Tutorial 5: Greedy algorithms

1 Total completion time

Definition 1. An instance of the *minimal total completion time* problem is a sequence of n jobs that have processing times $p_1, p_2, \ldots, p_n$ which are positive integers. A valid solution to such an instance is an ordering of the jobs $1, \ldots, n$ such that when the jobs are processed one at a time in that order, the sum of their completion times is minimized. For simplicity, assume all jobs have *unique* processing times.

Design a greedy algorithm for solving the minimal total completion time problem and prove that it is correct.

1.1 Solution

Sort jobs ascending by processing time (shortest job first). Let $J_1, J_2, ..., J_n$ be the list of jobs ordered this way (so $p_1, p_2, ..., p_n$ are ordered from shortest to longest processing times).

Note that the completion time C_{J_k} for job J_k is equal to the sum of *processing times* for jobs that are performed before it, plus its own processing time. That is,

$$C_{J_k} = \sum_{i=1}^{i \leq k} p_i.$$

Thus, the sum S of *completion times* for all jobs $J_1 ... J_n$ is

$$S = \sum_{k=1}^{k \leq n} C_{J_k} = \sum_{k=1}^{k \leq n} \sum_{i=1}^{i \leq k} p_i = np_1 + (n-1)p_2 + (n-2)p_3 + ... + 3p_{n-2} + 2p_{n-1} + p_n.$$

Note that the ith job in this order appears $(n - i + 1)$ times in the sum!

Consider a hypothetical *optimal* solution O that picks the job order $J_{\ell_1}, J_{\ell_2}, ..., J_{\ell_n}$. Let G be the greedy solution $J_1, J_2, ..., J_n$. We want to prove that $O = G$ (i.e., $\ell_i = i$ for all i).

To obtain a contradiction, assume $O \neq G$. Consider any two jobs J_a and J_b in G. Since the greedy algorithm picks shorter jobs first, if $a < b$, then we know $p_a < p_b$. Therefore, because $O \neq G$, there are two jobs J_a and J_b in O such that $a < b$ but $p_a > p_b$.

Observe that J_a appears $(n - a + 1)$ times in the sum of completion times for O. Similarly, J_b appears $(n - b + 1)$ times in the sum. We use A to denote $(n - a + 1)$ and B to denote $(n - b + 1)$.

We construct a slightly different solution $O' = J_{\ell'_1}, ..., J_{\ell'_n}$ by taking O and *swapping the order* of jobs J_a and J_b. (In other words, we set $J'_a = J_b$ and $J'_b = J_a$.) In this new solution, J_a is now at position b', which is simply b, so it appears $(n - b + 1)$ times in the sum of completion times. Similarly, J_b is now at position a', which is simply a, so it appears $(n - a + 1)$ times in the sum.

Every other job appears the exact same number of times in the sum $S_{O'}$ of completion times for O' as it does in the sum S_O of completion times for O.

So, we have

$$S_{O'} = S_O + p_a(n-b+1) + p_b(n-a+1) - p_a(n-a+1) - p_b(n-b+1).$$

This expands to

$$S_{O'} = S_O + p_a n - p_a b + p_a + p_b n - p_b a + p_b - p_a n + p_a a - p_a - p_b n + p_b b - p_b.$$

This in turn simplifies to

$$S_{O'} = S_O - p_a b - p_b a + p_a a + p_b b.$$

Rearranging, we get

$$S_{O'} = S_O + p_a(a-b) + p_b(b-a).$$

Rearranging to get two $(b-a)$ terms, we get

$$S_{O'} = S_O + p_b(b-a) - p_a(b-a) = (b-a)(p_b - p_a).$$

Since $p_b > p_a$, the second term is positive. Since $a < b$, the first term is negative. Thus, $(b-a)(p_b-p_a) < 0$. Consequently, $S_{O'} < S_O$, which *contradicts the optimality of O!* Since the only thing we assumed to obtain this contradiction is that $O \neq G$, that assumption must be incorrect. Therefore, $G = O$, so the greedy algorithm is optimal.

2 Efficient Line Breaks

Whenever a word processor or web browser displays a long passage of text, it must be broken up into lines of text. Determining where to make these breaks is known as the **line breaking** problem.

Definition 2. An instance of the *efficient line break* problem is as follows. Suppose you are given a sequence W of n words $w_1, ..., w_n$, where each word w_i has a given *length* l_i. For simplicity, we ignore any white space or punctuation that might appear around these words. Suppose further that you are given a *line length* L that represents the maximum number of text characters that fit on a line (assuming a fixed-length font). So, a sequence of words can fit on a line only if the sum of their lengths is at most L. The problem is to specify how to break words into lines by outputting a sequence of indices, $i_1, ..., i_k$, where $i_1 = 1$ and $i_k = n$, to indicate that we should break W into the lines $w_{i_j}, ..., w_{i_{k+1}-1}$ for all $j \in \{1, 2, ..., k-1\}$ subject to the **following constraints**:

$$i_j < i_{j+1} \text{ and } \sum_{r=i_j}^{i_{j+1}-1} l_r \leq L.$$

A correct (optimal) solution must **minimize** the following **penalty function**:

$$\sum_{j=1}^{k} \left| L - \sum_{r=i_j}^{i_{j+1}-1} l_r \right|.$$

(In other words, the penalty is the total amount of unused space over all lines, so you must *minimize* the total unused space.)

To be clear: the words **cannot be reordered**, since the text must retain its meaning.

The goal is to design a greedy algorithm to solve the problem, and prove it is correct.

2.1 Solution

The crucial realization is that the penalty for a *fixed number* k of line breaks is the same regardless of how words are distributed between lines. This can be seen easily by inspecting the penalty function, and noting that the total penalty is simply L times the number of line breaks used, less the total lengths of all words. Therefore, we can think only about minimizing the number of line breaks that are used.

Consider an algorithm that simply iterates through the words $w_1, ..., w_n$ and greedily packs as many words as it can into each line, before starting a new line. We prove that this algorithm will use the minimum possible number of line breaks.

Fix any arbitrary input $w_1, ..., w_n$. Let $G = i_1, ..., i_k$ be the solution that results from running the greedy algorithm on this input. Suppose, to obtain a contradiction, that an optimal algorithm creates a solution $O = j_1, ..., j_t$. We prove that $k = t$ (i.e., O uses the same number of line breaks as G).

Consider the following iterative procedure. Suppose it is **not** possible to move any word into the previous line (without violating the line length constraint L). Then O is the same as G, by construction. (This is because G will simply fill up lines greedily, so that no word can be moved onto the previous line.)

However, if it **is** possible to move some word into the previous line, then **move** that word to the previous line, and go to the next iteration. Moving a word to the previous line does **not** increase the number of line breaks used (and may even reduce the number that are needed). Furthermore, eventually this iterative procedure will end, when no word can be moved to the previous line, at which point we will have G.

If this iterative procedure results in one or more empty lines, then O is not optimal, which is a contradiction. On the other hand, if it does not result in any empty lines, then O and G use the same number of lines. In each case, we have proved that G is optimal.