

CS 222 Fall 2007,

Homework 2, due October 17. It looks very long because of expositional material for Problem 3, but it should be shorter and more related to lecture material than was HW 1.

Put in the homework box in Kemper hall (Television students - find out from your contacts how to submit homeworks)

Problem 1. In the unweighted interval scheduling problem discussed in class, October 5, we are given a set P of intervals on the real line and we want to find a maximum cardinality (largest) subset of intervals $S \subset P$ such that no pair of intervals in S overlap. Let $maxD(P)$ denote the size of the largest such set S . For simplicity, assume that if two intervals touch, they overlap for more than a point. We gave the following greedy algorithm for that problem:

- 1) Sort the intervals in order of their right endpoints.
- 2) Start with no intervals in the “chosen” set C .
- 3) Scan the intervals left to right using the order established in step 1.

If a scanned interval i does not overlap any of the chosen intervals, then put i into C ; otherwise do not put i into C .

To prove the correctness of this method, we want to show that $|C| = maxD(P)$. We didn't do that in class, but will do so here by considering a related problem, The unweighted interval *stabbing* problem:

Given the set of intervals P as before, find the *minimum* number of vertical lines needed to intersect each of the intervals in P at least once. Let $minS(P)$ denote that number of vertical lines.

Problem 1a): Give a simple argument that $minS(P) \geq maxD(P)$.

Now consider the following way to stab the intervals: put one vertical line through the rightmost point of each of the intervals in the chosen set C from the above algorithm.

Problem 1b) Give a simple argument that those lines do correctly intersect each of the intervals in P at least once, and conclude that $minS(P) \leq |C|$.

Problem 1c) Explain how to combine the results from 1a) and 1b) with the fact that $|C| \leq maxD(P)$ (why is this true?), to conclude that the greedy algorithm is correct. Explain the conclusion that $minS(P) = maxD(P)$.

This is an example of a Min-Max Duality theorem. The stabbing and the interval problems are dual to each other. Such problems and theorems arise frequently in problems deriving from linear programming.

Now recall the *weighted* interval scheduling problem that we discussed in class: Given a set P of intervals and a number $N(I) > 0$ for each interval $I \in P$, define $WmaxD(P)$ as the maximum sum $\sum_{I \in S} N(I)$ such no pair of intervals in $S \subseteq P$ overlap.

Now define $WminS(P)$ as the *minimum* number of vertical lines needed so that each interval $I \in P$ intersects at least $N(I)$ of those $WminS(P)$ vertical lines.

Problem 1d) What can you establish about the relationship between $WmaxD(P)$ and $WminS(P)$? For example, is it always true that they are equal? I don't know the answer to this problem, so do what you can do, but don't exhaust yourself if you are not getting anywhere on it.

What about when $N(I) = 1$ for each I ?

Problem 2. Solve Problem 19 on page 329 of the text. To get started, let $L(i, j, k)$ be a variable that will be set to 1 if the first k characters of the input string s can be expressed as an interleaving of x and y that ends on the i 'th character of x and the j 'th character of y . (Note that the interleaving for the first k characters of s may use some full copies of x and y also, and only the last copy of x or the last copy of y may be incomplete). Otherwise set $L(i, j, k)$ to value 0. Set up DP recurrences for $L(i, j, k)$ and show how to evaluate them. Explain correctness, the traceback procedure, and the time analysis. Can you see a faster algorithm to solve the problem?

Problem 2', modified the way we discussed it in the discussion section. You can solve either Problem 2 or 2' as you wish, but be clear on which one it is. Don't worry if you have already handed in your homework, we will try to figure out which interpretation you used.

Solve Problem 19 on page 329 of the text. To get started, let x' be the string consisting of repetitions of x so that $|x'| \geq |s|$. Similarly, let y' be the string consisting of repetitions of y so that $|y'| \geq |s|$. Then let $L(i, j, k)$ be a variable that will be set to 1 if and only if the first k characters of the input string s can be expressed as an interleaving of the first i characters of x' and first j characters of y' . Note that $k = i + j$. Otherwise set $L(i, j, k)$ to value 0. Set up DP recurrences for $L(i, j, k)$ and show how to evaluate them. Explain correctness, the traceback procedure, and the time analysis. Can you see a faster algorithm to solve the problem?

Problem 3. Maximizing alignment

The text defined similarity of strings aligning to minimize costs. An

alternative way is to align to maximize values. This approach is chosen in most biological applications for technical reasons. We now begin to develop precise definitions.

Definition An *alignment* \mathcal{A} of two strings S_1 and S_2 is obtained by first inserting chosen spaces (or dashes), either into or at the ends of S_1 and S_2 and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string. However, we never allow a space to be placed opposite to a space, so that the number of alignments of two strings is finite.

Definition: Let Σ be the alphabet used for strings S_1 and S_2 , and let Σ' be Σ with the added character “-” denoting a space. Then, for any two characters x, y in Σ' , $s(x, y)$ denotes the value (or *score*) obtained by aligning character x against character y .

Definition: For a given alignment \mathcal{A} of S_1 and S_2 , let S'_1 and S'_2 denote the strings after the chosen insertion of spaces, and let l denote the (equal) length of the two strings S'_1 and S'_2 in \mathcal{A} . The *value* of alignment \mathcal{A} is defined as $\sum_{i=1}^l s(S'_1(i), S'_2(i))$.

That is, every position i in \mathcal{A} specifies a pair of opposing characters in the alphabet Σ' , and the value of \mathcal{A} is obtained by summing the value contributed by each pair.

For example let $\Sigma = \{a, b, c, d\}$ and let the pairwise scores be defined in the following matrix:

s	a	b	c	d	$-$
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
$-$					0

Then the alignment

c	a	c	$-$	d	b	d
c	a	b	b	d	b	$-$

has a total value of $0+1-2+0+3+3-1 = 4$.

In string similarity problems, scoring matrices usually set $s(x, y)$ to be greater or equal to zero if characters x, y of Σ' match and less than zero if they mismatch. With such a scoring scheme, one seeks an alignment with as *large* a value as possible. That alignment will emphasize matches between the two strings while penalizing mismatches or inserted spaces. Of course, the meaningfulness of the resulting alignment may depend heavily on the scoring scheme used and how match scores compare to mismatch and space scores.

Definition: Given a pairwise scoring matrix over the alphabet Σ' , the *Max similarity* of two strings S_1, S_2 is defined as the value of the alignment \mathcal{A} of S_1 and S_2 which *maximizes* total alignment value. This is also called the *optimal alignment value* of S_1 and S_2 .

Computing Max similarity

The Max similarity of two strings S_1 and S_2 , and the associated optimal alignment can be computed by dynamic programming, using recurrences that are related to, but different from, the recurrences in the text.

Definition $V(i, j)$ is defined as the Maximum possible *value* of an alignment of prefixes $S_1[1..i]$ and $S_2[1..j]$, i.e., using the first i characters of S_1 and the first j characters of S_2 .

The task is to efficiently compute $V(n, m)$ using dynamic programming.

Problem: Write out the recurrences and the base case and explain how to evaluate the recurrences, and how to find the optimal alignment itself, not just its value. Explain the running time of the solution.