Dynamic Programming-2

Department of Computer Science, Tsinghua University

Outline

- Challenge I: what are the subproblems in DP?
  - Take one choice of the optimal solution away!
  - Maximum subarray (Ch4.1), Knapsack
- Challenge II: establish recurrences
  - Optimal substructure (Ch15.3)
  - Knapsack, Matrix Chain Multiplication (Ch15.2)
- Challenge III: implementations
  - Bottom-up vs. Top-down
  - Matrix Chain Multiplication, Knapsack, All-pairs shortest paths (Ch 25.2)

DP Solution
Maximum Subarray

- Given an array A of n numbers, find the nonempty, contiguous subarray of A whose values have the largest sum.

① Make a choice to split the problem into one or more subproblems;

The maximum subarray contains $A[n]$ or not?

② Just assume you are given the choice that leads to an optimal solution S;

Yes / No

③ Given this choice, try to characterize the remaining subproblems? $Maxending[n]$ / $Max_{sub[n-1]}$

Define $Max\_sub[i]$ is an opt. solution for the subproblem of finding maximum subarray from $A[1..i]$; $Maxending[i]$ is an opt. solution for the subproblem of finding maximum subarray from $A[1..i]$;that ends at $A[i]$.

DP Solution Knapsack

- Given n items: $a_1, a_2, \dots, a_n$, and their weights: $w_1, w_2, \dots, w_n$, and values: $v_1, v_2, \dots, v_n$. A knapsack of capacity w. Find a subset of items of total weight $\leq w$ and of maximum value.

① Make a choice to split the problem into one or more subproblems;

The optimal subset contains a_n or not?

② Just assume you are given the choice that leads to an optimal solution S;

Yes/No

③ Given this choice, try to characterize the remaining subproblems? $KS_{n-1,w-w_n}$ / $KS_{n-1,w}$

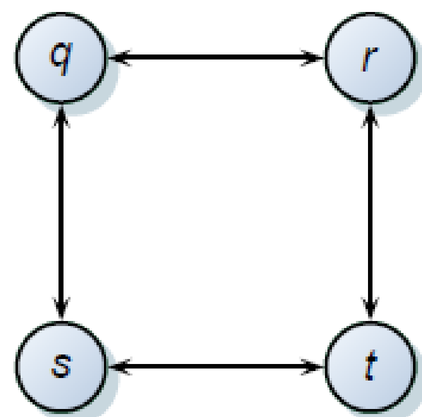
Define $KS_{i,u}$ is the subproblem of selecting from $a_1, \dots, a_i$ with a total weight $\leq u$

Optimal Substructure

- An optimal solution to a problem (instance) contains optimal solutions to subproblems.
  - In other words, we can build an optimal solution to the problem from optimal solutions to subproblems.
  - Is the foundation to build the recursive relationship
  - Reduces time complexity by safely ignoring non-optimal solutions to subproblems in the recursive relationship.
- Verify it by "***cut and paste***" technique!
  - We suppose that the subproblem solution is not optimal and then deriving a contradiction.
  - By "cutting out" the nonoptimal subproblem solution and "pasting in" the optimal one, you show you can get a better solution for the original problem, contradicting the fact that you already have an optimal solution for the original problem.

Optimal Substructure

- Characterizing the resulting space of subproblem.
  - Given an optimal choice, identify the resulting subproblems
- Not all problems exhibit optimal substructure property.
- For example, when subproblems are not independent, i.e., longest simple path problem:



Optimal Substructure in Knapsack

- How to discover optimal substructure?
  1. Make a choice to split the problem into one or more subproblems;

     Including item a_n or not?

  2. Just assume you are given the choice that leads to an optimal solution;

     The optimal solution contains a_n.

  3. Given this choice, try to best characterize the resulting space of subproblems;

     The subproblem becomes finding the opt. solution from $a_1, a_2, \dots, a_{n-1}$ with knapsack capacity of $w - w_n$.

  4. Prove optimal substructure by using a "cut-and-paste" technique.

     We need to prove if we take away a_n, then the remaining solution is opt. for the subproblem described in step 3.

Optimal Substructure in Knapsack

- Prove optimal substructure by using a "cut-and-paste" technique.
  - We need to prove if we take away a_n, then the remaining solution is opt. for the following subproblem: finding the opt. solution from $a_1, a_2, \ldots, a_{n-1}$ with knapsack capacity of $w - w_n$.

Optimal Substructure in Knapsack

- How to discover optimal substructure?
  1. Make a choice to split the problem into one or more subproblems;

  Including item a_n or not?

  2. Just assume you are given the choice that leads to an optimal solution;

  The optimal solution does not contain a_n.

  3. Given this choice, try to best characterize the resulting space of subproblems;

  The sub-problem becomes finding the opt. solution from $a_1, a_2, \dots, a_{n-1}$ with knapsack capacity of w.

  4. Prove optimal substructure by using a "cut-and-paste" technique.

  We need to prove the optimal solution is opt. for the subproblem described in step 3.

Recurrence of Knapsack

- Theorem (optimal substructure property): Define $z_{i,u}$ is an opt. profit for the subproblem of selecting from $a_1, \ldots, a_i$ with a total weight $\leq u$. Then,
  - If a_i belongs to the opt. solution, then $z_{i,u} = z_{i-1,u-w_i} + v_i$
  - If a_i does not belong to the opt. solution, then $z_{i,u} = z_{i-1,u}$
  So we compare both cases, and take the larger one.
- Recursive solution:
- $z_{1,u} = \begin{cases} 0 & if\ u < w_1 \\ v_1 & otherwise \end{cases}$
- $z_{i,u} = \begin{cases} 0 & if\ u \leq 0 \\ z_{i-1,u} & if\ u < wi \\ \max\left(z_{i-1,u}, z_{i-1,u-w_i} + v_i\right) & otherwise \end{cases}$

Recurrence of Knapsack

Example: Knapsack of capacity $W = 5$

item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20
4	2	\$15

$$z_{i,u} = \begin{cases} 0 & if\ u \le 0 \\ z_{i-1,u} & if\ u < w_i \\ \max\left(z_{i-1,u}, z_{i-1,u-w_i} + v_i\right) & otherwise \end{cases}$$

capacity u

	i	0	1	2	3	4	5
$w_1 = 2, v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1, v_2 = 10$	2	0					
$w_3 = 3, v_3 = 20$	3						
$w_4 = 2, v_4 = 15$	4						

Matrix Chain Multiplication

- Problem:
  - Given a sequence $\langle A_1, A_2, \dots, A_n \rangle$ of n matrices to be multiplied, and we wish to compute the product: $A_1 A_2 \dots A_n$, we want to determine the order of this multiplication such that the total cost is minimized.
  - Matrix multiplication is associative, so all orders yield the same product.
  - Cost defined by the number of scalar multiplications.
- For example:
  - $A_{10\times 100} B_{100\times 5} C_{5\times 50}$
  - $A_{10\times 100}(B_{100\times 5} C_{5\times 50})$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$
  - $(A_{10\times 100} B_{100\times 5}) C_{5\times 50}$: $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

Matrix Chain Multiplication

- Use parentheses to represent order.
- Fully parenthesized
  - A product of matrices is **fully parenthesized** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses.
- Example
  - $A_1A_2A_3A_4$
  - $(A_1(A_2(A_3A_4)))$, $(A_1((A_2A_3)A_4))$, $((A_1A_2)(A_3A_4))$, $((A_1(A_2A_3))A_4)$, $(((A_1A_2)A_3)A_4)$

Solution Space

- Counting the number of ways of parenthesization
  - Denote the number of alternative ways of parenthesization for a sequence of n matrices by $P(n)$.
- The first split can happen between the kth and $k+1$th matrices ($k = 1,2,\cdots,n-1$), then parenthesize the two resulting subsequences independently.
- $P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases}$
- In-class exercise: $P(5) = ?$
- $P(n)$ is the sequence of Catalan number
- $P(n) = \Omega(4^n / n^{3/2})$

Optimization Problems

- Matrix Chain Multiplication:
  - Instance: a possible input, e.g. a matric chain.
  - Solutions for an instance: Each instance has an exponentially large set of feasible solutions (space of solutions), e.g. orders
  - Cost of Solution: Each solution has an easy-to-compute cost or value, e.g. the number of scalar multiplications.
  - Optimal Solution: determine the order of this multiplication such that the total cost is minimized.

Steps of DP

- Steps of dynamic programming
  - Step1: Characterize the structure of an optimal solution

Step1: the structure of an optimal solution

- $A_1A_2A_3 |\ A_4A_5$
- $A_1A_2A_3$ has two alternative orders $(A_1A_2)A_3$ and $A_1(A_2A_3)$
- Brute force will check both $\big((A_1A_2)A_3\big)(A_4A_5)$, $\big((A_1A_2)A_3\big)(A_4A_5)$
- DP will only check one of them (by optimal substructure).

- In stead of all combinations, DP only checks the combination of optimal solutions to subproblems.

Step1: the structure of an optimal solution

① Make a choice to split the problem into one or more subproblems;

Split the matrix chain

② Just assume you are given the choice that leads to an optimal solution S;

Split the matrix chain between A_k and A_{k+1} in S

③ Given this choice, try to best characterize the resulting space of subproblems;

Two remaining subproblems: subchains $A_1 A_2 \dots A_k$ and $A_{k+1} A_{k+2} \dots A_n$

Step1: the structure of an optimal solution

④ Prove optimal substructure by using a "cut-and-paste" technique.

The order of multiplications of $A_1A_2 \dots A_k$ in S must be an optimal order for multiplying $A_1A_2 \dots A_k$;The order of $A_{k+1}A_{k+2} \dots A_n$ in S must be an optimal order for $A_{k+1}A_{k+2} \dots A_n$.

e.g. if $((A_1A_2)A_3)A_4$ is the optimal solution, $(A_1A_2)A_3$ must be optimal for $A_1A_2A_3$.

Steps of DP

- Steps of dynamic programming
  - Step1: Characterize the structure of an optimal solution
  - Step2: Recursively define the value of an optimal solution

Step2: Recurrence

- Let $m[i,j]$ be the minimum number of scalar multiplications needed to computer the matrix chain $A_{i..j}$
- **Setup a recurrence for** $m[i,j]$, then the original problem: a cheapest way would thus be $m[1,n]$

- If the optimal solution of $A_{i..j}$ cuts at k
  - $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
- Try all possible k and choose the minimum
  - $m[i,j] = \begin{cases} 0 & if\ i = j \\ \min\limits_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & if\ i < j \end{cases}$