

## Dynamic Programming: Knapsack Problem and Sequence Alignment

CMSC 451, Summer 2009

### Knapsack Problem

**Knapsack problem.**

- Given  $n$  objects and a "knapsack."
- Item  $i$  weighs  $w_i > 0$  kilograms and has value  $v_i > 0$ .
- Knapsack has capacity of  $W$  kilograms.
- Goal: fill knapsack so as to maximize total value.

**Ex:** { 3, 4 } has value 40.

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$W = 11$

**Greedy:** repeatedly add item with maximum ratio  $v_i / w_i$ .

**Ex:** { 5, 2, 1 } achieves only value = 35  $\Rightarrow$  greedy not optimal.

### Knapsack Problem: Example

```

Input: n, W, w1, ..., wn, v1, ..., vn
for w = 0 to W
  M[0, w] = 0
for i = 1 to n
  for w = 1 to W
    if (wi > w)
      M[i, w] = M[i-1, w]
    else
      M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi]}
return M[n, W]
    
```

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$W = 11$

### Knapsack Algorithm

		W + 1											
		0	1	2	3	4	5	6	7	8	9	10	11
$n+1$	$\phi$	0	0	0	0	0	0	0	0	0	0	0	0
	{1}	0	1	1	1	1	1	1	1	1	1	1	1
	{1, 2}	0	1	6	7	7	7	7	7	7	7	7	7
	{1, 2, 3}	0	1	6	7	7	18	19	24	25	25	25	25
	{1, 2, 3, 4}	0	1	6	7	7	18	22	24	28	29	29	40
	{1, 2, 3, 4, 5}	0	1	6	7	7	18	22	28	29	34	34	40

**OPT:** { 4, 3 }  
value = 22 + 18 = 40

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$W = 11$

### Knapsack Problem: Proof of correctness

**Proof of correctness.**

- The algorithm examines the entire solution space and only the solution space by construction
  - All possibilities are considered since we consider all paths associated with choosing or not choosing to pack each object
  - The chosen solution is feasible since any item which puts us over the weight limit is not included
- The algorithm chooses the maximum value over the solution space by construction, since the recurrence returns the max of its two recursive calls

### Knapsack Problem: Running Time

**Running time.**  $\Theta(nW)$ .

It takes  $O(1)$  time to fill in each entry and there are  $nW$  such entries that are each filled in exactly once. Again, our progress measure is the number of nonempty entries, and this begins at 1 and increases by 1 until there are  $nW$  nonempty entries.

- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]  
(Can a value of at least  $v$  be achieved without exceeding  $W$ ?)

**Knapsack approximation algorithm.** There exists a poly-time algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]

## 6.6 Sequence Alignment

### String Similarity

How similar are two strings?

- occurrence
- occurrence

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

0 mismatches, 3 gaps

### Edit Distance

**Applications.**

- Basis for Unix diff.
- Speech recognition.
- Computational biology.

**Edit distance.** [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty  $\delta$ ; mismatch penalty  $\alpha_{pq}$ .
- Cost = sum of gap and mismatch penalties.

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

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

$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$

$2\delta + \alpha_{CA}$

### Sequence Alignment

**Goal:** Given two strings  $X = x_1 x_2 \dots x_m$  and  $Y = y_1 y_2 \dots y_n$  find alignment of minimum cost.

**Def.** An alignment  $M$  is a set of ordered pairs  $x_i-y_j$  such that each item occurs in at most one pair and no crossings.

**Def.** The pair  $x_i-y_j$  and  $x_{i'}-y_{j'}$  **cross** if  $i < i'$ , but  $j > j'$ .

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i: x_i \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta}_{\text{gap}}$$

**Ex:** CTACCG VS. TACATG.

**Sol:**  $M = \{x_2-y_1, x_3-y_2, x_4-y_3, x_5-y_4, x_6-y_5\}$ .

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
	C	T	A	C	C	G

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	
	-	T	A	C	A	T	G

### Sequence Alignment: Problem Structure

**Def.**  $OPT(i, j)$  = min cost of aligning strings  $x_1 x_2 \dots x_i$  and  $y_1 y_2 \dots y_j$ .

- Case 1:  $OPT$  matches  $x_i-y_j$ .
  - pay possible mismatch for  $x_i-y_j$  + min cost of aligning two strings  $x_1 x_2 \dots x_{i-1}$  and  $y_1 y_2 \dots y_{j-1}$
- Case 2a:  $OPT$  leaves  $x_i$  unmatched.
  - pay gap for  $x_i$  and min cost of aligning  $x_1 x_2 \dots x_{i-1}$  and  $y_1 y_2 \dots y_j$
- Case 2b:  $OPT$  leaves  $y_j$  unmatched.
  - pay gap for  $y_j$  and min cost of aligning  $x_1 x_2 \dots x_i$  and  $y_1 y_2 \dots y_{j-1}$

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i=0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) & \text{otherwise} \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{if } i > 0 \\ i\delta & \text{if } j=0 \end{cases}$$

### Sequence Alignment: Algorithm

```

Sequence-Alignment(m, n, x1x2...xm, y1y2...yn,  $\delta$ ,  $\alpha$ ) {
  for i = 0 to m
    M[0, i] = i $\delta$ 
  for j = 0 to n
    M[j, 0] = j $\delta$ 

  for i = 1 to m
    for j = 1 to n
      M[i, j] = min( $\alpha[x_i, y_j]$  + M[i-1, j-1],
                     $\delta$  + M[i-1, j],
                     $\delta$  + M[i, j-1])

  return M[m, n]
}
    
```

## Sequence Alignment: Example

```

Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
  for i = 0 to m
    M[0, i] = iδ
  for j = 0 to n
    M[j, 0] = jδ

  for i = 1 to m
    for j = 1 to n
      M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                  δ + M[i-1, j],
                  δ + M[i, j-1])
  return M[m, n]
}

```

X = CCGT      m=4    Y = CGTA      n=4  
 δ = 2    α<sub>AA</sub>=0   α<sub>AC</sub>=4   α<sub>AG</sub>=4   α<sub>AT</sub>=1   α<sub>CC</sub>=0   α<sub>CG</sub>=1   α<sub>CT</sub>=2  
          α<sub>GG</sub>=0   α<sub>GT</sub>=2   α<sub>TT</sub>=0

13

## Sequence Alignment: Analysis

```

Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
  for i = 0 to m
    M[0, i] = iδ
  for j = 0 to n
    M[j, 0] = jδ

  for i = 1 to m
    for j = 1 to n
      M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                  δ + M[i-1, j],
                  δ + M[i, j-1])
  return M[m, n]
}

```

**Analysis.**  $\Theta(mn)$  time and space.

English words or sentences:  $m, n \leq 10$ .

Computational biology:  $m = n = 100,000$ . 10 billions ops OK, but 10GB array?

14