CS 4800: Algorithms & Data

Lecture 10

February 10, 2017

Packing words into lines

- Sequence of words w_1, …, w_n
- w_i is the width of i-th word
- Want to pack words into lines in the most aesthetically pleasing way

Objective

- Partition words into lines so that
  - Total width of each line is at most page width W

$$(\#words\ on\ line\ i\ ) - 1 + \sum_{j\ on\ line\ i} w_j \leq W$$

Spaces

  - Slack on line i
    $W - (\#words\ on\ line\ i\ ) + 1 - \sum_{j\ on\ line\ i} w_j$
  - Minimize sum of slacks cubed

Identify subproblems

- Best(n): minimum badness for typesetting first n words

Recursive relation

- Want to compute Best(j)
- Suppose the last line consists of
$$w_{i,}w_{i+1}, \dots, w_j$$
- The previous lines should form optimal solution for words $w_{1,}w_2, \dots, w_{i-1}$
  - This is exactly $Best(i-1)$
- Can compute P(i,j): penalty if $w_{i,}w_{i+1}, \dots, w_j$ are used to form a single line
- We don't know the best choice for i, so try all of them
- $Best(j) = \min_{0<i\leq j}\big(Best(i-1) + P(i,j)\big)$

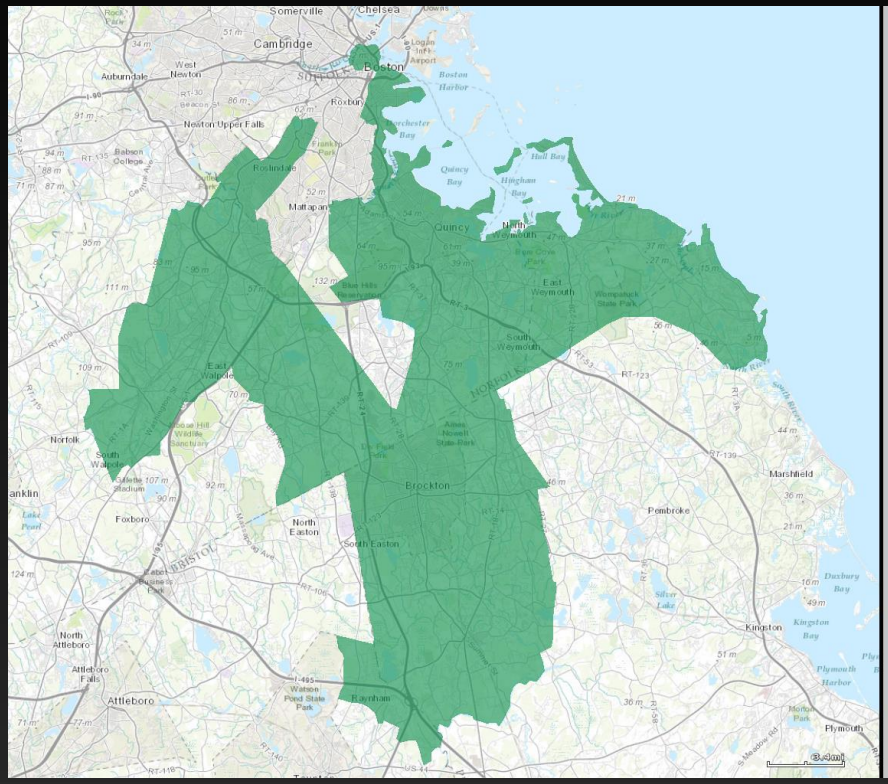
Compute penalties

- $\mathrm{L}(i, j)$: slack of line consisting of words i, i+1, …, j
- $P(i, i) = \begin{cases} (L(i,j)^3 & if\ L(i,j) \geq 0 \\ \infty & other\ wise \end{cases}$
- For $i \leftarrow 1$ to n
  - $\mathrm{L}(i, i) \leftarrow W - w_i$
  - For j $\leftarrow i + 1$ to n
    - $\mathrm{L}(i, j) \leftarrow L(i, j-1) - 1 - w_j$

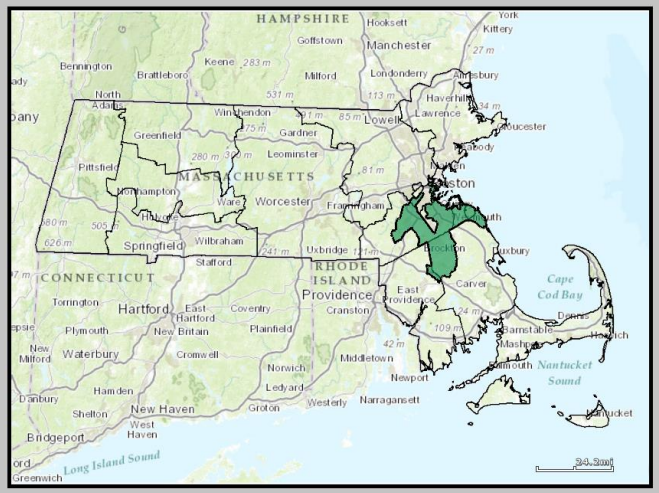
Python code

```
def justify(words, W):
    w = [len(x) for x in words]
    n = len(words)

    # compute slack l[i][j] for line w_i...w_j
    l = [[0]*n for i in range (n)]
    for i in range(n):
        l[i][i] = W - w[i]
        for j in range(i+1, n):
            l[i][j] = l[i][j-1] - 1 - w[j]
    p = [[]] * n
    for i in range(n):
        p[i] = [x*x*x for x in l[i]]

    # compute best
    best = [0] * (n+1)
    choice = [0] * (n+1)
    for i in range(1, n+1):
        j = i
        best[i] = 1000000000
        while (j > 0) and (l[j-1][i-1] >= 0):
            if best[i] > best[j-1] + p[j-1][i-1]:
                best[i] = best[j-1] + p[j-1][i-1]
                choice[i] = j-1
            j -= 1

    # backtrack to find the ends of all lines
    i = n
    ends = [n]
    while i > 0:
        i = choice[i]
        ends.append(i)
    for i in range(len(ends)-1, 0, -1):
        print(" ".join(words[ends[i]:ends[i-1]]))
```

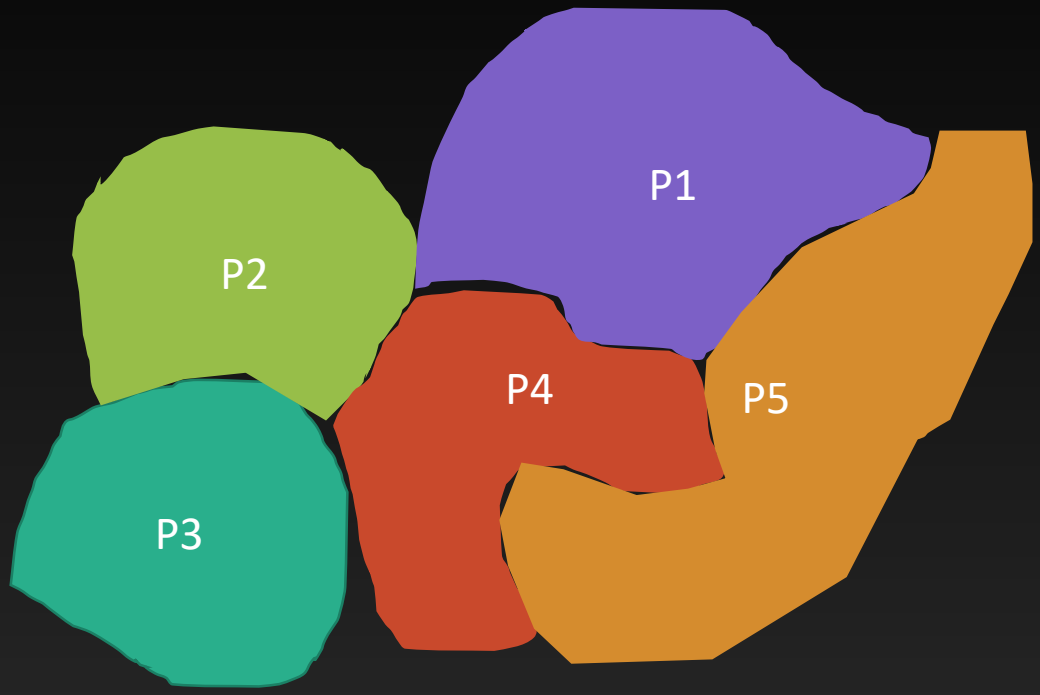
Gerrymander



Massachusetts US District 8



US Congressional districts since 2013
Source: http://nationalatlas.gov, 1 Million Scale project.


P1
P2
P3
P4
P5

Gerrymander

- Given $A_1, A_2, \ldots, A_n$: # supporters for party A in precinct 1, 2, …,n (n even)
- Each precinct has M voters
- Want 2 districts D_1, D_2
  - $|D_1|=|D_2|$: same # precincts per district
  - $A(D_1) > Mn/4$
  - $A(D_2) > Mn/4$

Subproblems

- S(i, k, x, y) = whether among precincts {i, i+1, …, n}, there is a split with
  - k precincts in D_1
  - x votes for A in D_1
  - y votes for A in D_2

Choice for first precinct

- S(i, k, x, y) ?
- Precinct i is in either D_1 or D_2
- S(i, k, x, y) = S(i+1, k-1, x-A_i, y) OR S(i+1, k, x, y-A_i)

Algorithm

- Initialize array S[1..n+1, 0..n, 0..nM, 0..nM] to false
- Initialize S[n+1, 0, 0, 0] = true // no precinct
- For i from n downto 1
  - For k from 0 to n-i+1
    - For x from 0 to (n-i+1)M
      - For y from 0 to (n-i+1)M
        - S[i,k,x,y] = S[i+1, k-1, x-A_i, y] OR S[i+1, k, x, y-A_i]
- Search for true entry among S[1, n/2, >Mn/4, >Mn/4]

Further optimization

- S[i, k, x, y] is true only if $x + y = \sum_{j=1}^{i} A_j$
- S'[i, k, x] is whether among precincts {i, i+1, …, n} there is a split with
  - K precincts in D_1
  - x votes in D_1
  - $\sum_{j=1}^{i} A_j - x$ votes in D_2

Algorithm

- Initialize S'[n+1, 0, 0] = true
- For i from n downto 1
  - For k from 0 to n-i+1
    - For x from 0 to (n-i+1)M
      - S'[i,k,x] = S'[i+1, k-1, x-A_i] OR S'[i+1, k, x]
- Search for true entry among S[1, n/2, x] with $Mn/4 < x < \sum_{j=1}^{n} A_j - Mn/4$

Sequence alignment

-	A	C	T	G	C	T	-	G	T	A
T	A	-	T	G	G	T	A	G	T	A

Comparing genomes

- Given 2 strings/genes
  - $X = x_1x_2…x_m$
  - $Y = y_1y_2…y_n$
- Find alignment of X and Y with min cost
  - Each position in X or Y that is not matched cost 1
  - For each pair of letters p, q, matching p and q incurs mismatch cost of $a_{p,q}$

S	T	E	P	-
-	T	O	-	S
Cost 1		Cost $a_{e,o}$	Cost 1	Cost 1