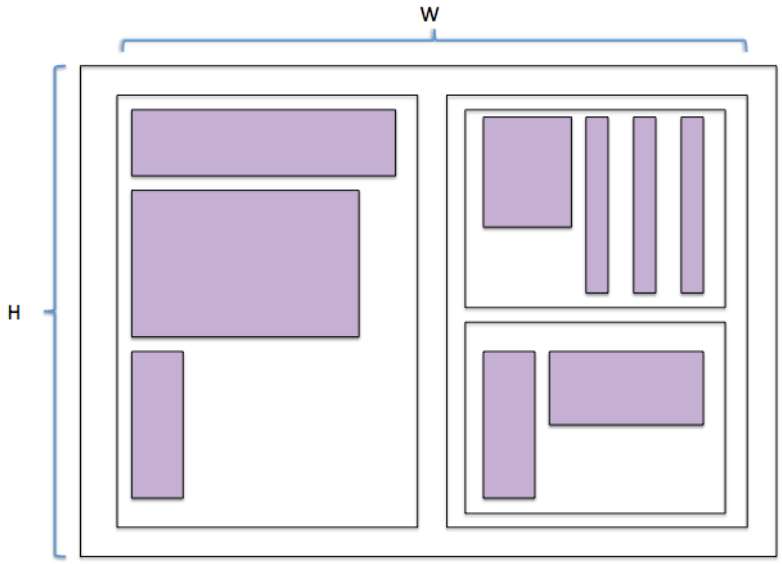
Dynamic Programming: Widget Layout

Setup

There are two types of widgets.

- A **leaf widget** is a visible widget that someone may see or use, such as a button or an image. Every leaf widget has a list of possible rectangular sizes it can be presented as. For example, a button leaf widget may be a rectangle size 3×1, 4×2, or 6×4.
- An **internal widget** is a rectangular container that contains widgets inside of it. It may contain a combination of leaf widgets and other internal widgets (may be referred to as children widgets). Widgets may be arranged horizontally or vertically within the internal widget. The size of an internal widget must be large enough to contain all the widgets inside of it (e.g. with a horizontal orientation, width of internal widget is the sum of the widths of the widgets inside and height of internal widget is the maximum height of the widgets inside).



In the widget layout problem, our goal is to fit widgets into a rectangular screen of size $W \times H$ given a hierarchy of widgets. A hierarchy of widgets defines which widgets are contained within which widgets and in what order. It does not state the orientation of the internal widgets or which size of the legal sizes of a leaf widget we choose. We are free to vary the orientation or the sizes of the leaf widgets to try to fit all the widgets in some rectangular screen size $W \times H$. Note that the hierarchy of widgets can be represented as a tree where the non-leaf nodes are internal widgets and the leaf nodes are leaf widgets. The root of the tree represents the internal widget that contains everything. We are deciding if this root widget can be contained in a rectangle size $W \times H$.

Defining Subproblems

This is a more complex dynamic programming problem with multiple types of widgets and several factors needed to be taken account. In the end though, there are two types of guesses that we're making. For each leaf widget, we need to choose a size from its list of acceptable sizes. For each internal widget, we need to choose the orientation of its children widgets. Once an internal widget's orientation and children sizes have been set, we can calculate the size of the internal widget by packing all the children together as tightly as possible.

Our subproblem will be $L(v, w)$, representing the minimum height of a widget v such that v fits into a rectangle size $w \times h$. We will want to calculate $L(v, w)$ for every widget v and every w where $0 < w < W$. Eventually, we will calculate $L(root, W)$ to determine the minimum height h of the root widget such that the root fits into a rectangle size $W \times h$. If $h \leq H$, then we can conclude that the widgets will all fit in a rectangle size $W \times H$.

Combining Subproblems

Leaf Widgets

It is straightforward to figure out what $L(v, w)$ should be if v is a leaf widget. Simply examine all of the widget's valid sizes and take the minimum height of all sizes with width less than w. In other words,

$$L(\text{leaf } v, w) = \text{ min}(h' \text{ for } (w', h') \text{ in } v.\text{sizes if } w' \leq w) \tag{1}$$

It is a little bit trickier to figure out what $L(v, w)$ should be if v is an internal node. The minimum height depends on the sizes of its children and the orientation of its children. Let's first make the distinction between vertical and horizontal orientation.

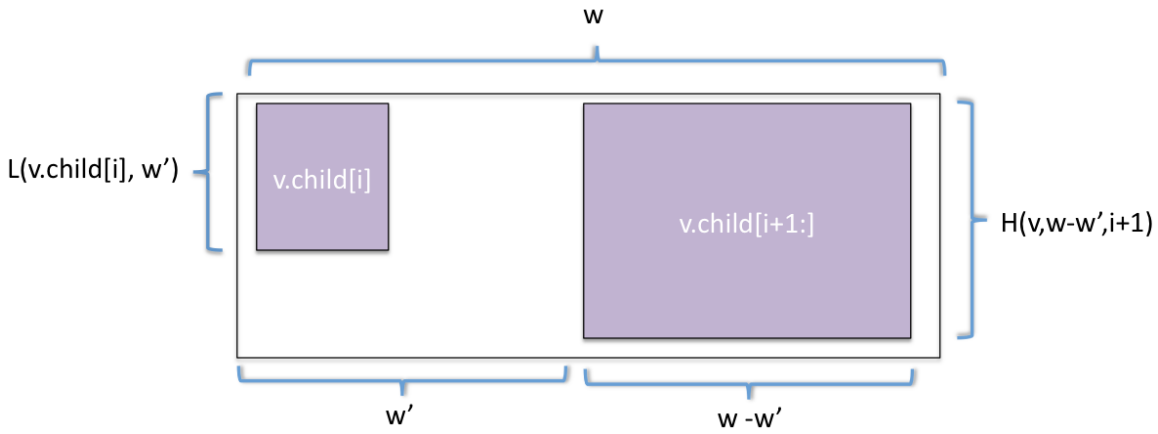
Internal Widgets(Vertical Orientation)

Say we want to find $L_{vert}(v, w)$, the minimum height of v given that we set v to have a vertical orientation. The minimum height in this case will be the sum of the minimum heights of the children given the width constraint w.

$$L_{vert}(\text{internal } v, w) = \text{ sum}(L(c, w) \text{ for } c \text{ in } v.\text{child}) \tag{2}$$

Internal Widgets(Horizontal Orientation)

Now if we wanted to find $L_{horiz}(v, w)$, the minimum height of v given that we set v to have a horizontal orientation, we have to use another level of dynamic programming to solve this subproblem. We introduce $H(v, w, i)$ to be the minimum height such that the horizontal layout of v.child[i:] (meaning the ith child of v to the last child of v) fits into a $w \times h$ rectangle. We eventually want to solve $H(v, w, 1)$, which represents the minimum height of v with all of its children in the horizontal layout. Note that $H(v, w, 1) = L_{horiz}(v, w)$.



In this problem, we want to guess the width of the ith child. Say we guess the width of the ith child to be at most w'. Then we have two components: the ith child and the group formed by the children after the ith child. The heights of these two components are the results of subproblems $L(v.\text{child}[i], w')$ and $H(v, w - w', i + 1)$, whose solutions we should have already calculated before. The height of this configuration is the maximum height of the two components and we want to minimize this height over all possible guesses of w'. Thus, we can construct solutions from subsolutions like so:

$$H(v, w, i) = \text{ min}(\text{ max}\{L(v.\text{ child }[i], w'), \tag{3}$$

$$H(v, w - w', i + 1)\} \text{ for } 1 \leq w' \leq w) \tag{4}$$

In order to ensure that we have all the subsolutions required to construct solutions, we want to iterate i from the last child to the first child. For each child, we need to make guesses for all possible w' from 1 to w. If we follow this order, we will eventually solve $H(v, w, 1)$ and consequently $L_{horiz}(v, w)$.

Internal Widgets

Now we have $L_{vert}(v, w)$ and $L_{horiz}(v, w)$. The last step is to figure out which results in the minimum height. Putting everything together, we get

$$L(\text{internal } v, w) = \text{ min}\{\text{ sum}(L(c, w) \text{ for } c \text{ in } v.\text{child}), \tag{5}$$

$$H(v, w, 1)\} \tag{6}$$

Choosing Order

We now have means to combine subsolutions to create solutions to larger problems. The last step is to choose an order to resolve problems so that we are guaranteed to have all the pieces to solve each problem. We can do this by traversing up the widget hierarchy, i.e. the widget tree.

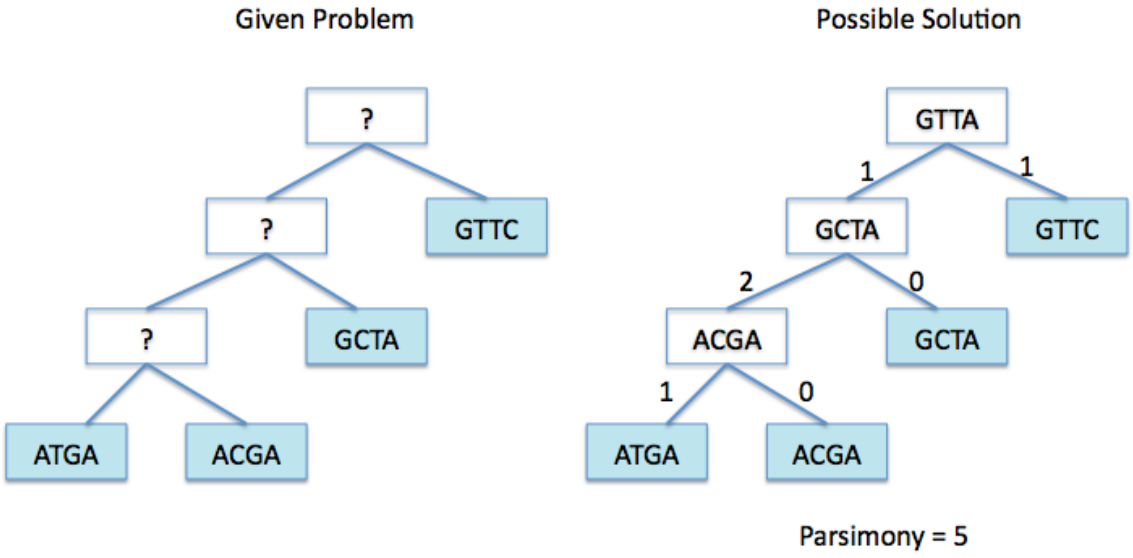
The leaf problems are resolved first since they do not depend on solutions to other problems. The parents of the leaves are then resolved since they just depend on the solutions to the leaf problems. At every level, we make sure to solve for all w from 0 to W before moving on to the next level.

Recursing upwards the tree, we will eventually get to the root widget, where we have the necessary subsolutions to solve $L(root, W)$. If $L(root, W) \leq H$, then the widgets will fit into a $W \times H$ rectangle. Else, it will not.

Dynamic Programming: Parsimony

Setup

We have DNA sequences of n species, all with length m. We are also given a tree in which these n sequences are leaf nodes. For each edge that connects two sequences, we can count the number of mutations (i.e. number of mismatched letters) that occurred on this edge. For example, if there were an edge connecting GCTA with ACGA, we would see that two mutations occurred on this edge (G turned into A, T turned into G).Our goal is to fill in all the non-leaf nodes with DNA sequences to minimize the total number of mutations in the entire tree, accumulated from all the edges.



A key observation here is that each letter is independent of the other letters. That is, we can consider just one letter at a time to figure out what the sequences in the inner nodes should be. We can use dynamic programming to figure out which first letters of the inner nodes produces the least number of total mutations. Then we can figure out which second letters produces the best results and so on. Since all sequences have length m. we essentially can break down this problem into m similar problems where each sequence is length 1 and combine the solutions to these m problems to get the solution that we want.

Defining Subproblems

So now we have reduced the problem to sequences of length 1. We introduce the subproblem $c(v, L)$, representing the minimum cost (number of total mutations) for the subtree rooted at node v if v is labeled some DNA character L. Our solution will be

$$\min\{c(\text{root}, L) \text{ for each character } L\} \tag{7}$$

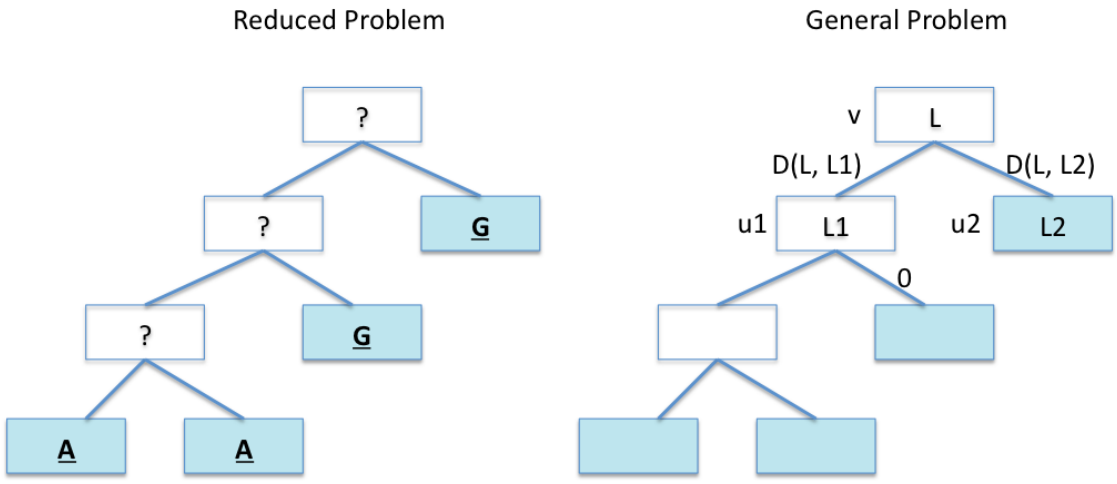
Combining Subproblems

We define letter distance as follows

$$D(L, L') = 0 \text{ if } L = L' \tag{8}$$
$$1 \text{ otherwise} \tag{9}$$

$D(L, L')$ just calculates the number of mutations going from label L to label L'. At some given node v, we can combine smaller subproblems to calculate $c(v, L)$ like so:

$$c(v, L) = \min\{D(L, L_1) + D(L, L_2) + c(u_1, L_1) + c(u_2, L_2) \text{ for each possible } L_1, L_2\} \tag{10}$$



Here, we try out every combination of labeling v's children, u_1 and u_2. Out of all the combinations, we select the labeling that produces the minimum cost. Note that if a child is a leaf node, we define $c(u, L) = 0$ if the label matches the actual sequence at the leaf node, or $c(u, L) = \infty$ if the label mismatches. We do this because our tree would be incorrect if there were any errors at the leaf nodes, we penalize harshly if this happens.

Choosing Order

Like the widget layout example, we want to start with the leaves and make our way up the tree. At each node, iterate through every possible labeling and store the best score for each labeling so that we can use this result to help us solve problems at higher levels. Using the combination of subproblems formula listed above, we will eventually propagate all the solutions up to the root and can tell what the inner node sequences are.