

Homework 2 Solutions

Note: there are usually several correct ways to decompose a DP problem. I'm picking some of my favorites here, but you may have a completely different decomposition that is nevertheless both correct and efficient.

1. (10 points) Think back to the aliens problem in Homework 1. In that problem, we tried to minimize the total number of projects that were permitted to use the telescope, and each project made it more likely that the alien home world would be discovered. Now we suppose that the likelihood that a project will discover the alien home world depends on how many hours the project gets on the telescope. So (to you) the penalty of assigning a job is proportional to the length of the job.

Additionally (!), Earth recently discovered parallel computing, so if two jobs overlap, *both* of them get to analyze the data from the telescope. Your goal is now to assign jobs to minimize the chance of discovering the home world.

As in the previous problem, you are to assume that there is some project that covers each part of the interval $[S, F)$, and that you must "avoid suspicion" by never allowing the telescope to be inactive. Each chosen project $p_i \in P$ will work the interval $[s_i, f_i)$ (precisely; no partial intervals are permitted, so projects may overlap). Give an efficient algorithm to select projects to cover the period $[S, F)$ that minimizes the total cost (i.e. total length of intervals) for all projects chosen, i.e.

$$\sum_{p_i \text{ chosen}} f_i - s_i.$$

Before we begin, we first sort the n projects in increasing order of their starting times. Let P_F be the subset of projects whose intervals reach or overlap F . For each p_i , let P_{s_i} be the subset of projects that reach or overlap time s_i .

Choice: any feasible schedule of projects must end with a project from P_F , since the entire time up to F must be covered. As a first choice, we consider selecting each project from P_F in turn.

Substructure 1: If we select project p_i from P_F , then the subproblem is to cover the interval $[S, s_i)$ using projects from the remaining set $P - \{p_i\}$. The interval is shorter, so this is a valid subproblem, and it has no constraints relative to the full problem.

Substructure 2: Let S_i be an optimal solution to the subproblem left after selecting first project $p_i \in P_F$. Observe that

$$\text{cost}(\{p_i\} \cup S_i) = \text{cost}(S_i) + (f_i - s_i),$$

and so the cost is divisible between the first choice and the subproblem cost. Apply the usual contradiction argument. QED

Recurrence: We address the general subproblem of finding the best set of projects for the interval $[S, s_i]$. Let $C(s_i)$ be this cost. Using our substructure properties, a general recurrence for $C(s_i)$ is

$$C(s_i) = \min_{b_j \in B_{s_i}} C(s_j) + (f_j - s_j).$$

As our base case, $C(s) = 0$ for any $s \leq S$. Our goal is to determine $C(F)$. If we evaluate $C(s_i)$ in increasing order of s_i (with F evaluated last), the dependencies of the recurrence are satisfied.

Since there are only n fragments, there are at most n distinct times s_i to consider, plus the time F . For each time, we need to look at all intervals overlapping that time, of which there may be up to n . Conclude that this algorithm is worst-case $\Theta(n^2)$.

2. *Professor Audubon has recorded the song of a bird in his backyard. The record consists of a time series Q of n successive pitch measurements $q_1 \dots q_n$, taken once every 100 milliseconds. The professor wants to divide the song into its individual notes (intervals of constant pitch). Unfortunately, experimental noise randomly perturbs the pitches measured over the course of a single note, so that they do not remain constant.*

To approximately divide the song into notes, the professor proposes the following approach. Let the variance of an interval $q_i \dots q_j$ of Q , denoted $\sigma^2(i, j)$, be defined in the usual statistical sense as

$$\sigma^2(i, j) = \sum_{k=i}^j (q_k - E(i, j))^2,$$

where

$$E(i, j) = \frac{1}{j - i + 1} \sum_{k=i}^j q_k$$

is the mean pitch of measurements $i..j$. If the measurement error in each pitch is small, then the variance of the interval corresponding to a single note should be low, while the variance over intervals combining distinct notes should be high. Hence, we want to divide Q into notes such that the sum of variances for all notes is minimized.

There is one important problem with the above approach. The number of notes is unknown a priori. One could simply create a new note for each measurement, making the variance of each note (and hence the sum of variances for all notes) zero! To avoid this trivial solution, we penalize a proposed division into m notes by adding $m \cdot p$ to its cost, where $p > 0$ is a user-defined penalty.

To summarize, the goal is to divide the song Q into notes by selecting $m \leq n$ break points $b_1 \dots b_m$ (the ends of each note) so as to minimize

$$m \cdot p + \sum_{i=1}^{m+1} \sigma^2(b_{i-1} + 1, b_i)$$

where $b_0 = 0$ and $b_{m+1} = n$.

Give an efficient algorithm for this problem. Your solution should run in time $O(n^2)$. Hint: show how to compute $\sigma^2(i, j)$ in constant time for any (i, j) .

We first show that variance can be efficiently computed for any interval (i, j) . Let $t = j - i + 1$. Observe that

$$\begin{aligned}
\sigma^2(i, j) &= \sum_{k=i}^j (q_k - E(i, j))^2 \\
&= \sum_{k=i}^j q_k^2 - 2E(i, j) \sum_{k=i}^j q_k + tE(i, j)^2 \\
&= \sum_{k=i}^j q_k^2 - 2E(i, j) \cdot tE(i, j) + tE(i, j)^2 \\
&= \sum_{k=i}^j q_k^2 - tE(i, j)^2
\end{aligned}$$

because $E(i, j) = \frac{1}{t} \sum_{k=i}^j q_k$. Hence, we can compute $\sigma^2(i, j)$ in constant time given the sums $\sum_{k=i}^j q_k$ and $\sum_{k=i}^j q_k^2$. Note further that in general,

$$\sum_{k=i}^j a_k = \sum_{k=1}^j a_k - \sum_{k=1}^{i-1} a_k.$$

Hence, if we precompute and store the values $\sum_{k=1}^j q_k$ and $\sum_{k=1}^j q_k^2$ for $1 \leq j \leq n$, we can derive all necessary sums, and hence the variance, in constant time from these values. This precomputation takes only linear time and space.

Now, let's solve the full problem.

Choice: we consider where to put the first breakpoint b_1 in the series. This breakpoint may be inserted after any q_k , $k \geq 1$. (If the breakpoint is inserted after q_n , then there is just one note.)

Substructure 1: after inserting breakpoint b_1 after q_k , we are left with the subproblem of optimally dividing $q_{k+1} \dots q_n$ into notes.

Substructure 2: Let $C(k+1)$ be the cost associated with an optimal division into notes of $q_{k+1} \dots q_n$, given that we inserted our first break point after q_k . Then we have

$$C(1) = C(k+1) + p + \sigma^2(1, q),$$

which separates the cost of the first interval from that of the subproblem. Apply the usual contradiction argument. QED

Recurrence: we address the general subproblem of dividing $q_j \dots q_n$ into notes. Let $C(j)$ be the optimal cost of such a division. By the above argument, we have

$$C(j) = \min_{j' \geq j} p + \sigma^2(j, j') + C(j'+1).$$

Our base case is $C(n+1) = 0$, and our goal is to optimize $C(1)$. If we compute $C(j)$ in decreasing order for $n \geq j \geq 1$, we will satisfy the dependencies.

We compute n values of $C(j)$, each of which takes $O(n-j)$ time, so our total complexity is $O(n^2)$.

3. Think back to the skis and skiers problem of Homework 1. Recall that the goal is to match each skier height h_i to a ski length l_i so as to minimize the sum of absolute differences $\sum_i |h_i - l_i|$.

Suppose that the numbers of skiers and skis are n and $m > n$ respectively. As before, we may not match more than one pair skis to the same skier; however, in this version of the problem, not all pairs of skis must be used. Given an efficient algorithm to assign a pair of skis to each skier. Be sure to analyze your time complexity as a function of both m and n .

(**Note:** you may want to look at the solution to the $m = n$ case in Homework 1 before attempting this problem! You need not re-prove anything about that algorithm on this homework.)

For this problem, we exploit the observation from Homework 1 that, given n skiers and n skis, each sorted in order from shortest to longest, an optimal solution pairs l_1 with h_1 , l_2 with h_2 , and so forth. For the general problem of n skiers and $m \geq n$ skis, our goal is therefore to decide which n out of the m skis will be used; given this information, we can easily derive an optimal solution.

We begin by sorting both from shortest to longest

Choice: We may choose to use the ski with l_1 , or not. If we do use it, it will be assigned to skier with height h_1 .

Substructure 1: If we select l_1 , we are left with the subproblem of matching $l_2 \dots l_m$ to $h_2 \dots h_n$. If not, we are left with matching $l_2 \dots l_m$ to $h_1 \dots h_n$.

Substructure 2: Let $C(i, j)$ be our cost for matching $l_i \dots l_m$ to $h_j \dots h_n$. If we do not use l_1 , then

$$C(1, 1) = C(1, 2).$$

If we do use l_1 , then

$$C(1, 1) = C(2, 2) + |l_1 - h_1|.$$

Apply the usual contradiction argument. QED

Recurrence: Our general subproblem is to find the best assignment of skis $l_j \dots l_m$ to heights $h_i \dots h_n$. Let the cost of this best assignment be $C(i, j)$. By our substructure properties, we have

$$C(i, j) = \min \begin{cases} C(i, j + 1) \\ C(i + 1, j + 1) + |l_j - h_i| \end{cases}$$

The base case occurs whenever $n - i = m - j$; in this case, the cost is simply that of the greedy solution. The target case is $C(1, 1)$. A valid ordering for the subproblems is to go in decreasing order of i and j (in either order).

This solution requires $O(n \log n + m \log m)$ time to sort the two lists, then solves $O(n(m - n))$ subproblems. Each non-base-case subproblem takes constant time to solve, and the n different base case costs can trivially be computed in total time $O(n)$. Hence, the total cost of the algorithm is $O(m \log m + nm)$.

4. (15 points) Consider the problem of word-wrapping a paragraph. A paragraph is an ordered list of n words, where word w_i is l_i letters long. You want to divide the paragraph into a sequence of lines, each containing at most L letters. (No word is more than L letters long.)

Suppose a line contains words $w_i \dots w_j$. The total length $W(i, j)$ of this line is defined by

$$W(i, j) = j - i + \sum_{k=i}^j \ell_k.$$

This length accounts for a single space between successive pairs of words on the line. The slop $S(i, j)$ of this line is defined to be $L - W(i, j)$, the total number of unused spaces at the end of the line. Note that in any feasible solution, the slop of each line must be non-negative. (The cubed slop criterion is a simplified version of what is actually used in, e.g., TeX for paragraph wrapping.)

Just to make things concrete, consider the example paragraph “Now is the time for all good men.”, and suppose $L = 10$. One feasible solution is

Now is the
time for
all good
men.

This solution has four lines of lengths 10, 8, 8, and 4; the corresponding slops are 0, 2, 2, and 6.

Your goal is to find a division of the input paragraph into lines that minimizes the sum, over all lines except the last, of the cubed slop of each line. (We omit the last line because it can in general be much shorter than the others.) For example, the total cost of the above solution is $0^2 + 2^3 + 2^3 = 16$.

Give an efficient algorithm for this problem.

We first solve the “core” problem, then deal with the special treatment of the last line. In the core problem, every line, *including the last*, counts toward the total slop.

Choice: Our choice will be before which word to begin the last line of text. For each word i , let P_i be the list of feasible split points for the paragraph, such that we may form a single line of text starting after the split and ending with word i . An optimal solution for words $1 \dots i$ must choose one of the splits in P_i . Note that $|P_i| = O(L)$, since each word takes up at least one space on the line.

Substructure 1: suppose we start with words $1 \dots n$ and split the list, leaving the last k words on the last line. We are left with the subproblem P' of formatting the words $1 \dots n - k$.

Substructure 2: Let Π' be an optimal formatting of words $1 \dots n - k$, and let Π be the solution that places words $n - k + 1 \dots n$ on the last line and formats the remaining text as for Π' . Then we have

$$\text{cost}(\Pi) = \text{cost}(\Pi') + S^2(n - k + 1, n).$$

Apply the contradiction argument. QED

Recurrence: Let $C(i)$ be the cost of an optimal formatting of words $1 \dots i$. Then we have

$$C(i) = \min_{j \in P_i} \left(C(j) + S^2(i - j + 1, i) \right).$$

The base case is $C(0) = 0$, and the target case (for the *core* problem) is $C(n)$. A valid ordering for subproblems is in increasing order of i .

To address the full problem, we may do one further “free” split. Let D be the cost of the best formatting of the whole paragraph, not counting the last line of text. Then

$$D = \min_{j \in P_n} C(j).$$

The algorithm solves n subproblems, each of which requires minimizing over all the split points in some P_i . Each P_i has size at most L ; hence, each subproblem takes time $O(L)$, and the total running time is $O(nL)$.

5. (541 only problem, 15 points) *The Euclidean traveling-salesman problem requires one to determine the shortest closed tour that connects a given set of n points in the plane. Unlike a “path”, a tour must start and finish at the same point. A farcical look at Euclidean Traveling Salesman problems is at:*

<http://www.oberlin.edu/math/faculty/bosch/making-tspart-page.html>

In this problem we consider a simpler problem: two-way tours, which start at the leftmost point, go strictly left to right to the rightmost point, and then go strictly right to left back to the starting point. An example of such a tour is shown on the left of:

http://www.cs.wustl.edu/~pless/546/lectures/f7_1.gif

Describe an $O(n^2)$ -time dynamic programming algorithm for determining an optimal two-way tour. You may assume that no two points have the same x coordinate.

Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.

Assume that the n points are sorted by their x -coordinates with v_1 being the leftmost point (if this is not the case it can be done in $O(n \log n)$ time). We can think of the bitonic tour as two left-to-right paths from v_1 to v_n that share no common vertices except for v_1 and v_n . We construct the paths simultaneously moving left to right (saving the last step of closing off tour outside of our subproblem formulation). Let i be the rightmost vertex so far in one of the paths and let j be the rightmost vertex so far in the other path. Since no two vertices are at the same x coordinate, one of i and j must be farther right than the other, thus $j < i$. Let $\overline{v_s v_t}$ be the length of an edge from v_s to v_t . Then we get the recursive definition for $m[i, j]$ which is the cost of the optimal solution for vertices v_1, \dots, v_i where there are two bitonic paths (with each vertex on exactly one) where one of the paths ends at v_j (and the other at v_i) for $1 \leq j < i \leq n$:

$$m[i, j] = \begin{cases} m[i-1, j] + \overline{v_{i-1} v_i} & \text{if } 1 \leq j \leq i-2 \\ \min(m[j, j-1] + \overline{v_{j-1} v_i}, m[i, j-1] + \overline{v_{j-1} v_j}) & \text{if } j = i-1 \end{cases}$$

We now argue that this recursive definition is correct. When computing the optimal solution for all points 1 to i , one of the paths must end with i and the other with some j for $1 \leq j \leq i-1$.

Note that once j is chosen any optimal pair of paths with i in one and j in the other must build on the path corresponding to $m[i-1, j]$. Namely, if an optimal path was constructed that did not build on $m[i-1, j]$ then there would be a better path for $m[i, j]$. When $1 \leq j \leq i-2$ then since both paths are left-to-right paths in the optimal solution corresponding to $m[i, j]$ it must be that i is connected to $i-1$. When $j = i-1$ then vertex $j-1$ must be connected to either vertex $j = i-1$ or vertex i and both cases are considered. Finally, it is easily verified that the optimal substructure property holds since for any i and j . Hence the recursive definition for $m[i, j]$ is correct.

We now show how the answer to the final problem which must be a closed tour versus two bitonic paths that include all vertices. The length L of the shortest bitonic tour is computed from the n th row of the matrix m by $L = \min_{1 \leq j \leq n-1} m[n, j] + \overline{v_j v_n}$.

This is clearly correct since the optimal solution must add some last edge from v_j to v_n to close the two paths computed in the subproblem, and we consider all possible values for j .

We now analyze the time complexity of our procedure. The pre-processing step to sort the vertices from left to right takes $O(n \log n)$ time. We compute the matrix m in row major order. There are $O(n^2)$ subproblems each which takes $O(1)$ time to compute. Finally, once the matrix has been computed $O(n)$ time is used to find the final solution (we pick the best solution out of the $n-1$ values for j where for each we must look up a value from m and add a distance to it). Hence the overall time complexity is $O(n^2)$.

An alternative recursive definition that could be used is:

$$m[i, j] = \begin{cases} m[i-1, j] + \overline{v_{i-1} v_i} & \text{if } 1 \leq j \leq i-2 \\ \min_{(1 \leq k \leq i-2)} (m[i-1, k] + \overline{v_k v_i}) & \text{if } j = i-1 \end{cases}$$

This is correct since when $j = i-1$, then i can be connected to any k in the solution corresponding to $m[i-1, k]$ for $1 \leq k \leq i-2$. The overall time complexity is still $O(n^2)$ since each row i (for $1 \leq i \leq n$) of m can be computed in $O(i)$ time.