S(c_1) \cap S(c_2)$ is non-empty, then $S(p) = S(c_1) \cap S(c_2)$. Otherwise, $S(p) = S(c_1) \cup S(c_2)$.

*Claim:* Let $p$ be an internal node and let $c_1$ and $c_2$ be its children. If $S(c_1) \cap S(c_2)$ is non-empty, then $S(p) = S(c_1) \cap S(c_2)$. Otherwise, $S(p) = S(c_1) \cup S(c_2)$.
*Proof:* Let $C(u)$ be the minimum cost of a labeling of the subtree rooted at $u$. Consider the two cases:

1) $I = S(c_1) \cap S(c_2)$ is non-empty. Then, any assignment with $s(p) \in I$ will have cost $C(c_1) + C(c_2)$, which is minimum for the subtree rooted at $p$. Moreover, any assignment with $s(p) \notin I$ will pay at least $C(c_1) + C(C_2) + 1$ and will not be optimal. Hence, $S(p) = I$.
2) If $I = S(c_1) \cap S(c_2)$ is empty, consider the union $U = S(c_1) \cup S(c_2)$. For any $s(p) \in U$, it is possible to construct an assignment of cost $C(c_1) + C(c_2) + 1$ by choosing a optimal assignment corresponding to $s(p)$ for one child's subtree and any assignment for the other. For $s(p) \notin U$, we pay $1 + C(c_i)$ for each child $i$ for whose subtree we use an optimal labeling and at least the same quantity $1 + C(c_i)$ for each child $i$ for which we do not use an optimal labeling. Hence, we pay at least $2 + C(c_1) + C(c_2)$. This shows that the optimal assignments for the subtree rooted at $p$ are those consisting of $s(p) \in U$ and optimal assignments for the children's subtrees.

We can then initialize $S(v) = \{s(v)\}$ for the leaves and proceed up the tree, updating $S(p)$ to be the intersection of $S(c_1)$ and $S(c_2)$, if this is non-empty, and their union otherwise. Once, we have reached the root and calculated $S(r)$ we can produce an actual labeling by setting $s(r)$ to any label in $S(r)$. The label for all other internal nodes $u$ can then be calculated from the label $s(p)$ of $u$'s parent by taking $s(u) = s(p)$ if $s(p) \in S(u)$ and $s(u)$ to be any label in $S(u)$ otherwise.

For each position of the string, we have $O(n)$ subproblems $S(u)$, each of which takes time $O(|\Sigma|)$ to update. The total running time is then $O(|\Sigma|nk)$.

**4. (20 pts.) Airline scheduling (vertex disjoint path cover)**

(a) Given a directed acyclic graph $G = (V, E)$, find the minimal number of vertex-disjoint paths that together cover all vertices (i.e. each vertex appears in exactly one path).
(*Hint #1:* Use max-flow.
*Hint #2:* $|V| =$ (number of paths) + (total number of edges in paths).)

(b) You run an airline. There are $n$ airports, with a flight from airport $i$ to airport $j$ taking $d_{ij}$ time. Based on consumer demand, there are $m$ pre-scheduled flights that you must make: each flight is from airport $a_i$ to airport $b_i$, and must depart at exactly time $t_i$.
What is the minimal number of planes you need in order to be able to handle all $m$ flights? (Planes can fly between airports outside of the $m$ required flights, these extra flights still take the same amount of time.)

**Solution:**

(a) We reduce this to a max flow problem.
Create a new graph $G'$ with $2|V| + 2$ vertices: for each vertex $v \in V$, create vertices $v^{(1)}$ and $v^{(2)}$, and also have two vertices $s$ and $t$. For each $v \in V$, have edges $(s \to v^{(1)})$ and $(v^{(2)} \to t)$. In addition, for each edge $e = (u \to v) \in E$, create an edge in $G'$ from $u^{(1)}$ to $v^{(2)}$.
We now run max flow from $s$ to $t$; if the max flow is $f$, then the minimal number of vertex-disjoint paths is $|V| - f$.

Proof:

- If we are given an integral flow of size $f$ in the new graph, we can construct $|V| - f$ vertex-disjoint paths in the original graph as follows: If there is positive flow through edge $(u^{(1)} \to v^{(2)})$, add edge $(u \to v)$ to the paths. Because of the capacity constraints on the flow from $s$ to $u^1$ and from $u^2$ to $t$, we know that each vertex has at most one incoming and one outgoing edge - so we have a collection of vertex disjoint paths. Finally, as the hint suggested, $f$ is exactly the number of edges in the collection of paths.
- Given a collection of paths $|V| - f$ in the original graph, we can construct a flow of size $f$ in $G'$ as follows: for each edge $(u \to v)$ in the path, we push one unit of flow from $s$ through $u^1$, $v^2$, to $t$.

(b) Create a graph with $m$ vertices, where each vertex represents one of the $m$ required flights. There is an edge between two vertices if a single plane is capable of taking care of both flights. Specifically, if flight $u$ ends at the same city that flight $v$ starts from, then there is an edge from $u$ to $v$ if $t_u + d_{a_u,b_u} \leq t_v$; if flight $u$ ends at a different city, then there is an edge if $t_u + d_{a_u,b_u} + d_{b_u,a_v} \leq t_v$. From here, just run the algorithm from part (a) on the graph, to get how many planes you need.

**5. (20 pts.) Verifying a max-flow**
Suppose someone presents you with a solution to a max-flow problem on some network. Give a *linear* time algorithm to determine whether the solution does indeed give a maximum flow.

**Solution:**

First, we construct the residual graph $G^f$, in linear time $O(|V|+|E|)$. Second, we can perform DFS on $G^f$ to determine the set $K$ of nodes that are reachable from $S$. This also takes linear $O(|V|+|E|)$ time. $K$ contains $T$ iff the flow is not maximum.

**6. (5 optional points pts.) Star-shaped polygons in 2D**

Consider a polygon in two dimensions. The polygon is specified by an ordered list $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$ where the $n$ line segments between $p_i$ and $p_{i+1}$ for $i \in \{1, 2, \ldots, n-1\}$ and $p_n$,$p_0$ do not intersect except at endpoints. Since a polygon does not cross itself, the boundary of the polygon splits the plane into two regions. Each segment is adjacent to both regions and, accordingly, has two sides. Suppose that for each segment we are given as part of the input which side corresponds to the interior of the polygon.

Call a polygon *star-shaped* if there a point inside of the polygon from which all points on the boundary of the polygon can be seen. That is, the polygon is star-shaped iff there is a point $x$ inside the polygon such that for every $y$ on the boundary of the polygon, the segment $xy$ only intersects the polygon at $y$.

Write a linear program that can be used to identify whether or not a polygon is star-shaped.

**Solution:**

Consider one segment of the polygon. Notice that the interior of the polygon is adjacent to exactly one side of the segment and we are given this side as part of the input. Parametrize side $i$ by the halfplane $a_i x + b_i y \geq c_i$ for some coefficients $a_i, b_i, c_i \in \mathbb{R}$. Call this halfplane $H_i$.

Notice that if some point $p = (x, y)$ is not in $H_i$, then the line segment between $p$ and $p_i$ or between $p$ and $p_{i+1}$ must exit the polygon. In particular, $p$ cannot affirm that the polygon is star-shaped. If $p$ is in every $H_i$, then the path between $p$ and any $p_i$ must be contained inside of the polygon.

Therefore, a polygon is star-shaped if and only if there exists a point $p$ in the intersection of all of the $H_i$s. This is the same as asking whether or not the linear program

$$\max_{x,y} 0$$
$$\text{subject to } a_i x + b_i y \geq c_i \forall i \in \{1, 2, \ldots, n\}$$

is feasible.