6.006 - Introduction to Algorithms

Lecture 21

Prof. Constantinos Daskalakis

CLRS 15
Menu

• Text Justification
• Structured Dynamic Programming
  – Vertex Cover on trees
  – Parsimony: recovering the tree of life
Menu

• **Text Justification**

• **Structured Dynamic Programming**
  – Vertex Cover on trees
  – Parsimony: recovering the tree of life
Text Justification – Word Processing

• A user writes stream of text
• WP has to break it into lines that aren’t too long
• obvious algorithm => greedy:
  – put as much on first line as possible
  – then continue to lay out rest
  – used by MSWord, OpenOffice
• Problem: suboptimal layouts !!

  e.g.  blah blah blah               blah        blah
        b l a h                       vs          blah        blah
        reallylongword               reallylongword
A Better Approach

• define a scoring rule
  – takes as input partition of words into lines
  – measures how good the layout is
• not an algorithm, just a metric
• find the layout with best score
  – here’s where you think of algorithm
Layout Function

- Want to penalize big spaces. What objective would do that?
  - sum of leftover spaces?
  - then

\[
\begin{array}{c}
\text{blah blah blah} \\
\text{b l a h} \\
\text{reallylongword}
\end{array}
\quad \text{as good as}
\quad
\begin{array}{c}
\text{blah blah} \\
\text{b l a h} \\
\text{reallylongword}
\end{array}
\]

- i.e. it’s the same for two layouts with the same number of lines (just total space minus number of characters)

- should penalize big spaces “extra”
  - (LaTeX uses sum of cubes of leftovers)
Formally

- input: array of words (lengths) $w[0..n]$
- split into lines $L_1, L_2 \ldots$
- **badness of a line:**
  $$\text{badness}(L) = (\text{page width} - \text{total length}(L))^3$$
  $$\text{or } \infty \text{ if total length of line } > \text{page width}$$
- **objective:** break into lines $L_1, L_2 \ldots$ minimizing
  $$\sum_i \text{badness}(L_i)$$
Can We DP?

- Subproblems?
  - $DP[i] = \min \text{ badness for words } w[i:n]$
    (i.e. the score of the best layout of words $w[i],...,w[n]$)
  - $n$ subproblems where $n$ is number of words

- Decision for problem $i$?
  - where to end first line in optimal layout for words $w[i:n]$

- Recurrence?
  - $DP[i] = \min_{j \in \text{range}(i+1,n)} (\text{badness}(w[i:j]) + DP[j])$
  - $DP[n+1]=0$
  - $OPT = DP[0]$

- Runtime? $O(n^2)$?
Menu

• Text Justification
• Structured Dynamic Programming
  – Vertex Cover on trees
  – Parsimony: recovering the tree of life
Vertex cover

- Find a minimum set of vertices that contains at least one endpoint of each edge
- (like placing guards in a house to guard all corridors)
- NP-hard in general
Vertex cover

- Find a minimum set of vertices that contains at least one endpoint of each edge
- (like placing guards in a house to guard all corridors)
- NP-hard in general
- We will see a polynomial (in n) time algorithm for trees of size n
- Ideas?
Vertex cover: algorithm

- Let \( \text{cost}(v, b) \) be the min-cost solution of the sub-tree rooted at \( v \), assuming \( v \)'s status is \( b \in \{\text{YES}, \text{NO}\} \)

- Recurrence for \( \text{cost}(v, b) \)
  \[
  \begin{align*}
  \text{cost}(v, \text{YES}) &= 1 + \min_{b_1} \text{cost}(u_1, b_1) + \min_{b_2} \text{cost}(u_2, b_2) + \cdots \\
  \text{cost}(v, \text{NO}) &= \text{cost}(u_1, \text{YES}) + \text{cost}(u_2, \text{YES}) + \cdots 
  \end{align*}
  \]

- Base case \( v = \text{leaf} \):
  \[
  \begin{align*}
  \text{cost}(v, \text{YES}) &= 1 \\
  \text{cost}(v, \text{NO}) &= 0 
  \end{align*}
  \]

- Running time \( \mathcal{O}(n) \)
The Tree of Life

- 3 million years

time

today
The Computational Problem

- 3 million years

today
The Computational Problem

- 3 million years

time

today
How to score a proposed tree?

- A desired property of a plausible tree: Explains how the observed DNA sequences came about using few mutations.
- Such tree has “high parsimony”.
- Algorithmic problem. Given:
  - n “leaf strings” of length m each, with letters from \{A, C, G, T\}
  - a tree
- Goal: find “inner node” sequences that minimize the sum of all mutations along all edges
- This is the parsimony of the tree.
- Algorithmic Ideas?

\[
\begin{align*}
&\text{parsimony} = 5 \\
&\text{GTTC} \\
&\text{GCTA} \\
&\text{ACGA} \\
&1 \quad 2 \quad 1 \quad 0 \quad 1 \\
&\text{GTAA} \\
&\text{GCTA} \\
&\text{ACGA} \\
&\text{ATGA} \\
&\text{parsimony} = 5
\end{align*}
\]
Parsimony: algorithm

- Observation I: we can consider one letter at a time
- Observation II: can use dynamic programming to find the best inner-node letters
Parsimony: dynamic program

- Define letter distance as follows:
  \[ D(a,b) = 0 \text{ if } a = b \text{ and } 1 \text{ otherwise} \]

- For any node \( v \) of the tree and label \( L \) in \{A, C, G, T\}, define \( \text{cost}(v, L) \):
  - This is the minimum cost for the subtree rooted at \( v \), if \( v \) is labeled \( L \)
  - \( \text{solution} = \min_L \text{cost}(\text{root}, L) \)

- Recurrence for \( \text{cost}(v, L) \) :
  \[ \text{cost}(v, L) = \min_{L_1, L_2} D(L, L_1) + D(L, L_2) + \text{cost}(u_1, L_1) + \text{cost}(u_2, L_2) \]

- Base case: if \( v \) is a leaf
  \[ \text{cost}(v, L) = \infty \cdot D(L, \text{leaf\_label}(v)) \]
Parsimony: analysis

• We have

\[
\text{cost}(v,L) = \min_{L_1,L_2} D(L,L_1) + D(L,L_2) + \text{cost}(u_1,L_1) + \text{cost}(u_2,L_2)
\]

• Equivalently

\[
\text{cost}(v,L) = \min_{L_1} D(L,L_1) + \text{cost}(u_1,L_1) + \min_{L_2} D(L,L_2) + \text{cost}(u_2,L_2)
\]

• Running time?

\[
O(n \, k) \times O(k) = O(nk^2)
\]

where \( k \) is the alphabet size
Time for digression (extracurricular)

- Back to vertex cover
- What if the graph is not a tree?
- Small separator good enough