DYNAMIC PROGRAMMING

View a problem as a set of interdependent problems. Solve subproblems to solve problem. Soln to subproblem $\equiv$ fn of solns to subproblems at preceding lvls.

Shortest path problem

$f(x) \equiv$ cost of shortest path from node ϕ to node x

$$f(x) = \begin{cases} 0 & x = 0 \\ \min\limits_{0 \leq j < x} \{f(j) + c(j,x)\} & 1 \leq x \leq (n-1) \end{cases}$$

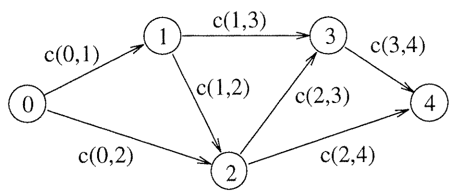


Figure 9.1 A graph for which the shortest path between nodes 0 and 4 is to be computed.

$$f(4) = \min\{f(3) + c(3,4),\ f(2) + c(2,4)\}$$

$$= \cancel{\min}\left\{ \left[\min\left\{ \begin{matrix} f(1) + c(1,3), \\ f(2) + c(2,3) \end{matrix} \right\}\right] + c(3,4),\ \left[\min\{f(1) + c(1,2),\ c(0,2)\}\right] + c(2,4) \right\}$$

- recursive eqn, LHS is unknown qty, RHS is min (max) expr.
  $\equiv$ functional eqn or optimization eqn
- functional eqn : cost fn has
  - single recursive term : <u>monadic</u>
  - multiple recursive terms : <u>polyadic</u>
- serial DP formulation : subproblems (at all lvls) depend only on results of immediately preceding levels
  non serial DP formulation : otherwise

	monadic	polyadic
serial	shortest path in a graph with levels 0/1 knapsack	Floyd's all-pairs shortest paths
nonserial	longest common subsequence	optimal matrix-parenthesization

- some DP problems cannot be classified in above categories

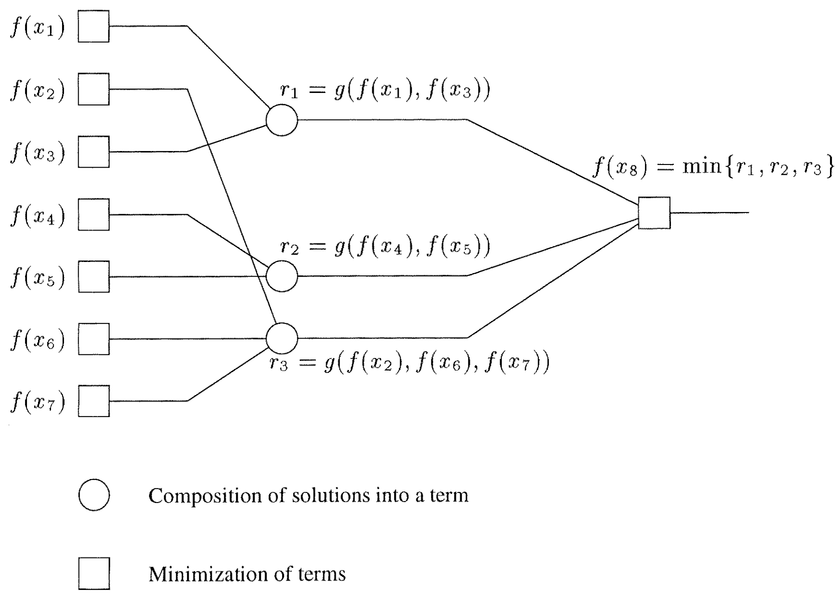


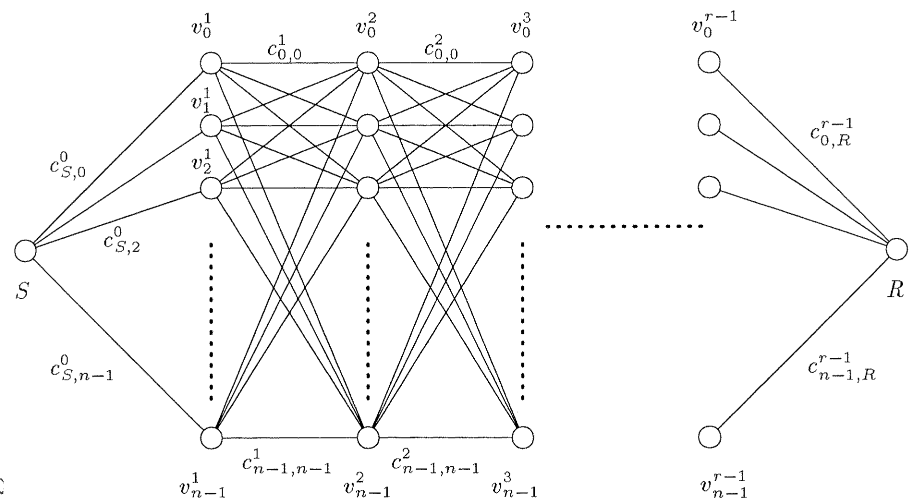
Figure 9.2 The computation and composition of subproblem solutions to solve problem $f(x_8)$.

Shortest path in a graph

- v_i^l : i^{th} node at level l
- $c_{i,j}^l$: cost of edge from v_i^l to v_j^{l+1}
- C_i^l : cost of reaching goal node R from v_i^l
- $\mathcal{C}^l$: vector $[C_0^l, C_1^l, C_2^l, \text{------} C_{n-1}^l]^T$, where there are n nodes at level l

$$C_i^l = \min\{(c_{i,j}^l + C_j^{l+1}) \mid j \text{ is a node at level } l+1\} \quad \text{——— (1)}$$

$$\mathcal{C}^{r-1} = [c_{0,R}^{r-1}, c_{1,R}^{r-1}, c_{2,R}^{r-1}, \text{------} c_{n-1,R}^{r-1}] \quad \text{——— (2)}$$



Modified matrix multiplication Reformulation

Figure 9.3 An example of a serial monadic DP formulation for finding the shortest path in a graph whose nodes can be organized into levels. [weighted multistage graph]
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

$\mathcal{C}^l = M_{l,l+1} \times \mathcal{C}^{l+1}$, where $\mathcal{C}^l$ & $\mathcal{C}^{l+1}$ are $n \times 1$ vectors, &

$M_{l,l+1}$ is an $n \times n$ matrix in which entry (i,j) stores the cost of edge (node i @ level l) to (node j @ level $(l+1)$)

$$M_{l,l+1} = \begin{bmatrix} c_{0,0}^l & c_{0,1}^l & c_{0,2}^l & \cdots & c_{0,n-1}^l \\ c_{1,0}^l & c_{1,1}^l & c_{1,2}^l & \cdots & c_{1,n-1}^l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{n-1,0}^l & c_{n-1,1}^l & c_{n-1,2}^l & \cdots & c_{n-1,n-1}^l \end{bmatrix}$$

- Sequential : Use $\mathcal{C}^{r-1}$, then compute $\mathcal{C}^{r-k-1}$, $k=1,2,\cdots,r-2$
  cost of computing each $\mathcal{C}^l$ is $\Theta(n^2)$
- Parallel : refer matrix multiplication chapter

0/1 Knapsack : $\bar{v}$ is solution vector

satisfy $\sum_{i=1}^{n} w_i v_i \leq c$, while maximizing profit $\sum_{i=1}^{n} p_i v_i$

- $F[i,x]$ = max profit for knapsack of capacity x using only objects $\{1, 2, \cdots i\}$
  Find $F[n, c]$
- $$F[i,x] = \begin{cases} 0 & i = 0, x \geq 0 \\ -\infty & i = 0, x < 0 \\ \max_{1 \leq i \leq n} \{F[i-1, x], (F[i-1, x-w_i] + p_i)\} & 1 \leq i \leq n \end{cases}$$

- sequential : table F of size $n \times c$; row-major construction ; $\Theta(nc)$
- CREW PRAM : c processors ; P_{r-1} computes r^{th} column ; parallel runtime $\Theta(n)$
  $p \times T_p = \Theta(nc)$ $\Rightarrow$ cost optimality

Table F

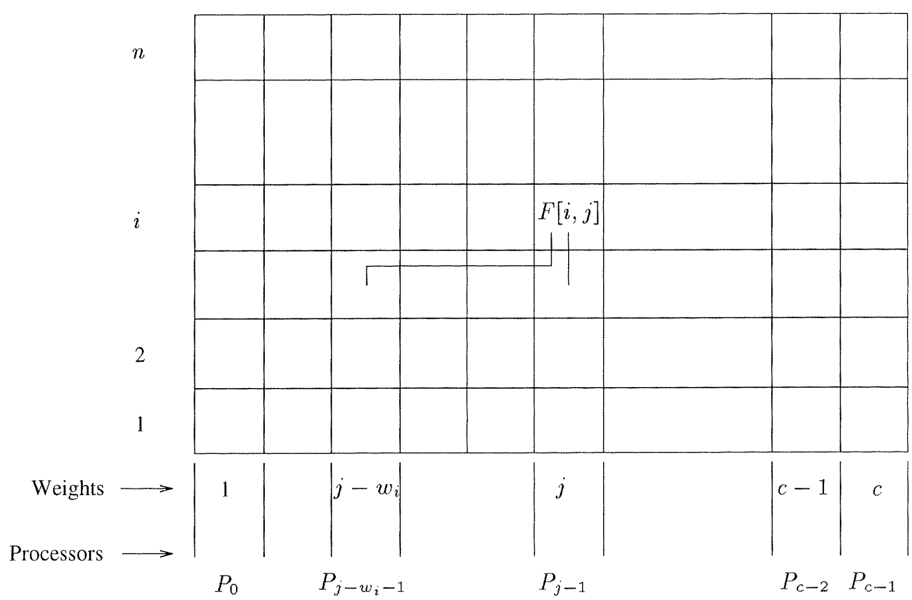


Figure 9.4 Computing entries of table F for the 0/1 knapsack problem. The computation of entry $F[i,j]$ requires communication with processors containing entries $F[i-1,j]$ and $F[i-1, j-w_i]$. Serial monadic.

- c-processor HC : each P_i responsible for column$_i$ & locally has $\bar{p}$ & $\bar{w}$
  - → j^{th} iteration : to compute $F[j, r]$, need to fetch $F[j-1, r-w_j]$, using circular w_j-shift $\equiv O(t_s + t_w + t_h \log c)$
  - → n iterations : $O((t_c + t_s + t_w + t_h \log c) n) = O(n \log c)$
  - → $pT_p = O(nc \log c)$ $\Rightarrow$ not cost optimal
- p-processor HC : each processor computes c/p elements of table
  - → parallel run-time: $n(t_c c/p + 2t_s + t_w c/p + 2t_h \log p) = O(\frac{nc}{p} + n \log p)$
  - → $pT_p = O(nc + np \log p)$ asymptotically
  - → cost-optimal if $c = \Omega(p \log p)$

LONGEST COMMON SUBSEQUENCE [nonserial Monadic DP]

- subsequence of $A = \langle a_1, a_2, \cdots a_n \rangle$ is formed by deleting some entries from A
- given seqs A & B, find the longest seq that is a subsequence of A & B
- eg $A = \langle c, a, d, b, r, z \rangle$, $B = \langle a, s, b, z \rangle$, lcs $= \langle a, b, z \rangle$
- $F[i,j] \equiv$ length of l.c.s of the $\begin{cases} \text{first } i \text{ elements of A and} \\ \text{first } j \text{ elements of B} \end{cases}$
- $$F[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ F[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max\{F[i,j-1], F[i-1,j]\} & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

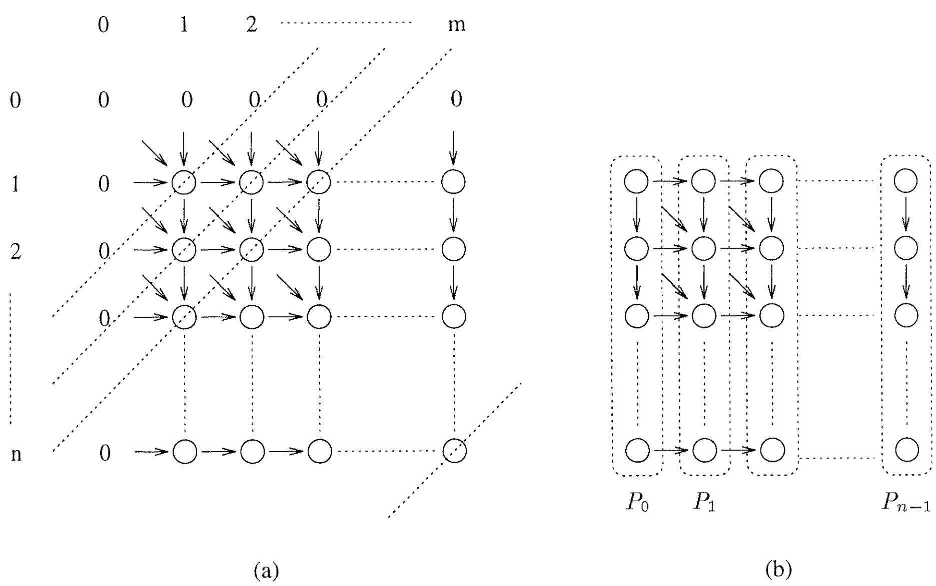


Figure 9.5 (a) Computing entries of table F for the longest-common-subsequence problem. Computation proceeds along the dotted diagonal lines. (b) Mapping elements of the table to processors.

- sequential : calculate table in row-major order $\Rightarrow \Theta(nm)$
- CREW PRAM : n procs, $n = m$, P_i computes i^{th} column of table
  - → diagonal sweep ≡ $2n-1$ diagonals $\Rightarrow \Theta(n)$ iterations
  - → runtime $\Theta(n) \Rightarrow$ cost optimal
- linear array of n processors : P_i stores $(i+1)^{th}$ column
  - → for $F[i,j]$, P_{j-1} may need $F[i-1,j-1]$ or $F[i,j-1]$ from LHS processor
  - → to communicate 1 word $\Rightarrow t_s + t_w$
  - → $T_p = (2n-1)(t_s+t_w+t_c)$; $E = \dfrac{n^2 t_c}{n(2n-1)(t_s+t_w+t_c)}$; if $t_s = t_w = 0$, $E = \dfrac{1}{2-1/n}$

upper bound

FLOYD's all-pairs Shortest Paths [Serial polyadic]

- $d^k_{i,j}$ = min cost of path from i to j, using only nodes $v_0, v_1, \dots v_{k-1}$

$$d^k_{i,j} = \begin{cases} c_{i,j} & k = 0 \\ \min\left\{ d^{k-1}_{i,j},\ \left(d^{k-1}_{i,k} + d^{k-1}_{k,j}\right)\right\} & 0 \leq k \leq n-1 \end{cases}$$

- polyadic because solns to $d^k_{i,j}$ require composition of solutions to two subproblems $d^{k-1}_{i,k}$ and $d^{k-1}_{k,j}$

- sequential : $\Theta(n^3)$

- CREW PRAM : n^2 processors in a 2-D array
  - $\rightarrow$ $P_{i,j}$ computes $d^k_{i,j}$; k iterations
  - $\rightarrow$ run time $\Theta(n)$
  - $\rightarrow$ $pT_p = \Theta(n^3) \Rightarrow$ cost-optimal