Dynamic programming

Class 6

Class 6

February 27th, 2018

Dynamic Programming

Preliminaries

Algo. techniques, so far

- recursion: elegantly consuming the instance
- on-line choice: probabilistic solution
- greedy: local optimization (sometimes)

Recursion may become ineffective

- we are required to find **an** optimal solution
- the specific measure is convex:

an optimal solution for I should contain an optimal sol. for $I' < I$.

- however, a recursive solution would require to solve sub-instances **repeteadly**

Chain matrix mult.

Assume textbook matrix multiplication

$A^1_{p_1 \times p_2} \circ A^2_{p_2 \times p_3}$ requires exactly $p_1 \cdot p_2 \cdot p_3$ scalar mult.

$$A^1_{p_1 \times p_2} \circ A^2_{p_2 \times p_3} \circ A^3_{p_3 \times p_4}$$

might require $p_1 \cdot p_2 \cdot p_3 + p_1 \cdot p_3 \cdot p_4$ mult.

or

$p_1 \cdot p_2 \cdot p_4 + p_2 \cdot p_3 \cdot p_4$ mult.

$$(A^1 \circ A^2) \circ A^3 = A^1 \circ (A^2 \circ A^3)$$

Optimal execution of CMM

Instance:

1. a sequence of matrices $A^1, A^2, \dots A^n$

s.t. each matrix product is defined:

$A^1_{p_1 \times p_2}, A^2_{p_2 \times p_3}, \dots A^n_{p_n \times p_{n+1}}$,

2. solution: a multiplication plan, e.g.,

$$((A_1 \cdot A_2) \cdot (A_3 \cdot A_4)) \cdot A_5$$

. . . or

$$((A_1 \cdot ((A_2 \cdot A_3) \cdot (A_4 \cdot A_5))$$

3. Measure: the overall no of scalar multiplications.

What is known?

A recursive a. would have to try all solutions:

$$MIN\{A^1 \circ (A^2..A^n),\ (A^1 \circ A^2) \circ (A^3..A^n), \dots\}$$

. . . with several repeated branches.

E.g., $(A^2 \circ A^3)$ is a subcase of $(A^1..A^3)$, $(A^2..A^4)$, $(A^1..A^4)$, $(A^2..A^5)$, ...

What is known? II

The number of recursive calls is proportional to the Catalan numbers series:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n} \frac{n+k}{k} \text{for } n \geq 0.$$

n	...	5	...	10	11	12
C_2	...	42	...	16,796	208,012	742,900

Dynamic programming

- allocate extra memory (size $\propto n$) to keep record of *intermediate results*
- work **bottom-up** to fill a table of intermediate results: each time we compute a new result by simply comparing existing results for **shortest subinstances**
- a Dynamic Programming algorithm fills up a solution table until the given instance is solved.
- no recursion, cost is normally $\propto n^2$. However, a table of $\propto n^2$ needs to be stored

Dynamic programming for CMM

Keep two tables:

1. Cost

$$i, j$$

   = min. cost for the chain $A^i..A^j$

2. Sol

$$i, j$$

   $= i \leq k < j$ tells where the outer paren. should fall:

$(A^i..A^k) \circ (A^{k+1}..A^j)$