

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

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

Substructure 1: If we select sitter b_i from B_F , then the subproblem is to cover the interval $[S, s_i)$ using sitters from the remaining set $B - \{b_i\}$. 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 sitter $b_i \in B_F$. Observe that

$$\text{cost}(\{b_i\} \cup S_i) = \text{cost}(S_i) + 20(f_i - s_i),$$

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 sitters 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

$$C(s_i) = \min_{b_j \in B_{s_i}} C(s_j) + 20(f_j - s_j).$$

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

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

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

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 break point 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. For this problem, we exploit the observation from Homework 1 that, given n measured and n known masses, each sorted in order from least to greatest, an optimal solution pairs x_1 with w_1 , x_2 with w_2 , and so forth. For the general problem of n measured and $m \geq n$ known masses, our goal is therefore to decide which n out of the m known masses will be used; given this information, we can easily derive an optimal solution.

We begin by sorting the known and measured masses from smallest to largest.

Choice: We may choose to use known mass w_1 , or not. If we do use it, it will be assigned to measured mass x_1 .

Substructure 1: If we select w_1 , we are left with the subproblem of matching $x_2 \dots x_n$ to $w_2 \dots w_m$. If not, we are left with matching $x_1 \dots x_n$ to $w_2 \dots w_m$.

Substructure 2: Let $C(i, j)$ be our cost for matching $x_i \dots x_n$ to $w_j \dots w_m$. If we do not use w_1 , then

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

If we do use w_1 , then

$$C(1, 1) = C(2, 2) + |w_1 - x_1|.$$

Apply the usual contradiction argument. QED

Recurrence: Our general subproblem is to find the best assignment of known masses $w_j \dots w_m$ to the measured masses $x_i \dots x_n$. Let the cost of this best assignment be $C(i, j)$. By our substructure properties, we have

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

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. 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. To avoid an explicit complexity dependence either on time or on the number of CDs used, we apply the following hack. Let C_j be the *minimum total number of CDs* needed by any collection of j out of the n songs. C_j is in general a *rational number*, not an integer; for example, if a collection of CDs uses 3 whole CDs and half the fourth, its total number of CDs is 3.5.

We will compute C_j for every $0 \leq j \leq n$, then find the largest of these $n + 1$ values that does not exceed k , our CD budget.

To compute C_j , we proceed from the last down to the first song. **Choice:** Our choice is whether or not to include the last song n in the collection. The best set of j songs either does or does not include the last song, so one of these choices is consistent with an optimal solution.

Substructure 1: Let $P(j, n)$ be the original problem. If we choose song n , then we are left with subproblem $P(j - 1, n - 1)$, since we have the remaining $n - 1$ songs but may choose only $j - 1$. Otherwise, we are left with subproblem $P(j, n - 1)$.

Substructure 2: Let S' be an optimal solution to the subproblem, and consider the solution S obtained by adjoining the first choice. If song n is not chosen, then the cost of S is that of S' , and the contradiction argument works.

If song n is chosen, then the cost of S is given by

$$\text{cost}(S) = \begin{cases} \text{cost}(S') + \frac{\ell_n}{m} & \text{if song } n \text{ fits on last CD} \\ \lceil \text{cost}(S') \rceil + \frac{\ell_n}{m} & \text{otherwise.} \end{cases}$$

In other words, if we cannot fit song n on the last CD, we close it and start a new one. Suppose S' is optimal, but S is not, and let S^* be any optimal solution that chooses song n . Let S'' be the subsolution obtained by removing song n from S^* . One may check that, regardless of which case of the above two each of S and S^* use, it must be that $\text{cost}(S'') < \text{cost}(S')$, which is impossible. Hence, S is optimal. QED

Recurrence: Let $C_j(i)$ be the minimum number of CDs used by any j songs from the set $1 \dots i$. Then by our substructure properties, we have

$$C_j(i) = \min \left\{ \begin{array}{l} C_j(i - 1) \\ \left\{ \begin{array}{l} C_{j-1}(i - 1) + \frac{\ell_i}{m} \\ \lceil C_{j-1}(i - 1) \rceil + \frac{\ell_i}{m} \end{array} \right. \begin{array}{l} \text{if song } n \text{ fits on last CD} \\ \text{otherwise.} \end{array} \end{array} \right.$$

The base cases for the recurrence are $C_j(j)$, which are simply the greedy solutions that pack all songs onto the CDs. The target cases are $C_j(n)$ for every j . A valid ordering of the subproblems is to loop over all j in increasing order, and within each j for i from j up to n .

To analyze the complexity, observe that we must compute $O(n^2)$ subproblems. All non-base-case subproblems take $O(1)$ time each, and we can obtain solutions to all base cases in total time $O(n)$. The final maximization over j is clearly $O(n)$, so the algorithm requires $O(n^2)$ time overall.

Note that solutions that are pseudopolynomial in k , the number of CDs, are OK here, since I said that k is fixed; the same goes for m , the CD length. Solutions that are pseudopolynomial in the total length of the song collection are *not* OK.