CS 3100
Data Structures and Algorithms 2
Lecture 19: Longest Common Subsequence

Co-instructors: Robbie Hott and Ray Pettit

Spring 2024

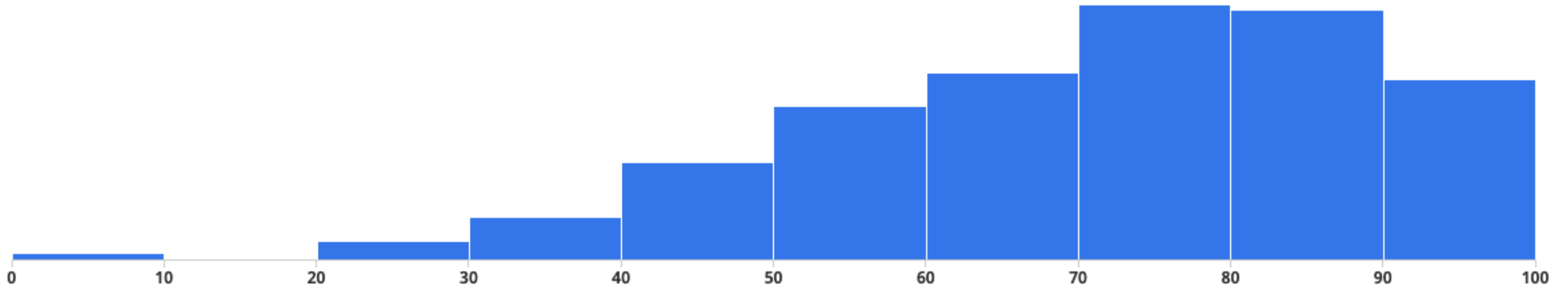
Readings in CLRS 4th edition:

- Chapter 14

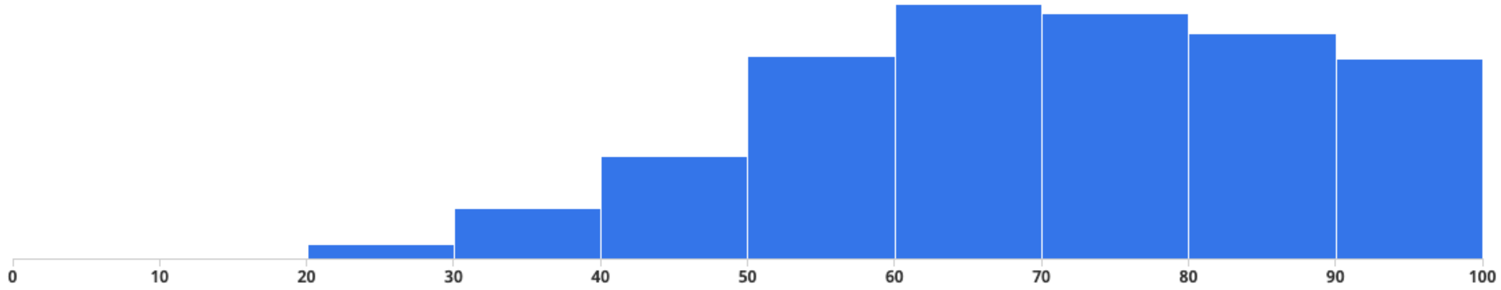
Announcements

- PS8 due tomorrow night
- Quizzes 3-4 next week
  – If you have SDAC, please schedule ASAP
- Grading updates
  – Quiz 2 scores released (mean and median: 70)
  – PS grading caught up! (PS4, 5, and 6 released over the weekend)
- Office hours updates
  – Prof Hott Office Hours:
    - Today 4/2: 2-3pm
    - Back to normal starting Friday

Quiz Statistics



Median	Maximum	Mean	Std Dev
73.0	100.0	70.56	--



Median	Maximum	Mean	Std Dev
70.0	112.0	70.81	18.47

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the (optimal) solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the “last thing” done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

Log Cutting

Given a log of length n

A list (of length n) of prices P ($P[i]$ is the price of a cut of size i)

Find the best way to cut the log

Price:	1	5	8	9	10	17	17	20	24	30
Length:	1	2	3	4	5	6	7	8	9	10

Select a list of lengths $\ell_1, \dots, \ell_k$ such that:

$\sum \ell_i = n$

to maximize $\sum P[\ell_i]$

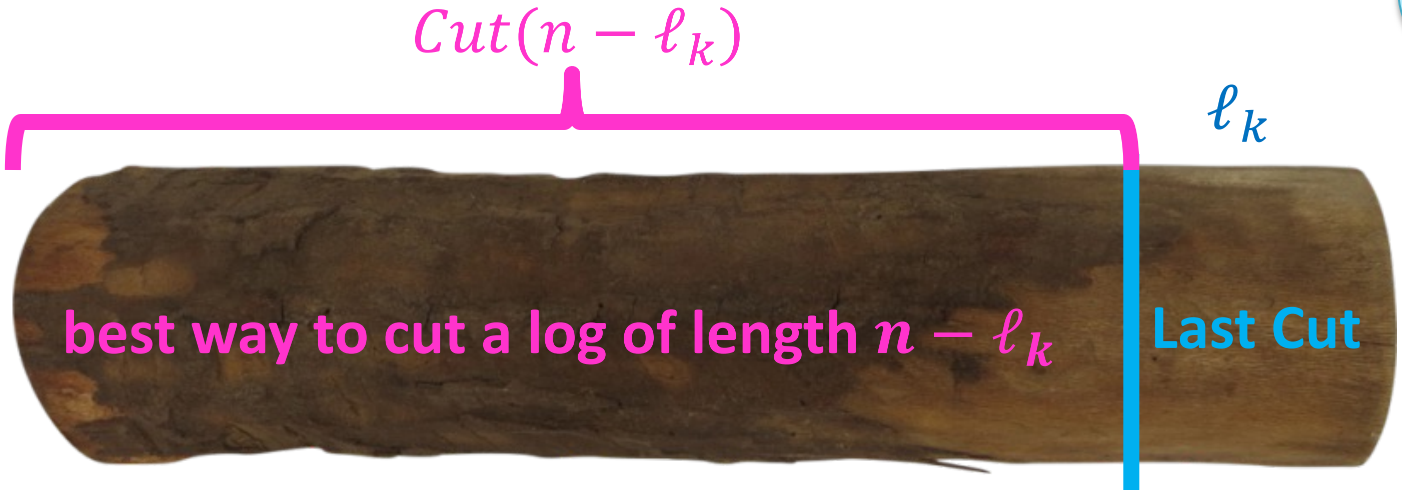
Brute Force: $O(2^n)$

1. Identify Recursive Structure

$P[i]$ = value of a cut of length i

$Cut(n)$ = value of best way to cut a log of length n

$$Cut(n) = \max \begin{cases} Cut(n-1) + P[1] \\ Cut(n-2) + P[2] \\ \dots \\ Cut(0) + P[n] \end{cases}$$



2. Save sub-solutions to memory!

3. Select a Good Order for Solving Subproblems

Solve Smallest subproblem first

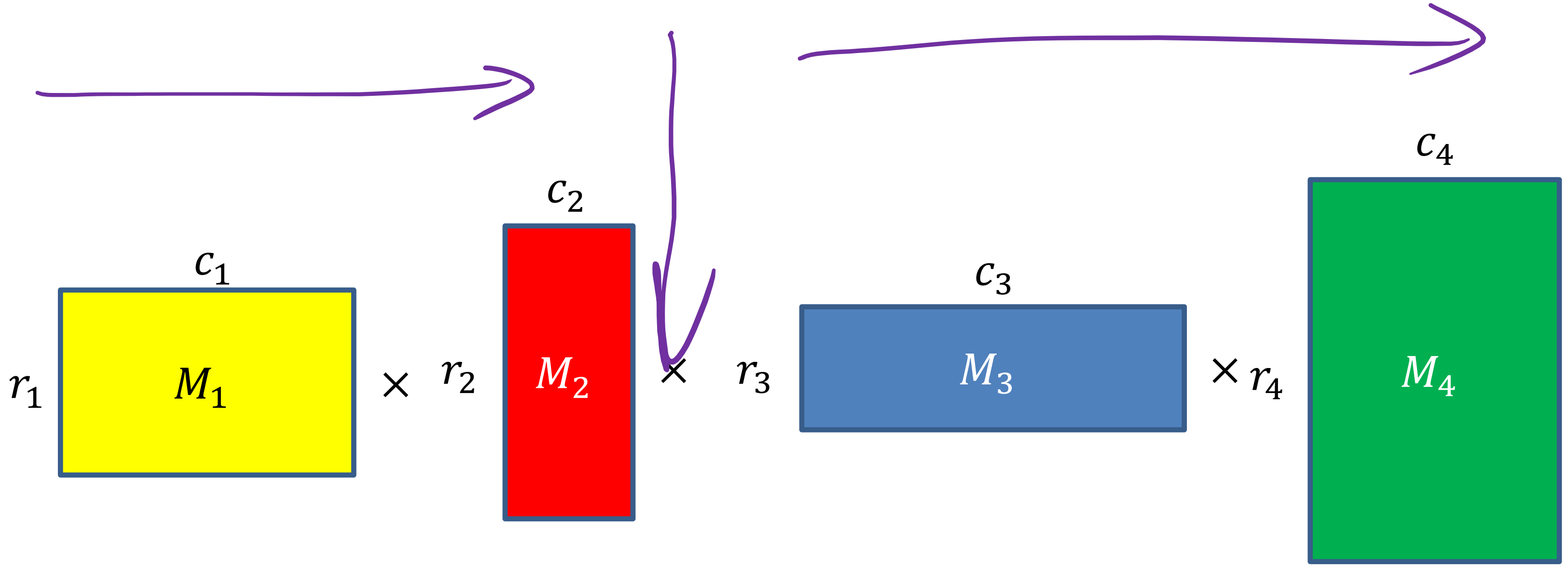
$$Cut(4) = \max \begin{cases} Cut(3) + P[1] \\ Cut(2) + P[2] \\ Cut(1) + P[3] \\ Cut(0) + P[4] \end{cases}$$

$Cut(i)$:	0										
Length:	0	1	2	3	4	5	6	7	8	9	10

4

Matrix Chaining

- Given a sequence of Matrices $(M_1, \dots, M_n)$, what is the most efficient way to multiply them?



1. Identify the Recursive Structure of the Problem

- In general:

$Best(i,j) =$ cheapest way to multiply together M_i through M_j

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$$Best(1,n) = \min \begin{cases} Best(2,n) + r_1 r_2 c_n \\ Best(1,2) + Best(3,n) + r_1 r_3 c_n \\ Best(1,3) + Best(4,n) + r_1 r_4 c_n \\ Best(1,4) + Best(5,n) + r_1 r_5 c_n \\ \quad \dots \\ Best(1,n-1) + r_1 r_n c_n \end{cases}$$

2. Save Subsolutions in Memory

- In general:

$Best(i, j)$ = cheapest way to multiply together M_i through M_j

$$Best(i, j) = \min_{k=i}^{j-1}\left(Best(i, k) + Best(k+1, j) + r_i r_{k+1} c_j\right)$$

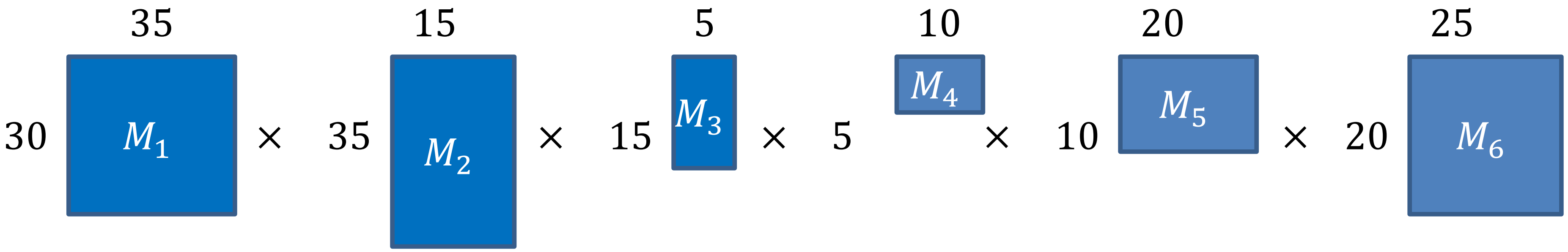
$$Best(i, i) = 0$$

Read from M[n] if present

Save to M[n]

$$Best(1, n) = \min \begin{cases} Best(2, n) + r_1 r_2 c_n \\ Best(1,2) + Best(3, n) + r_1 r_3 c_n \\ Best(1,3) + Best(4, n) + r_1 r_4 c_n \\ Best(1,4) + Best(5, n) + r_1 r_5 c_n \\ \ldots \\ Best(1, n-1) + r_1 r_n c_n \end{cases}$$

3. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$= i$
	0	15750	7875				1
		0	2625				2
			0	750			3
				0	1000		4
					0	5000	5
						0	6

To find $Best(i,j)$: Need all preceding terms of row i and column j

Conclusion: solve in order of diagonal

Coin Changing: Identify Recursive Structure

$\text{Change}(n)$: minimum number of coins needed to give change for n cents

Possibilities for last coin	Coins needed	
	$\text{Change}(n-25)+1$	if $n \geq 25$
	$\text{Change}(n-11)+1$	if $n \geq 11$
	$\text{Change}(n-10)+1$	if $n \geq 10$
	$\text{Change}(n-5)+1$	if $n \geq 5$
	$\text{Change}(n-1)+1$	if $n \geq 1$

Identify Recursive Structure

$\text{Change}(n)$: minimum number of coins needed to give change for n cents

$$\text{Change}(n) = \min \begin{cases} \text{Change}(n-25)+1 & \text{if } n \geq 25 \\ \text{Change}(n-11)+1 & \text{if } n \geq 11 \\ \text{Change}(n-10)+1 & \text{if } n \geq 10 \\ \text{Change}(n-5)+1 & \text{if } n \geq 5 \\ \text{Change}(n-1)+1 & \text{if } n \geq 1 \end{cases}$$

Correctness: The optimal solution must be contained in one of these configurations

Base Case: $\text{Change}(0) = 0$

Running time: $O(kn)$

k is number of possible coins

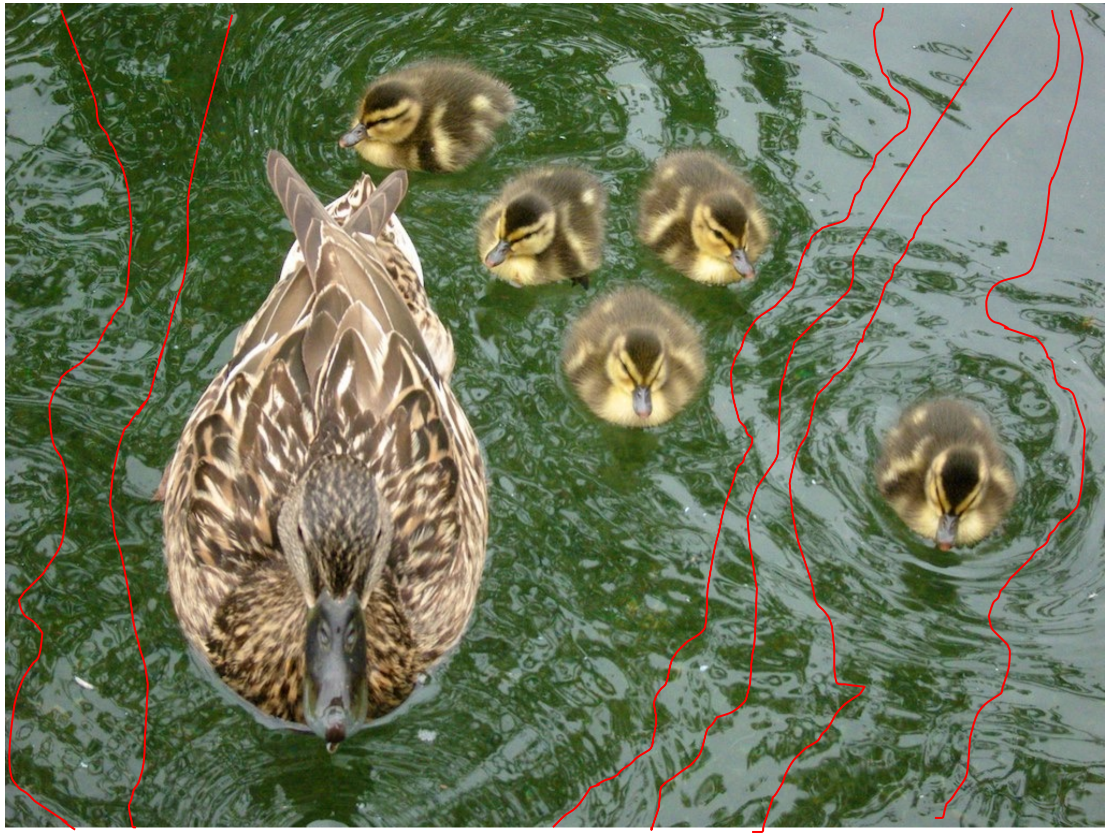
Is this efficient?

Input size is $O(k \log n)$

No, this is pseudo-polynomial time

Seam Carving

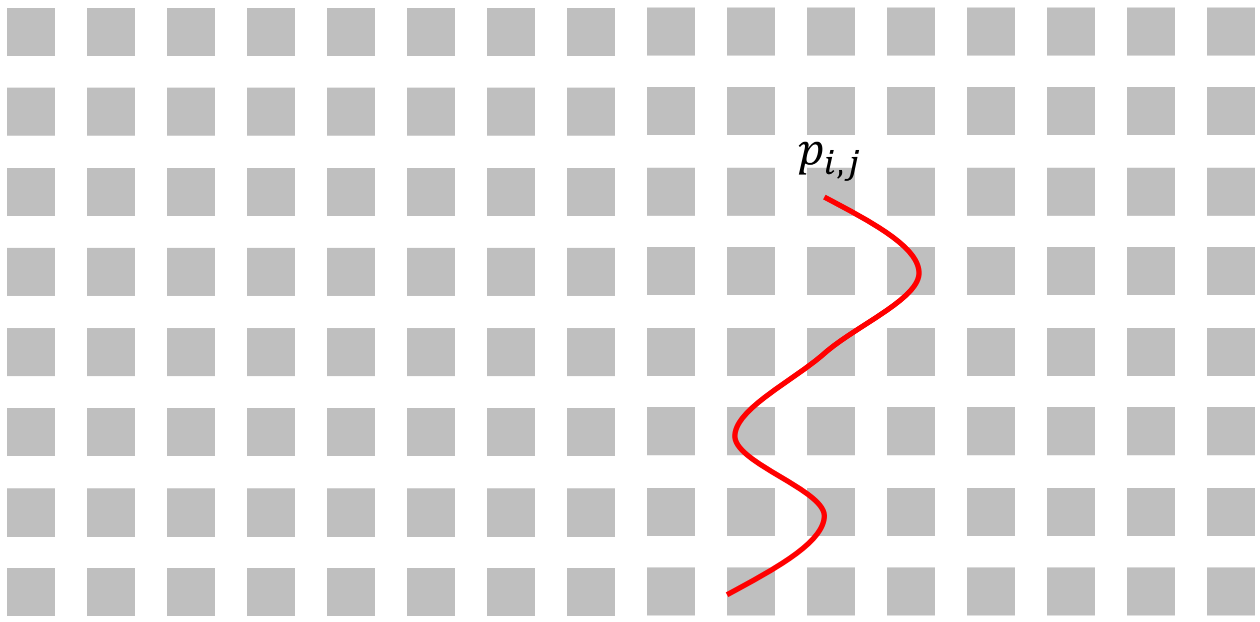
- Removes "least energy seam" of pixels
- https://trekhleb.dev/js-image-carver/



Carved

Identify Recursive Structure

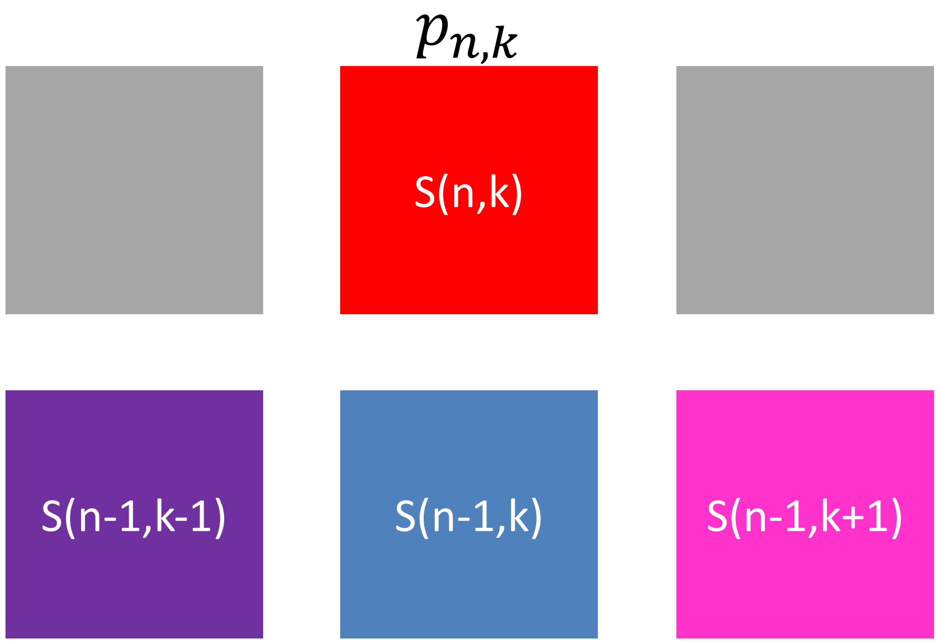
Let $S(i,j)$ = least energy seam from the bottom of the image up to pixel $p_{i,j}$



Computing $S(n,k)$

Assume we know the least energy seams for all of row $n-1$
(i.e. we know $S(n-1,\ell)$ for all ℓ)

$$S(n,k) = min\begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$$



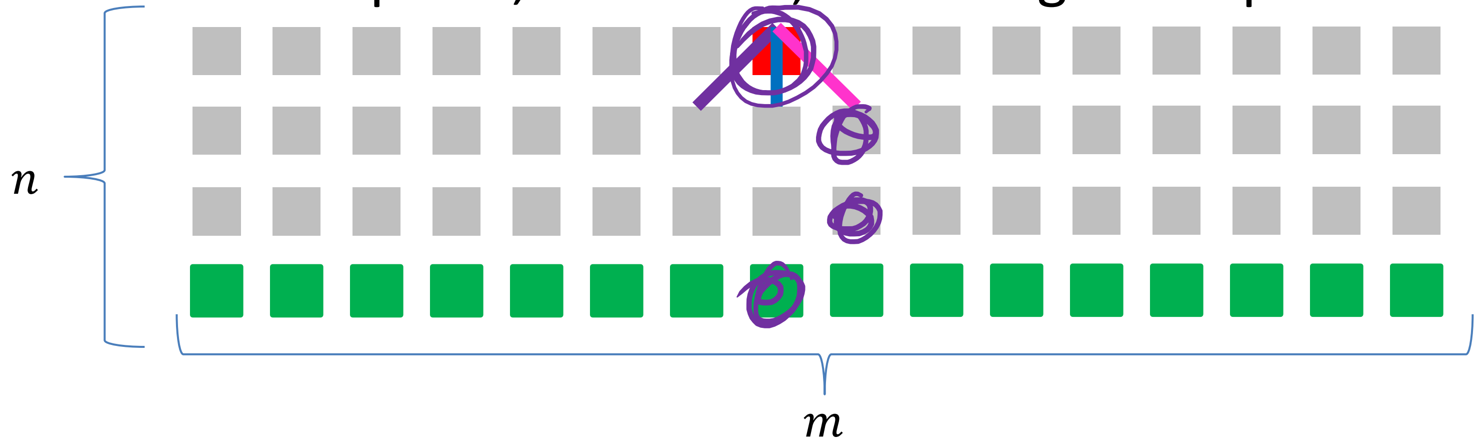
Finding the Seam

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$

For $i > 2$ find $S(i,k) = \min \begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$

Pick smallest from top row, backtrack, removing those pixels



Energy of the seam initialized to the energy of that pixel

Longest Common Subsequence

Given two sequences X and Y, find the length of their longest common subsequence

Example:
$\mathrm{X} = TGCATA$
$\mathrm{Y} = ATCTGAT$
$LCS = TCTA$

Brute force: Compare every subsequence of X with Y
$\Omega(2^n)$

1. Identify Recursive Structure

Let $LCS(i,j)$ = length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i,j)$:

Case 1: $X[i] = Y[j]$

$$X = TGCATAT$$
$$Y = ATCTGCGT$$

$$LCS(i,j) = LCS(i-1, j-1) + 1$$

Case 2: $X[i] \neq Y[j]$

$$X = TGCATAC$$
$$Y = ATCTGCGT$$

max ($LCS(i,j) = LCS(i, j-1)$,

$$X = TGCATAT$$
$$Y = ATCTGCGA$$

$LCS(i,j) = LCS(i-1, j)$)

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  – Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

1. Identify Recursive Structure

Let $LCS(i,j)$ = length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i,j)$:

Case 1: $X[i] = Y[j]$

$X = TGCATAT$
$Y = ATCTGCGT$

$$LCS(i,j) = LCS(i-1, j-1) + 1$$

Case 2: $X[i] \neq Y[j]$

$X = TGCATAC$
$Y = ATCTGCGT$

$$LCS(i,j) = LCS(i, j-1)$$

$X = TGCATAT$
$Y = ATCTGCGA$

$$LCS(i,j) = LCS(i-1, j)$$

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Read from M[i,j] if present

Save to M[i,j]

Top-Down Solution with Memoization

We need two functions; one will be recursive.

```
LCS-Length(X, Y) // Y is M's cols.
1. n = length(X)
2. m  = length(Y)
3. Create table M[n,m]
4. Assign -1 to all cells M[i,j]
// get value for entire sequences
5. return LCS-recur(X, Y, M, n, m)
```

```
LCS-recur(X, Y, M, i, j)
1. if (i == 0 || j == 0) return 0
// have we already calculated this subproblem?
2. if (M[i,j] != -1) return M[i,j]
3. if ( X[i] == Y[j] )
4.   M[i,j] = LCS-recur(X, Y, M, i-1, j-1) + 1
5. else
6.   M[i,j] = max( LCS-recur(X, Y, M, i-1, j),
                   LCS-recur(X, Y, M, i, j-1) )
7. return M[i,j]
```

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

3. Solve in a Good Order

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Y =

X =

		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	0	0	0	0	0	0	0	0
T	1	0	0	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	2	2
A	4	0	1	1	2	2	2	3	3
T	5	0	1	2	2	3	3	3	4
A	6	0	1	2	2	3	3	4	4

To fill in cell (i,j) we need cells $(i-1, j-1), (i-1, j), (i, j-1)$

Fill from Top->Bottom, Left->Right (with any preference)

LCS Length Algorithm

LCS-Length(X, Y) // Y for M's rows, X for its columns

1. n = length(X) // get the # of symbols in X
2. m = length(Y) // get the # of symbols in Y
3. for i = 0 to n M[i,0] = 0 // special case: X_0
4. for j = 0 to m M[0,j] = 0 // special case: Y_0
5. for i = 1 to n // for all X_i
6. for j = 1 to m // for all Y_j
7. if (X[i] == Y[j])
8. M[i,j] = M[i-1,j-1] + 1
9. else M[i,j] = max(M[i-1,j], M[i,j-1])
10. return M[n,m] // return LCS length for Y and X

Run Time?

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$; $X =$

		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	0	0	0	0	0	0	0	0
T	1	0	0	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	2	2
A	4	0	1	1	2	2	2	3	3
T	5	0	1	2	2	3	3	3	4
A	6	0	1	2	2	3	3	4	4

Run Time: $\Theta(n \cdot m)$ (for $|X| = n$, $|Y| = m$)

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$X =$ \ $Y =$		0	A 1	T 2	C 3	$\boxed{T}$ 4	$\boxed{G}$ 5	$\boxed{A}$ 6	$\boxed{T}$ 7
	0	0	0	0	0	0	0	0	0
$\boxed{T}$	1	0	0	1	1	1	1	1	1
$\boxed{G}$	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	2	2
$\boxed{A}$	4	0	1	1	2	2	2	3	3
$\boxed{T}$	5	0	1	2	2	3	3	3	4
A	6	0	1	2	2	3	3	4	4

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$X =$ \ $Y =$		0	A 1	$\boxed{T}$ 2	$\boxed{C}$ 3	T 4	G 5	$\boxed{A}$ 6	$\boxed{T}$ 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\boxed{T}$	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
$\boxed{C}$	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
$\boxed{A}$	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
$\boxed{T}$	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

Start from bottom right,

if symbols matched, print that symbol then go diagonally

else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$X =$ \ $Y =$		0	$\boxed{A}$ 1	$\boxed{T}$ 2	$\boxed{C}$ 3	$\boxed{T}$ 4	G 5	$\boxed{A}$ 6	T 7
	0	0	0	0	0	0	0	0	0
$\boxed{T}$	1	0	0	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2
$\boxed{C}$	3	0	0	1	2	2	2	2	2
A	4	0	1	1	2	2	2	3	3
$\boxed{T}$	5	0	1	2	2	3	3	3	4
$\boxed{A}$	6	0	1	2	2	3	3	4	4

Start from bottom right,

if symbols matched, print that symbol then go diagonally

else go to largest adjacent

Supreme Court Associate Justice Anthony Kennedy gave no sign that he has abandoned his view that extreme partisan gerrymandering might violate the Constitution. I Eric Thayer/Getty Images

Supreme Court eyes partisan gerrymandering

Anthony Kennedy is seen as the swing vote that could blunt GOP's map-drawing successes.

SUPREME COURT OF THE UNITED STATES

Syllabus

VIRGINIA HOUSE OF DELEGATES ET AL. *v.* BETHUNE-HILL ET AL.

APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE EASTERN DISTRICT OF VIRGINIA

No. 18–281. Ar

After the 2010 cens
State's Senate and
districts sued two s
ly, State Defendan
cially gerrymander
Equal Protection C
(collectively, the H
the bench trial, on
where a three-judg
unconstitutionally
tions for those dist
General Assembly
torney General ann
to this Court. The
Held: The House lack
ests or in its own ri

SUPREME COURT OF THE UNITED STATES

Syllabus

RUCHO ET AL. *v.* COMMON CAUSE ET AL.

APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE MIDDLE DISTRICT OF NORTH CAROLINA

No. 18–422. Argued March 26, 2019—Decided June 27, 2019*

Voters and other plaintiffs in North Carolina and Maryland filed suits
challenging thei
tutional partisa
claimed that the
crats, while the
discriminated ag
of the First Am
teenth Amendme
trict Courts in b
fendants appeale
Held: Partisan ger
yond the reach o
(a) In these ca
tion of constituti
the question is p
Judiciary Natur
342. While it is

Next Gerrymanderin
in North Carolina: C

A North Carolina court threw out
an illegal gerrymander. Now the s
state to redraw the state's congres

The New York Times

How to Police Gerrymanders? Some Judges Say the Courts Can't.

A North Carolina court, following the lead of the U.S. Supreme Court, ruled that courts don't have the ability to determine if a political map is legal, giving legislators a free pass.

Share full article

Gerrymandering

- Manipulating electoral district boundaries to favor one political party over others
- Coined in an 1812 Political cartoon
- Governor **Elbridge Gerry** signed a bill that redistricted Massachusetts to benefit his Democratic-Republican Party

The Gerrymander

According to the Supreme Court

- Gerrymandering cannot be used to:
  - Disadvantage racial/ethnic/religious groups
- It can be used to:
  - Disadvantage political parties

SUPREME COURT OF THE UNITED STATES

Syllabus

VIRGINIA HOUSE OF DELEGATES ET AL. *v.* BETHUNE-HILL ET AL.

APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE EASTERN DISTRICT OF VIRGINIA

No. 18–281. Argued March 18, 2019—Decided June 17, 2019

After the 2010 census, Virginia redrew legislative districts for the State's Senate and House of Delegates. Voters in 12 impacted House districts sued two state agencies and four election officials (collectively, State Defendants), charging that the redrawn districts were racially gerrymandered in violation of the Fourteenth Amendment's Equal Protection Clause. The House of Delegates and its Speaker (collectively, the House) intervened as defendants, participating in the bench trial, on appeal to this Court, and at a second bench trial, where a three-judge District Court held that 11 of the districts were unconstitutionally drawn, enjoined Virginia from conducting elections for those districts before adoption of a new plan, and gave the General Assembly several months to adopt that plan. Virginia's Attorney General announced that the State would not pursue an appeal to this Court. The House, however, did file an appeal.

Held: The House lacks standing, either to represent the State's interests or in its own right. Pp. 3–12.

SUPREME COURT OF THE UNITED STATES

Syllabus

RUCHO ET AL. *v.* COMMON CAUSE ET AL.

APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE MIDDLE DISTRICT OF NORTH CAROLINA

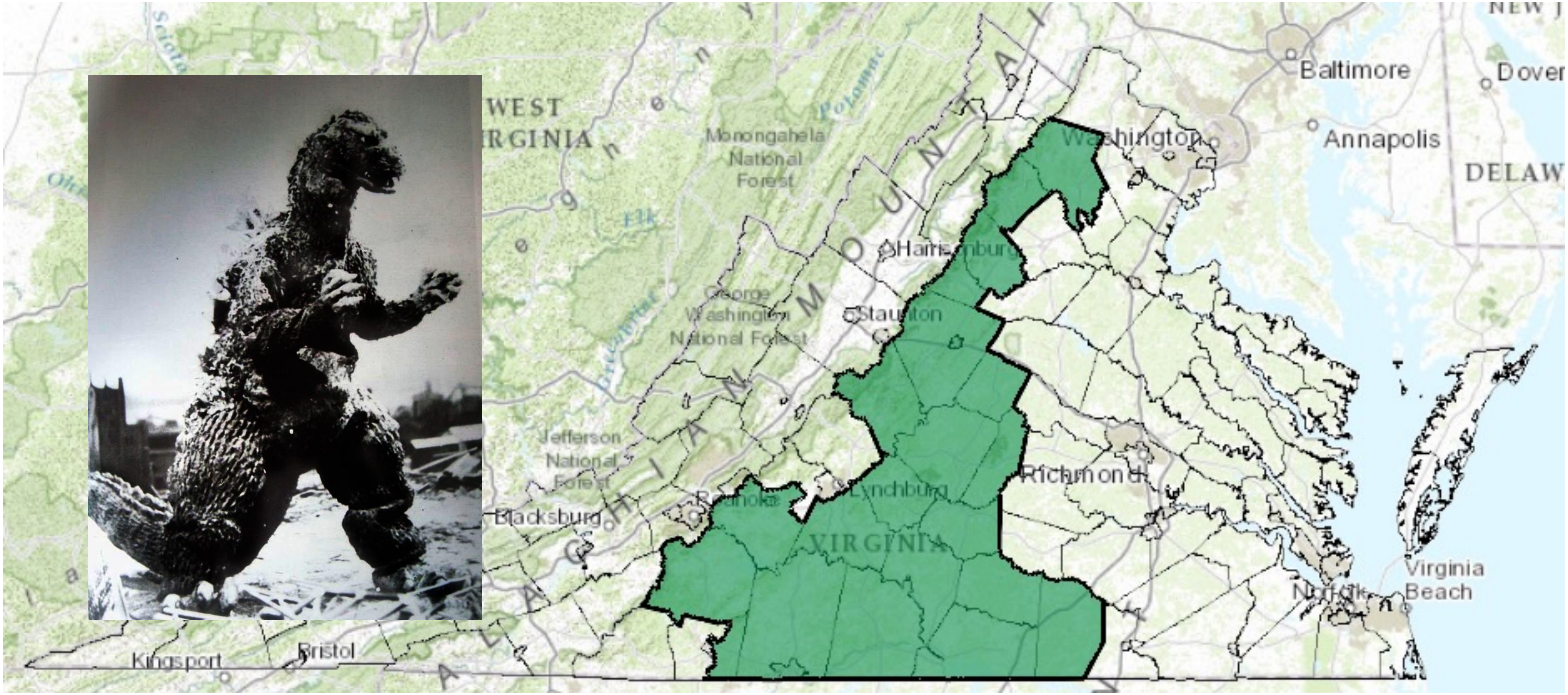
No. 18–422. Argued March 26, 2019—Decided June 27, 2019*

Voters and other plaintiffs in North Carolina and Maryland filed suits challenging their States' congressional districting maps as unconstitutional partisan gerrymanders. The North Carolina plaintiffs claimed that the State's districting plan discriminated against Democrats, while the Maryland plaintiffs claimed that their State's plan discriminated against Republicans. The plaintiffs alleged violations of the First Amendment, the Equal Protection Clause of the Fourteenth Amendment, the Elections Clause, and Article I, §2. The District Courts in both cases ruled in favor of the plaintiffs, and the defendants appealed directly to this Court.

Held: Partisan gerrymandering claims present political questions beyond the reach of the federal courts. Pp. 6–34.

(a) In these cases, the Court is asked to decide an important question of constitutional law. Before it does so, the Court "must find that the question is presented in a 'case' or 'controversy' that is . . . 'of a Judiciary Nature.'" *DaimlerChrysler Corp.* v. *Cuno*, 547 U. S. 332, 342. While it is "the province and duty of the judicial department to

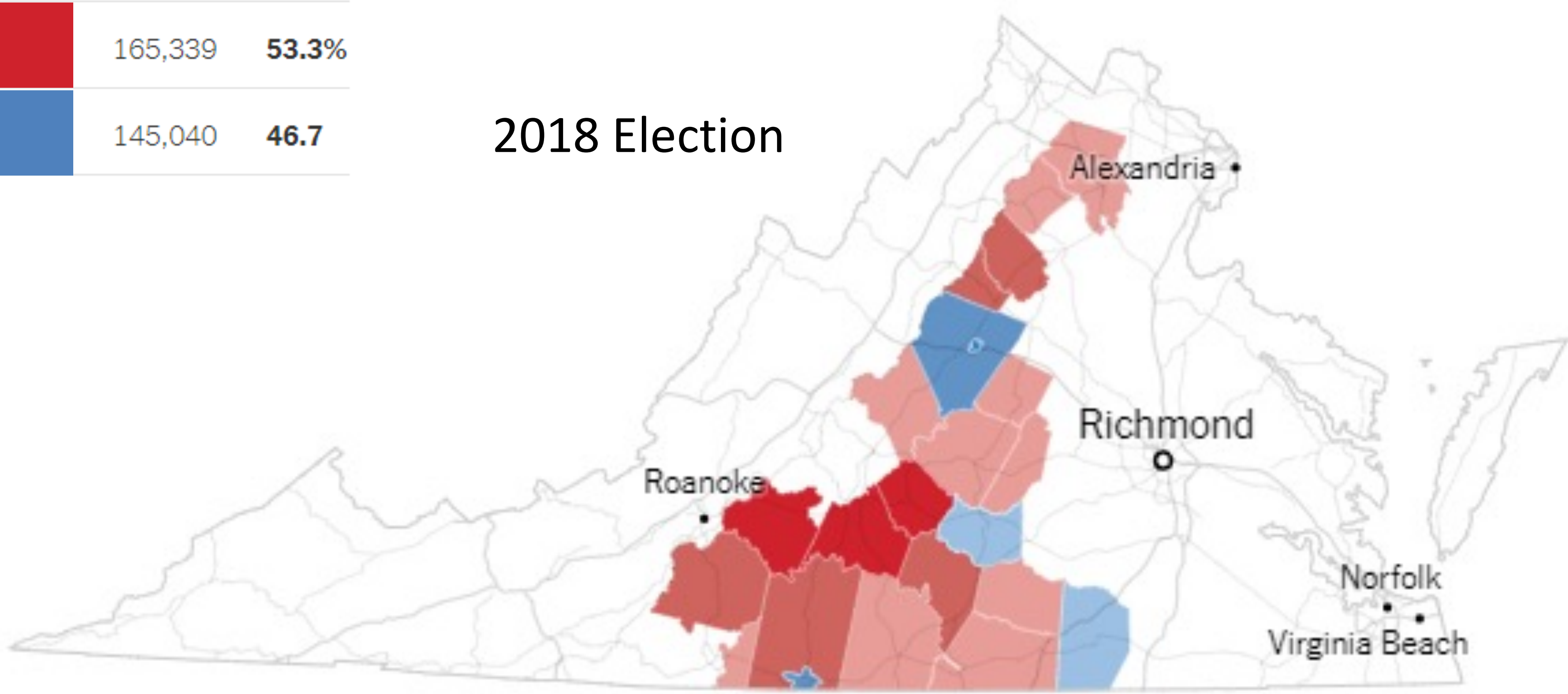
VA 5th District



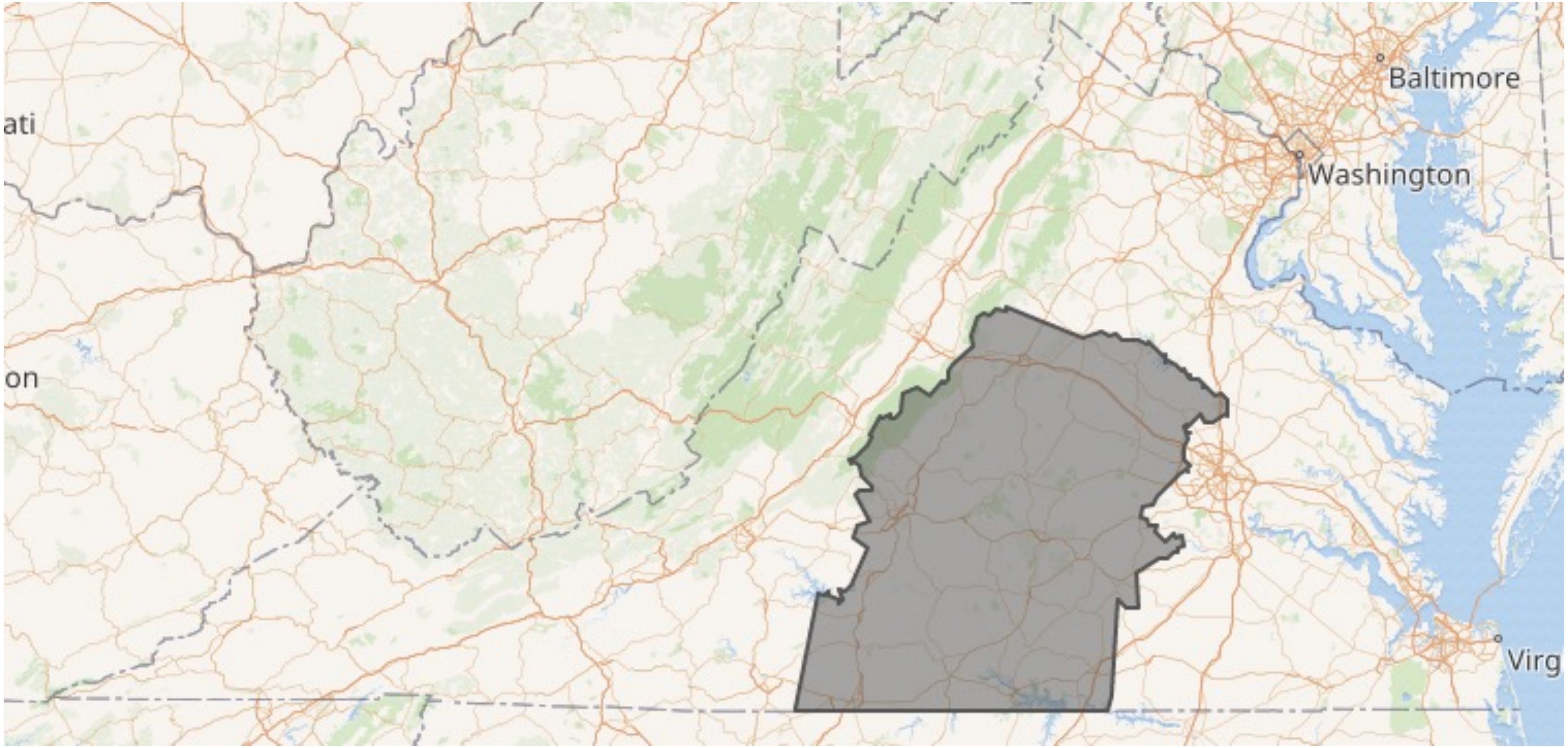
VA 5th District

	Votes	Pct.
	165,339	**53.3%**
	145,040	**46.7**

2018 Election

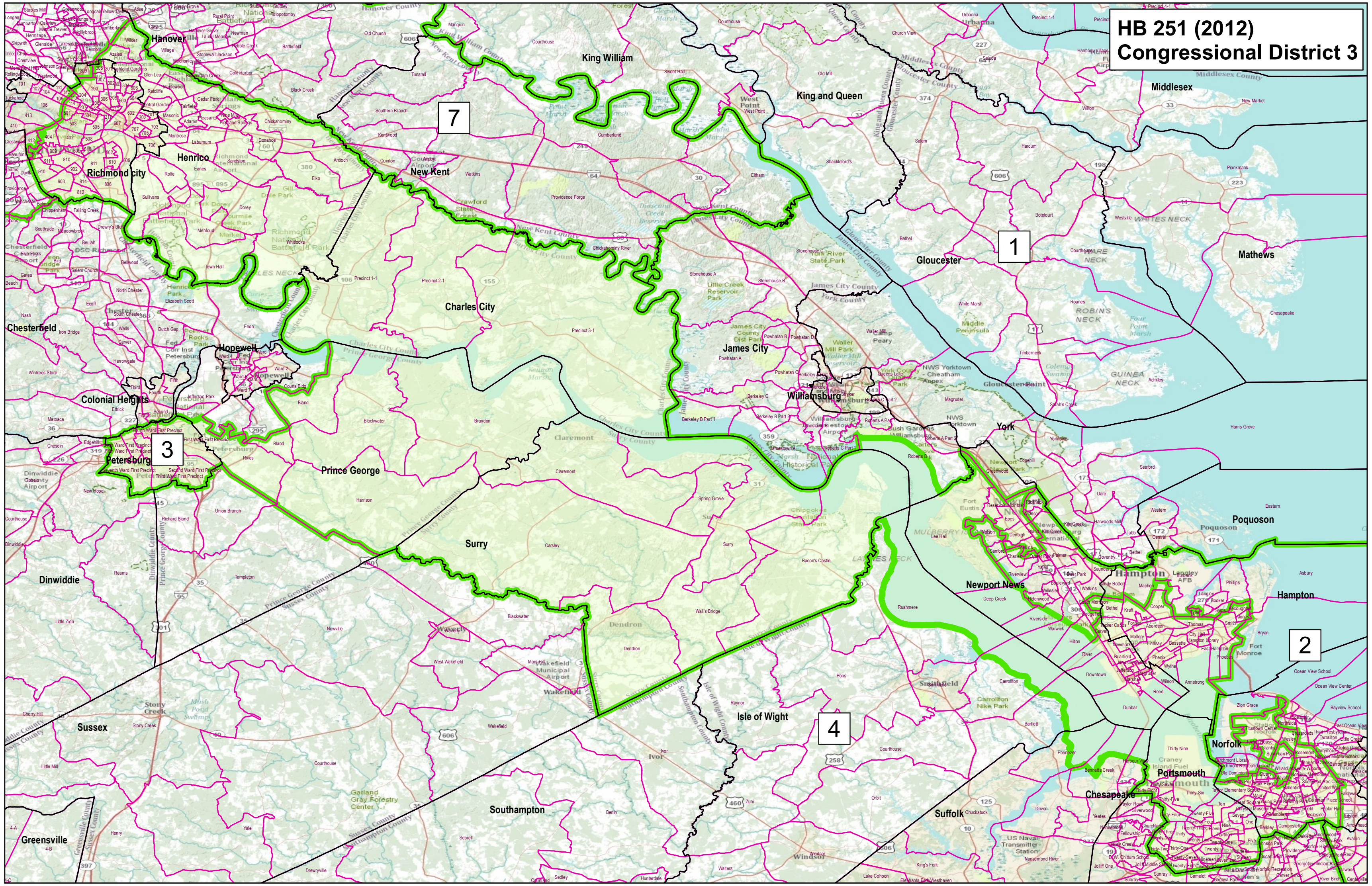


VA 5th District (today)



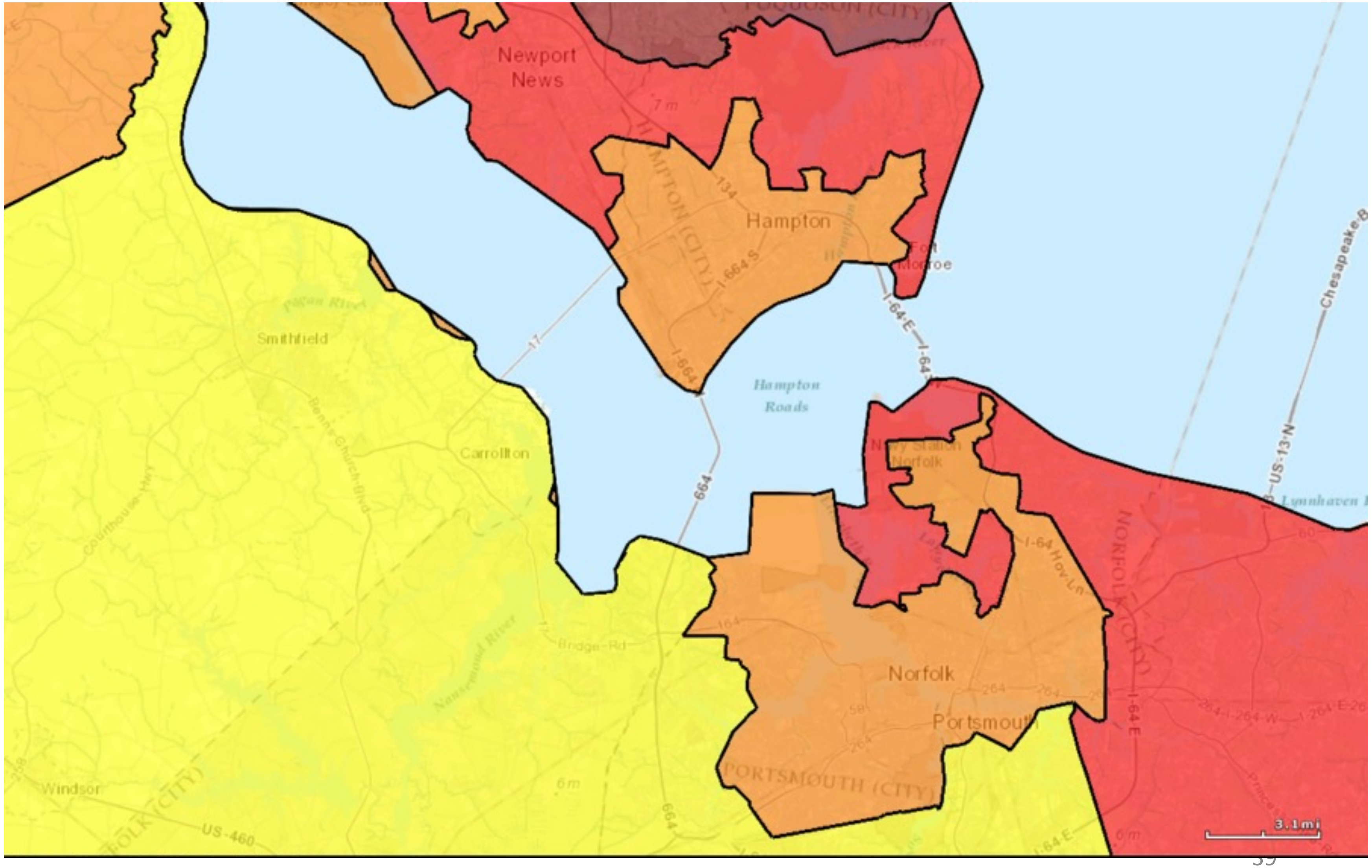
Gerrymandering Today

- Computers make it really effective



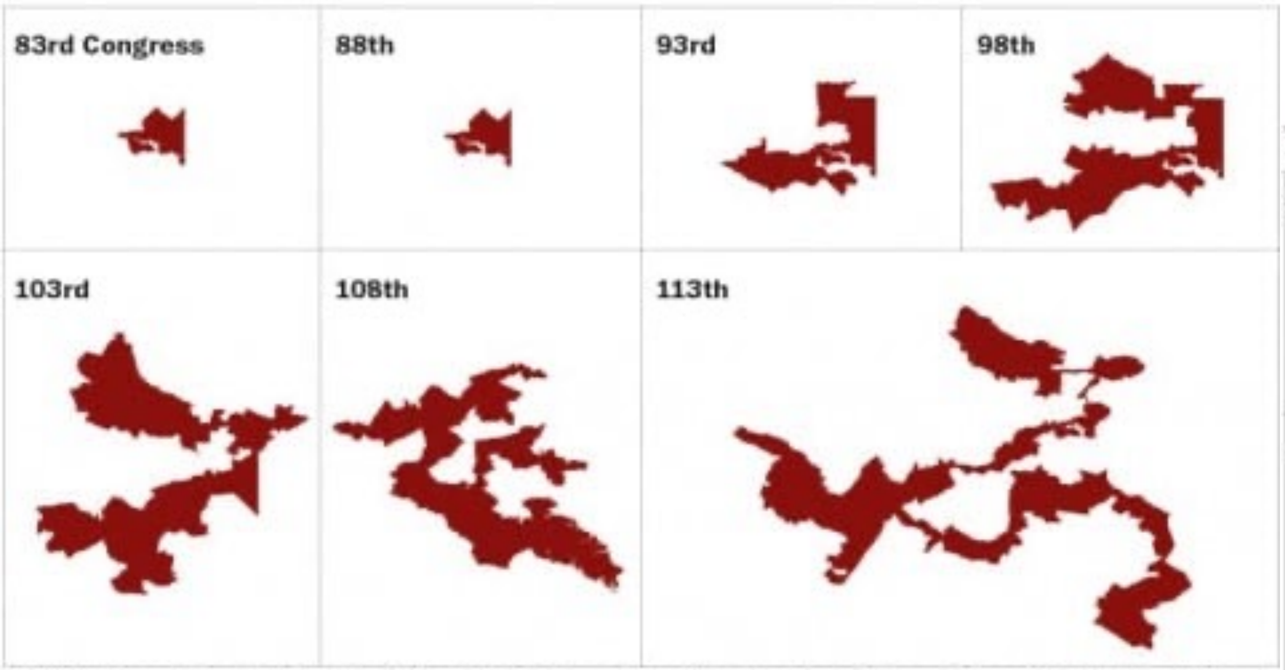
Gerrymandering Today

- Computers make it really effective



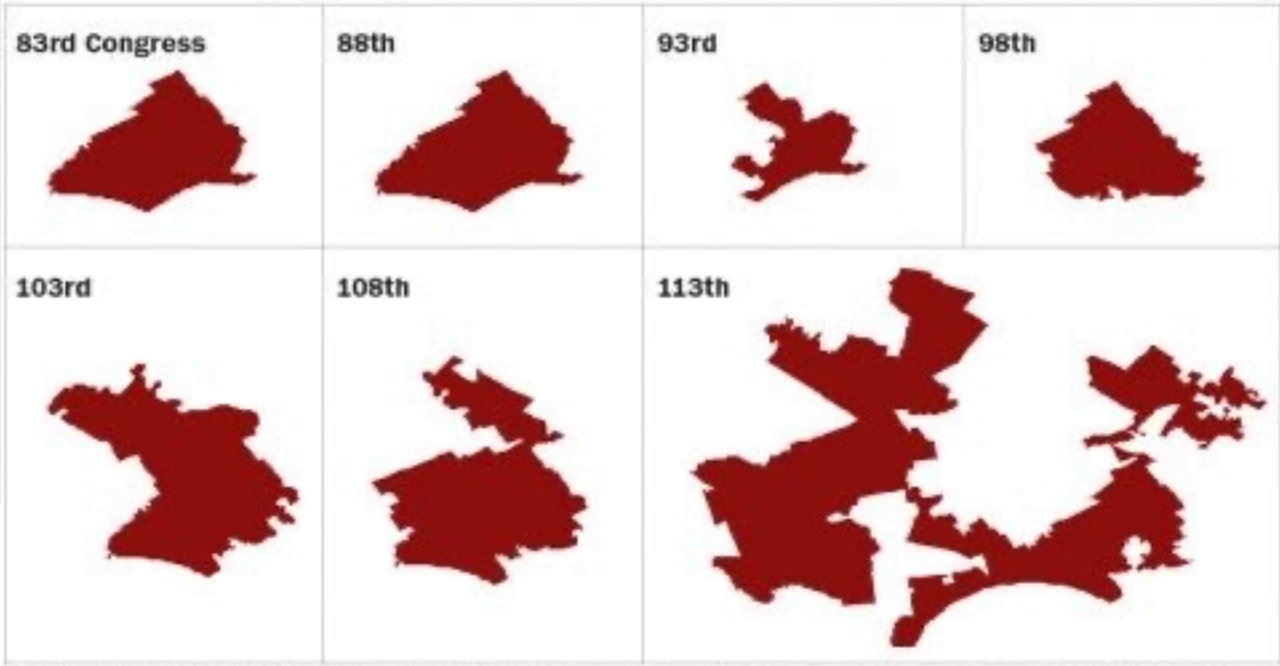
Gerrymandering Today

THE EVOLUTION OF MARYLAND'S THIRD DISTRICT



SOURCE: Shapefiles maintained by Jeffrey B. Lewis, Brandon DeVine, Lincoln Pritcher and Kenneth C. Martis, UCLA.
Drawn to scale.
GRAPHIC: The Washington Post. Published May 20, 2014

THE EVOLUTION OF PENNSYLVANIA'S SEVENTH DISTRICT



SOURCE: Shapefiles maintained by Jeffrey B. Lewis, Brandon DeVine, Lincoln Pritcher and Kenneth C. Martis, UCLA.
Drawn to scale.
GRAPHIC: The Washington Post. Published May 20, 2014

How does it work?

- States are broken into precincts
- All precincts have the same size
- We know voting preferences of each precinct
- Group precincts into districts to maximize the number of districts won by my party

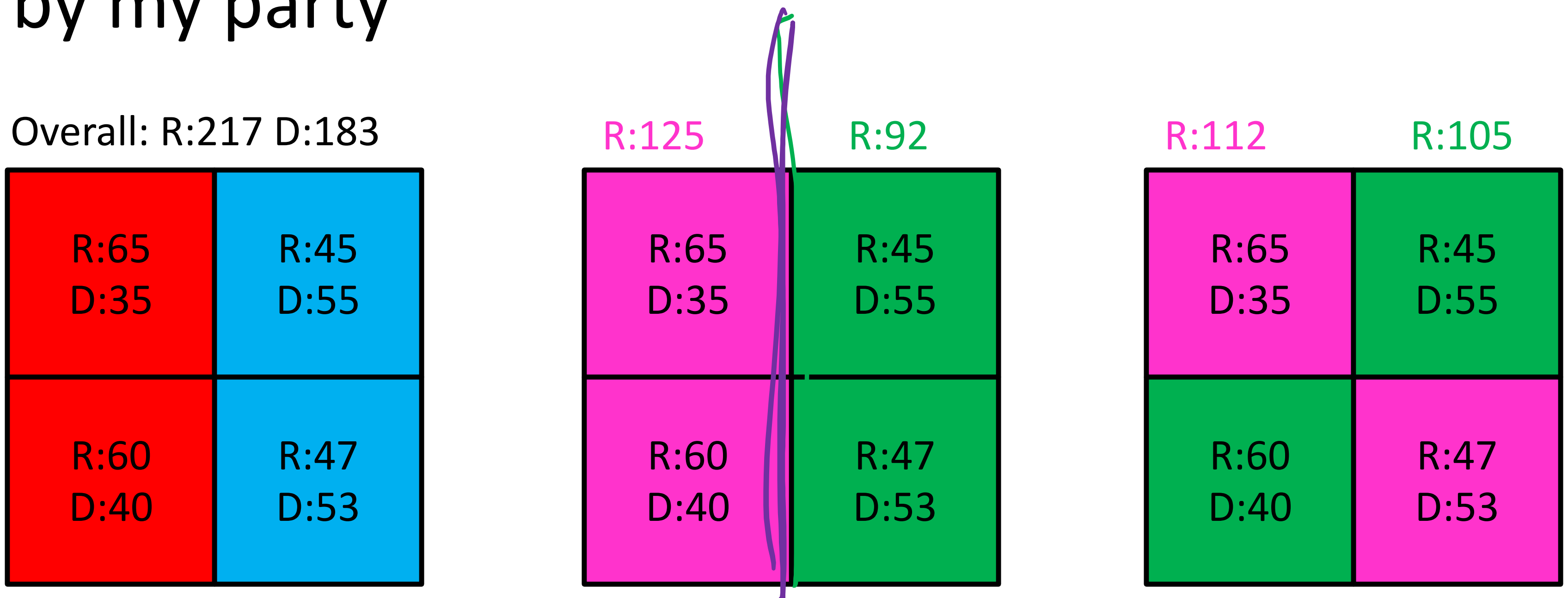
Overall: R:217 D:183

R:65 D:35	R:45 D:55
R:60 D:40	R:47 D:53

vs

How does it work?

- States are broken into precincts
- All precincts have the same size
- We know voting preferences of each precinct
- Group precincts into districts to maximize the number of districts won by my party



Gerrymandering Problem Statement

- Given:
  - A list of precincts: $p_1, p_2, \ldots, p_n$
  - Each containing m voters
- Output:
  - Districts $D_1, D_2 \subset \{p_1, p_2, \ldots, p_n\}$
  - Where $|D_1| = |D_2|$
  - $R(D_1) > \frac{mn}{4}$ and $R(D_2) > \frac{mn}{4}$
    - $R(D_i)$ gives number of "Regular Party" voters in D_i
    - $R(D_i) > \frac{\text{mn}}{4}$ means D_i is majority "Regular Party"
  - "failure" if no such solution is possible

Valid Gerrymandering!

$m \cdot \frac{n}{2} \cdot \frac{1}{2}$

Dynamic Programming

- Requires Optimal Substructure
  – Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the “last thing” done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

Consider the last precinct

After assigning the first $n-1$ precincts $p_1, p_2, \dots, p_{n-1}$

D_1
k precincts
x voters for R

D_2 $(n-1)-k$
$n-k-1$ precincts
y voters for R

$n-1, k, x, y$

p_n

If we assign p_n to D_1

D_1
$k+1$ precincts
$x + R(p_n)$ voters for R

Valid gerrymandering if:
$k+1 = \frac{n}{2}$,
$x + R(p_n), y > \frac{mn}{4}$

If we assign p_n to D_2

D_2
$n-k$ precincts
$y + R(p_n)$ voters for R

Valid gerrymandering if:
$\mathrm{n} - k = \frac{n}{2}$,
$x, y + R(p_n) > \frac{mn}{4}$

D_1
$k+1$ precincts
$x + R(p_n)$ voters for R

D_2
$n-k-1$ precincts
y voters for R

World One

D_1
k precincts
x voters for R

D_2
$n-k$ precincts
$y + R(p_n)$ voters for R

World Two

Define Recursive Structure

$S(j, k, x, y) =$ True if from among the first $\boldsymbol{j}$ precincts:
$\boldsymbol{k}$ are assigned to D_1
exactly $\boldsymbol{x}$ vote for R in D_1
exactly $\boldsymbol{y}$ vote for R in D_2

$n \times n \times mn \times mn$

4D Dynamic Programming!!!

True here means that this is a valid state of the world; not a valid Gerrymander!

Two ways to satisfy $S(j,k,x,y)$:

$S(j,k,x,y) =$ True if:

from among the first j precincts

k are assigned to D_1

exactly x vote for R in D_1

exactly y vote for R in D_2

D_1
$k-1$ precincts
$x - R(p_j)$ voters for R

D_2
$j-k$ precincts
y voters for R

p_j

Then assign p_j to D_1

OR

D_1
k precincts
x voters for R

D_2
$j-1-k$ precincts
$y - R(p_j)$ voters for R

p_j

Then assign p_j to D_2

D_1
k precincts
x voters for R

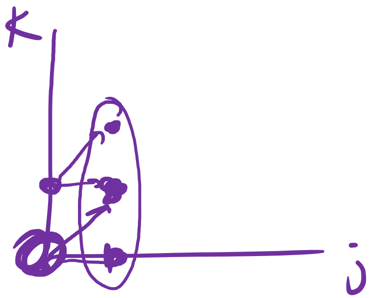
D_2
$j-k$ precincts
y voters for R

$$S(j,k,x,y) = S(j-1, k-1, x-R(p_j), y) \vee S\left(j-1, k, x, y-R(p_j)\right)$$

Final Algorithm

$$S(j,k,x,y) = S\big(j-1, k-1, x-R(p_j), y\big) \vee S\Big(j-1, k, x, y-R(p_j)\Big)$$

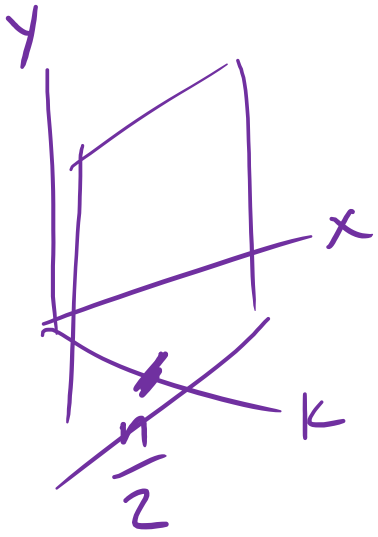
Initialize $S(0,0,0,0) = \text{True}$
for $j = 1, \ldots, n$:
  for $k = 1, \ldots, \min(j, \frac{n}{2})$:
    for $x = 0, \ldots, jm$:
      for $y = 0, \ldots, jm$:
        $S(j,k,x,y) =$
          $S\big(j-1, k-1, x-R(p_j), y\big) \vee S\Big(j-1, k, x, y-R(p_j)\Big)$

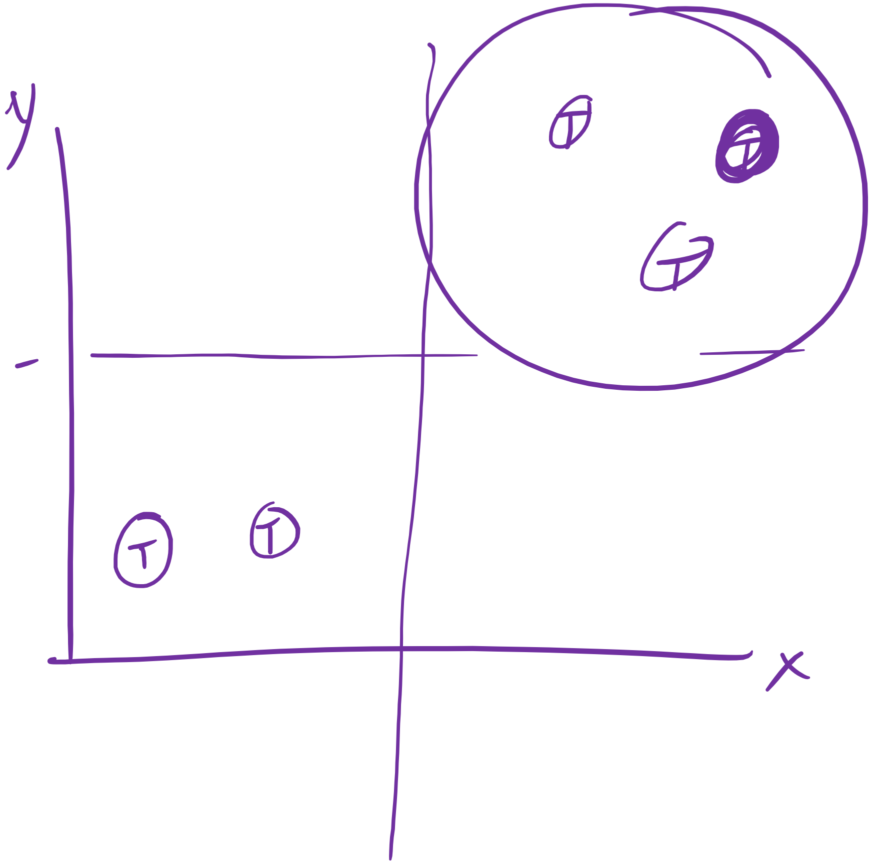


Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

$S(j,k,x,y) = \text{True}$ if:
- from among the first j precincts
- k are assigned to D_1
- exactly x vote for R in D_1
- exactly y vote for R in D_2

Where is Solution?





Run Time

$$S(j,k,x,y) = S(j-1,k-1,x-R(p_j),y) \vee S\left(j-1,k,x,y-R(p_j)\right)$$

Initialize $S(0,0,0,0) = \text{True}$

n for $j = 1, \ldots, n$:

$\frac{n}{2}$ for $k = 1, \ldots, \min(j, \frac{n}{2})$:

nm for $x = 0, \ldots, jm$:

nm for $y = 0, \ldots, jm$:

$S(j,k,x,y) =$

$S(j-1,k-1,x-R(p_j),y) \vee S\left(j-1,k,x,y-R(p_j)\right)$

Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

$\Theta(n^4 m^2)$

$\Theta(n^4 m^2)$

- Input: list of precincts (size n), number of voters (integer m)
- Runtime depends on the *value of* m, not *size of* m
  - Run time is exponential in *size* of input
  - Input size is $n + |m| = n + \log m$
- Note: Gerrymandering is NP-Complete