A strategy for designing greedy algorithms and proving optimality[*]

Hung Q. Ngo

This document outlines a strategy for designing greedy algorithms and proving their optimality. I illustrated the strategy with two examples in the lectures on Monday and Wednesday. We will begin with the INTERVAL SCHEDULING problem, then the generic description of the strategy, and finally the HUFFMAN CODING example.

1 The INTERVAL SCHEDULING problem

In the INTERVAL SCHEDULING problem, we are given a set $\mathcal{R}$ of n intervals $[s_i, f_i)$, $1 \leq i \leq n$. We are supposed to output a maximum-size subset of $\mathcal{R}$ that are mutually non-overlapping.

Our algorithm $\mathbf{A}$ takes $\mathcal{R}$ as input and outputs $\mathbf{A}(\mathcal{R})$ – a subset of non-overlapping intervals. The algorithm is a recursive one:

(1) let $I \in \mathcal{R}$ be the interval with earliest finish time (i.e. smallest f_i),

(2) let $\mathcal{R}'$ be the set of intervals which do not overlap with I

(3) return $\{I\} \cup \mathbf{A}(\mathcal{R}')$

The hard part is to prove that the algorithm returns a best solution possible. Let $\text{OPT}(\mathcal{R})$ denote the maximum number of non-overlapping intervals in $\mathcal{R}$. We want to show that $\mathbf{A}(\mathcal{R}) = \text{OPT}(\mathcal{R})$, for all $\mathcal{R}$. We will prove this by induction on $|\mathcal{R}|$. The base case when $|\mathcal{R}| = 1$ obviously holds. Suppose $\mathbf{A}(\mathcal{R}) = \text{OPT}(\mathcal{R})$, for all input interval set $\mathcal{R}$ with at most $n-1$ intervals in it.

Consider an input $\mathcal{R}$ with n intervals. From the recursive structure of the algorithm, it is easy to see that

$$\mathbf{A}(\mathcal{R}) = 1 + \mathbf{A}(\mathcal{R}').$$

By the induction hypothesis, we have

$$\mathbf{A}(\mathcal{R}') = \text{OPT}(\mathcal{R}')$$

Thus,

$$\mathbf{A}(\mathcal{R}) = 1 + \text{OPT}(\mathcal{R}').$$

Consequently, we would be in business if we can prove that

$$\text{OPT}(\mathcal{R}) = 1 + \text{OPT}(\mathcal{R}'). \tag{1}$$

We will prove (1) by proving two claims.

[*]Notes based on Hung Ngo's guest lectures in Fall 09 version of CSE 331.

Claim 1.1. There exists an optimal solution O for the instance $\mathcal{R}$ (i.e. a set of non-overlapping intervals for which $|O| = \text{OPT}(\mathcal{R})$) such that $I \in O$.

Claim 1.2. Let O be the optimal solution in the previous claim. Let $O' = O - \{I\}$. Then, O' is optimal for the instance $\mathcal{R}'$; namely $|O'| = \text{OPT}(\mathcal{R}')$.

Before proving the two claims, let's assume they are true and prove (1) first. By Claim 1, we have $|O| = \text{OPT}(\mathcal{R})$. By Claim 2, we have $|O'| = \text{OPT}(\mathcal{R}')$. But $O = \{I\} \cup O'$. Thus, $|O| = 1 + |O'|$, which implies (1) as desired. Now we prove the two claims.

Proof of Claim 1.1. Let $\bar{O}$ be any optimal solution to the instance $\mathcal{R}$. If $\bar{O}$ contains I then we are done. If not, let $\bar{I}$ be the interval in $\bar{O}$ with the earliest finish time. Since I was the interval with the earliest finish time overall, I's finish time is at least as early as $\bar{I}$. Hence, I does not overlap with any interval in $\bar{O} - \{\bar{I}\}$. Consequently, the set $O = \bar{O} \cup \{I\} - \{\bar{I}\}$ is a set of non-overlapping intervals. Since $|O| = |\bar{O}|$ and $\bar{O}$ is optimal, O is optimal as well. And, O contains I as desired. $\square$

Proof of Claim 1.2. Suppose O' is not optimal for the instance $\mathcal{R}'$. Let $\bar{O}'$ be any optimal solution to $\mathcal{R}'$. In particular, $|\bar{O}'| > |O'|$. Recall that $\mathcal{R}'$ consists of all intervals in R that do not overlap with I. This means that non of the intervals in $\bar{O}'$ overlaps with I either. Thus, $\bar{O}' \cup \{I\}$ is a set of non-overlapping intervals. However,

$$|\bar{O}' \cup \{I\}| = 1 + |\bar{O}'| > 1 + |O'| = |O| = \text{OPT}(\mathcal{R}).$$

This is a contradiction as there can be no set of non-overlapping intervals with size more than $\text{OPT}(\mathcal{R})$. $\square$

2 A general greedy algorithm design strategy and its analysis

The above analysis of the recursive algorithm might seem convoluted at first read. However, it follows a very clear algorithm design and proof strategy, which is the subject of this section.

2.1 The design strategy

Let us consider an instance $\mathcal{R}$ of some optimization problem, in which we want to find a solution to minimize or maximize something. In many cases, a solution can be built piece-by-piece, using a recursive structure similar to the algorithm described in the previous section.

```
Generic Greedy Algorithm A on instance $\mathcal{R}$
(1) Construct a "piece" $I$ in a "greedy" manner.
(2) Define a sub-problem $\mathcal{R}'$
(3) "Glue" $I$ to $\mathbf{A}(\mathcal{R}')$ and output the result
```

You should try to compare the above generic description with the algorithm from Section 1. Here's another example with precisely the same structure. Given an input array $\mathbf{a}[1..n]$, sort the array in ascending order. The famous *bubble sort* algorithm has the same structure:

(1) let e be the smallest element of $\mathbf{a}$

(2) let $\mathbf{a}'$ be the array $\mathbf{a}$ with element e removed

(3) return $e\circ$ bubble-sort$(\mathbf{a}')$, where $\circ$ denotes the "concatenation" operation

An important point to notice here is that different problems typically require different ways to "glue" I to $\mathbf{A}(\mathcal{R}')$. Sorting in ascending order can be formulated as the problem of minimizing the number of *inversions* in the array, which we will encounter later.

2.2 The analysis

The cost of the final solution returned by $\mathbf{A}$, denoted by $\mathbf{A}(\mathcal{R})$, can often be computed from the "cost" of the piece I and the cost of the sub-solution $\mathbf{A}(\mathcal{R}')$:

$$\mathbf{A}(\mathcal{R}) = \text{cost}(I) + \mathbf{A}(\mathcal{R}'). \tag{2}$$

In some problem, the cost is not "additive" as above. It could be the case that $\mathbf{A}(\mathcal{R}) = \text{cost}(I)\cdot\mathbf{A}(\mathcal{R}')$, or some other way of computing $\mathbf{A}(\mathcal{R})$ from cost(I) and $\mathbf{A}(\mathcal{R}')$. However, we will use the "additive" case as an illustration because it occurs most often in practice.

Our objective is to prove that $\mathbf{A}(\mathcal{R}) = \text{OPT}(\mathcal{R})$. And, we often can do this by induction on $|\mathcal{R}|$: the size of the problem instance. By the induction hypothesis, we have $\mathbf{A}(\mathcal{R}') = \text{OPT}(\mathcal{R}')$. Thus, all we actually have to do is to prove that

$$\text{OPT}(\mathcal{R}) = \text{cost}(I) + \text{OPT}(\mathcal{R}'). \tag{3}$$

Note the parallel between (1) and (3). Now, we will prove (3) by proving two claims.

Claim 2.1 (Generic Claim 1)**.** There exists an optimal solution O to the instance $\mathcal{R}$ which "contains" the greedy choice I.

Claim 2.2 (Generic Claim 2)**.** Given the O in Generic Claim 1, $O' = O - I$ ("cut off" I from O) is an optimal solution to the instance $\mathcal{R}'$

It is extremely important to note that the statements above are meaningless if they are not explicitly stated in the context of a particular problem. See the section on INTERVAL SCHEDULING and HUFFMAN CODING for concrete example. Our objective here is to outline the general line of reasoning only.

Now, after proving the two claims, we can prove (3) by noting that

$$\text{OPT}(\mathcal{R}) = \text{cost}(O) = \text{cost}(I) + \text{cost}(O') = \text{cost}(I) + \text{OPT}(\mathcal{R}').$$

The first equality holds because O is optimal for $\mathcal{R}$. The second equality holds because O' is O with I "cut off." The third equality holds due to Generic claim 2.

3 Huffman coding

Given a text file over some alphabet $\mathcal{C}$ in which letter $c \in \mathcal{C}$ occurs $f(c)$ times, what is the best way to encode this text file such that the total number of bits needed is minimized?

For example, suppose $\mathcal{C}$ has only 4 letters a, b, c, d. We can use 2 bits to encode each letter: $00 = a, 01 = b, 10 = c, 11 = d$. The text string "*abbcccccccd*" will thus need $11 \cdot 2 = 22$ bits. In 1951, David Huffman, then a graduate student at MIT worked on a term paper whose topic was precisely this problem: how to design the most efficient coding scheme for a given a text file/string. He came up with the Huffman code. The idea is very simple: characters which appear more frequently in the text file should be encoded with less bits.

Huffman's idea is as follows. An encoding scheme can be represented by a full binary tree, where each leaf represents a character. For every internal node in the tree, mark the left branch with a 0, and the right branch with a 1. The sequence of 0s and 1s encountered when we travel from the root to a leaf is the encoding of the corresponding character. For instance, an encoding tree for the text string "*abbccccccccd*" might be

```
          *
       0/ \1
       *    c
    0/ \1
    *    b
 0/ \1
 a    d
```

The encoding is thus $000 = a, 001 = d, 01 = b, 1 = c$. It is not hard to see that we can uniquely decode any string encoded using this method, which is called a "prefix code." The total number of bits needed to encode "*abbccccccccd*" is now only 17 instead of 22.

Given an encoding tree T, let $d_T(c)$ denote the depth of character $c \in \mathcal{C}$ in the tree T, then we need $d_T(c)$ bits to encode c. Thus, a text file with character frequencies $f(c)$ will need

$$\text{cost}(T) = \sum_{c \in \mathcal{C}} f(c) d_T(c)$$

many bits to be encoded. The problem is to seek a full encoding tree T minimizing its cost.

Huffman proposed the following algorithm, following precisely the strategy described in the previous section. Initially, each character in $\mathcal{C}$ is an isolated node. No tree is formed yet, and they will be "merged" gradually to form a full binary tree.

(1) Let c_1 and c_2 be two least frequently appeared characters in $\mathcal{C}$. Create a new "fake" character c_{12}.

(2) Define the frequency of c_{12} by $f(c_{12}) = f(c_1) + f(c_2)$. Let $\mathcal{C}' = \mathcal{C} \cup \{c_{12}\} - \{c_1, c_2\}$.

(3) Let T' be the optimal encoding tree for the new character set $\mathcal{C}'$ (with the frequency for c_{12} defined above). T' is constructed recursively. To obtain T, join two nodes c_1, c_2 to the node c_{12} in T'. (This is how we "glue" the greedy choice to the sub-solution T'.)

Now, it's easy to see that

$$\begin{aligned}
\text{cost}(T) &= \sum_{c\in\mathcal{C}} f(c)d_T(c) \\
&= \sum_{c\in\mathcal{C}-\{c_1,c_2\}} f(c)d_T(c) + f(c_1)d_T(c_1) + f(c_2)d_T(c_2) \\
&= \sum_{c\in\mathcal{C}-\{c_1,c_2\}} f(c)d_T(c) + f(c_1)(d_{T'}(c_{12})+1) + f(c_2)(d_{T'}(c_{12})+1) \\
&= \sum_{c\in\mathcal{C}-\{c_1,c_2\}} f(c)d_T(c) + f(c_1)d_{T'}(c_{12}) + f(c_2)d_{T'}(c_{12}) + f(c_1) + f(c_2) \\
&= \sum_{c\in\mathcal{C}-\{c_1,c_2\}} f(c)d_T(c) + (f(c_1)+f(c_2))d_{T'}(c_{12}) + f(c_1) + f(c_2) \\
&= \sum_{c\in\mathcal{C}-\{c_1,c_2\}} f(c)d_{T'}(c) + f(c_{12})d_{T'}(c_{12}) + f(c_1) + f(c_2) \\
&= \sum_{c\in\mathcal{C}'} f(c)d_{T'}(c) + f(c_1) + f(c_2) \\
&= \text{cost}(T') + f(c_1) + f(c_2)
\end{aligned}$$

This is where I stopped on Wednesday!

The equality

$$\text{cost}(T) = f(c_1) + f(c_2) + \text{cost}(T') \tag{4}$$

plays the role of (2). Now, we prove two claims along the general line of reasoning.

Claim 3.1. There exists an optimal encoding tree $\bar{T}$ for the instance $\mathcal{C}$ in which c_1 and c_2 are siblings.

Claim 3.2. Let $\bar{T}$ be the optimal encoding tree from the previous claim. Let c_{12} be the parent of c_1 and c_2. Let $\bar{T}'$ be the tree $\bar{T}$ with c_1 and c_2 removed. Then $\bar{T}'$ is an optimal encoding tree for the sub problem $\mathcal{C}' = \mathcal{C} \cup \{c_{12}\} - \{c_1, c_2\}$, where $f(c_{12}) = f(c_1) + f(c_2)$.

Let's assume the two claims are true. We want to prove that

$$\text{cost}(T) = \text{cost}(\bar{T}) \text{ which is the optimal cost for } \mathcal{C}.$$

We shall prove by induction on $|\mathcal{C}|$ that our algorithm always constructs an optimal encoding tree, which certainly holds when $|\mathcal{C}| = 2$. Let's assume that our algorithm is optimal for $|\mathcal{C}| \leq n-1$. Consider the case when $|\mathcal{C}| = n$. By an identical reasoning as above, we can show that

$$\text{cost}(\bar{T}) = f(c_1) + f(c_2) + \text{cost}(\bar{T}').$$

By the induction hypothesis, we have $\text{cost}(T') = \text{cost}(\bar{T}')$, because $|\mathcal{C}'| \leq n-1$. This along with (4) give

$$\text{cost}(\bar{T}) = f(c_1) + f(c_2) + \text{cost}(T') = \text{cost}(T)$$

as desired. Now, we prove the two claims.

Proof of Claim 3.1. Let $\bar{T}$ be any optimal solution for $\mathcal{C}$. If c_1 and c_2 are siblings, then we are done. Otherwise, without loss of generality, we assume that $d_{\bar{T}}(c_1) \geq d_{\bar{T}}(c_2)$ and that c is a sibling of c_1. Let T^* be the tree obtained from $\bar{T}$ by switching the nodes c_2 and c. Then, since everything else is the same except for c_2 and c, we have

$$\begin{aligned}
\text{cost}(T^*) - \text{cost}(\bar{T}) &= (f(c_2)d_{T^*}(c_2) + f(c)d_{T^*}(c)) - (f(c_2)d_{\bar{T}}(c_2) + f(c)d_{\bar{T}}(c)) \\
&= (f(c_2)d_{\bar{T}}(c) + f(c)d_{\bar{T}}(c_2)) - (f(c_2)d_{\bar{T}}(c_2) + f(c)d_{\bar{T}}(c)) \\
&= (f(c_2) - f(c))d_{\bar{T}}(c) + (f(c) - f(c_2))d_{\bar{T}}(c_2) \\
&= (f(c_2) - f(c))(d_{\bar{T}}(c) - d_{\bar{T}}(c_2)) \\
&= (f(c_2) - f(c))(d_{\bar{T}}(c_1) - d_{\bar{T}}(c_2)) \\
&\leq 0.
\end{aligned}$$

The last inequality follows because $f(c_2) \leq f(c)$ (c_2 was one of the two *least* frequent characters), and $d_{\bar{T}}(c_1) \geq d_{\bar{T}}(c_2)$ by our assumption above. Since $\bar{T}$ is already optimal, cost$(T^*) \leq$ cost$(\bar{T})$ implies that T^* must be optimal as well; moreover, c_1 and c_2 are siblings in T^* as we wanted to show. □

Proof of Claim 3.2. Suppose $\bar{T}'$ is not optimal for the problem $\mathcal{C}'$. Then, there's an even better encoding tree $\hat{T}'$ for the sub-problem $\mathcal{C}'$: cost$(\hat{T}') <$ cost$(\bar{T}')$. Now, let $\hat{T}$ be the tree obtained by "gluing" c_1 and c_2 to the parent node c_{12} in $\hat{T}'$. Then,

$$\text{cost}(\hat{T}) = f(c_1) + f(c_2) + \text{cost}(\hat{T}') < f(c_1) + f(c_2) + \text{cost}(\bar{T}') = \text{cost}(\bar{T}).$$

This means that $\bar{T}$ is not the best solution to the problem $\mathcal{C}$, a contradiction. □