IO543: Scheduling

Lecture 6

Brian Denton
Associate Professor, IOE

IOE Bld 2893
btdenton@umich.edu

Today

Goals:

- Introduction to dynamic programming
- Learn to solve:
  - $1|prec\ |L_{max}$
  - $1|\ |\sum T_j$

Maximum Lateness: 1 | | $L_{\max}$

<u>Definition:</u> **Lateness** is the difference between the completion time and the due date.

- The **maximum lateness** is:

$$L_{max} = \max_{i \in \{1,\ldots,n\}} L_i \text{ where } L_i = C_i - d_i$$

$1\ |prec|h_{max}$

Consider the following **generalization** of 1 || $L_{\max}$:

$$1\ |prec|h_{max}$$

Where $h_{max} = \max\{h_1(C_1), h_2(C_2), \dots, h_n(C_n)\}$ and $h_j(\cdot), j = 1, \dots, n$, are **any** nondecreasing cost functions.

This problem can be solved by **<u>dynamic programming</u>**.

Introduction to DP

Dynamic programming (DP) dates back to early work of **Richard Bellman** in the 1940's

Since it's development DP has been applied to fields of engineering, biology, chemistry, and many others

DP is a powerful technique for scheduling problems

For more history on Richard Bellman see: http://www.gap-system.org/~history/Biographies/Bellman.html

Dynamic Programming (DP) Intro

The elements of a DP are:

- Stages: t
- States: $\mathrm{s_t}$
- Actions: a_t
- Rewards: $r_t(s_t, a_t)$

Goal: Find the **policy** that maximizes total rewards

Dynamic Programming (DP) Primer

DPs are solved via **optimality equations**:

$$v_t(s_t) = \max_{a_t \in A}\{r_t(s_t, a_t) + v_{t+1}(s_{t+1})\}, \qquad for\ all\ s_t$$

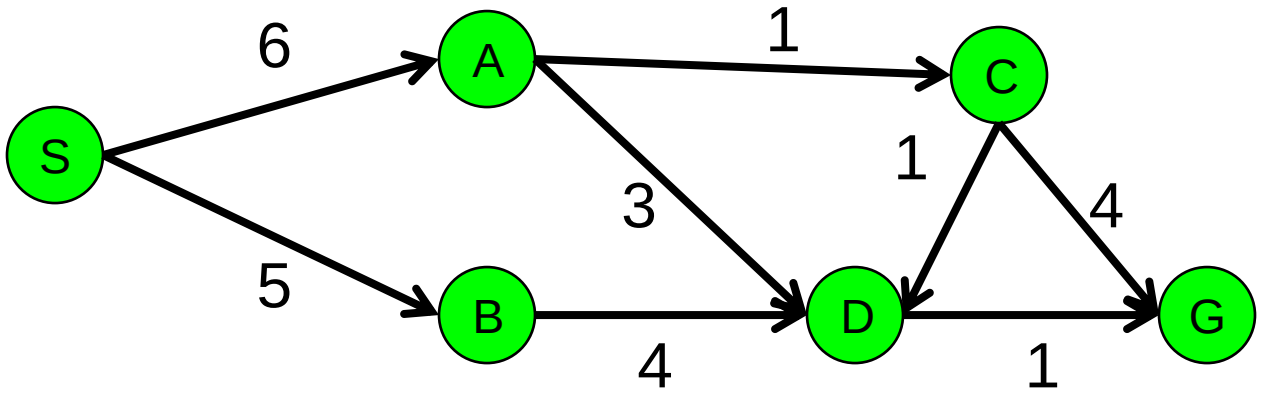
$$\mathrm{v}_T(s_T) = R(s_T), \qquad for\ all\ s_T$$

$v_t(s_t)$ is the **value to go** for stages t, t+1,…,T

$R(s_T)$ is the **terminal reward** at the last stage

Example: Shortest Path

What is the shortest path in this directed graph?



Example: DP Formulation

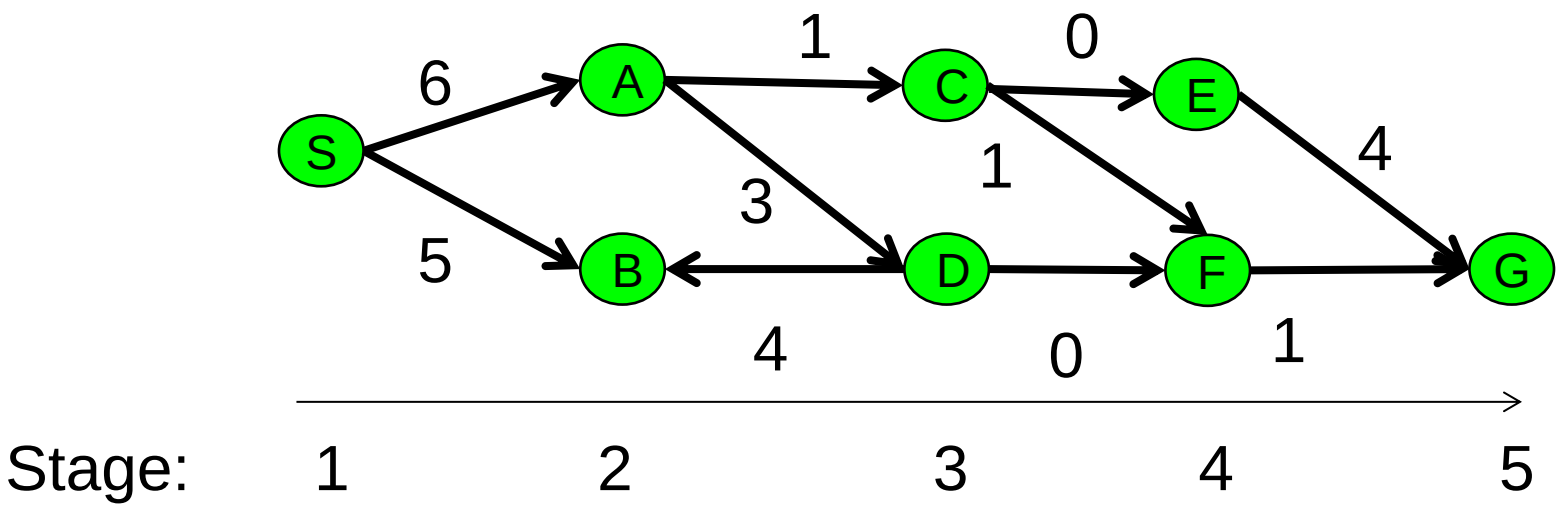
Following are the elements of the DP formulation

- States: vertices of the graph
- Actions: which vertex to move to?
- Rewards: distance associated with each edge

Goal: Starting from vertex S, select the action at each vertex that will minimize total edge distance travelled to reach vertex G

Example: Shortest Path Problem

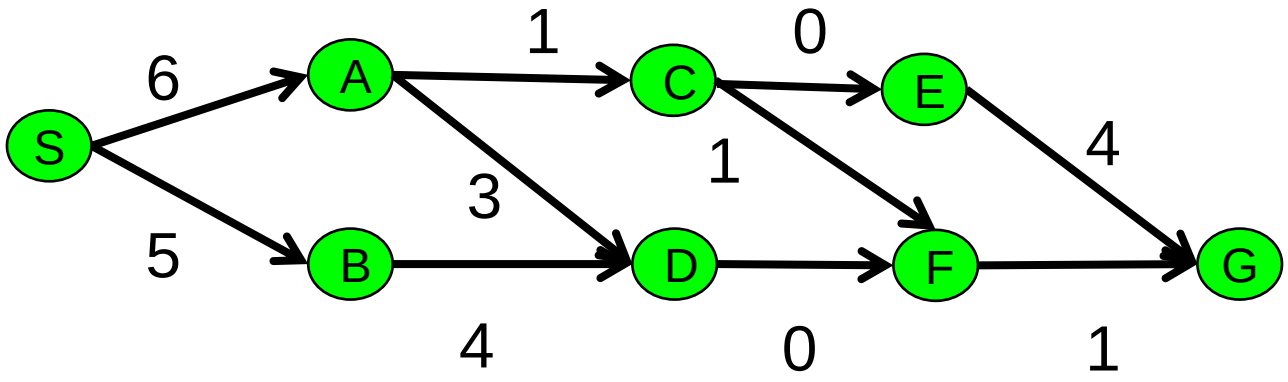
- Modify the graph to create a one-to-one correspondence between vertices (states) and stages



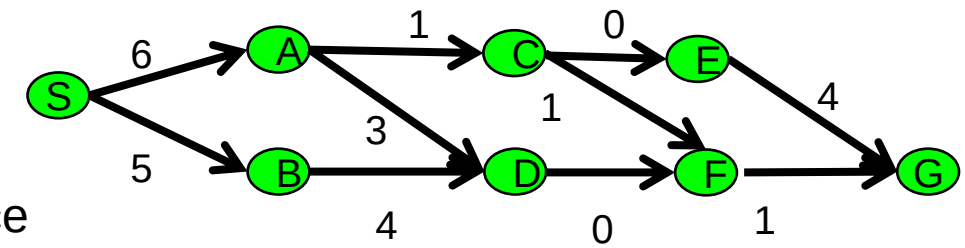
Example: Shortest Path Problem

Principle of Optimality:

- Any “subpath” in the optimal path must be the shortest path between the two vertices
- To find the best action at node S compare
  - Distance from S to A + Shortest Path from A to G
  - Distance from S to B + Shortest Path from B to G



Example: DP Formulation



- Initial condition: $v_5(G) = 4$,
- Start by finding the **Stage 4** distance

  from E→ G and F→ G: $v_4(E) = 4$, $v_4(F) = 1$
- **Stage 3**: compute the shortest path from C and D:

  $v_3(C) = \min\{0+V_4(E), 1+V_4(F)\}=2$, $v_3(D) = 0+ v_4(F) = 1$
- **Stage 2**: compute the shortest path from A and B:

  $v_2(A) = \min\{1+v_3(C), 3+v_3(D)\} = 3$, $v_2(B) = 4+v_3(D) = 5$
- **Stage 1**: compute the shortest path from S:

  $v_1(S) = \min\{6+v_2(A), 5+v_2(B)\} = 9$

<u>Answer:</u> Shortest path is S→A→C→F→G with a path length of 9

Learn More About DP

DP will be appear several times in this course. Following are good references:

- Operations Research, 4th Edition, Wayne Winston, Duxbury Press, 2003 — *Beginners*
- The Art and Theory of Dynamic Programming, Stuart Dreyfus and Averill Law, 1977 (on Ctools) — *I'm feelin' lucky*
- Dynamic Programming and Optimal Control, Volumes 1 and 2, Dimitri Bertsekas, Athena Scientific — *Bring it on!*

Applying DP to $1\ |prec|h_{max}$

DP works **backwards** through the problem

At a given stage, *J* is the set of jobs already scheduled, in the interval

$$[C_{max} - \sum_{j \in J} p_j\ , C_{max}]$$

J^c is the set of jobs <u>remaining</u> to be scheduled and J' is the set of <u>schedulable</u> jobs

DP Algorithm for $1|prec|h_{max}$

Empty set

Algorithm (3.2.1 Pinedo):

Step 1: Set $J = \emptyset, J^c = \{1, .., n\}$, and let J' be all jobs that have no successors.

Step 2: Determine $j^* = argmin_{j \in J'}\{h_j(\sum_{k \in J^c} p_k)\}$

Add $j*$ to J and delete $j*$ from J^c. Modify J' to represent the new set of schedulable jobs.

Step 3: If $J^c = \emptyset$ then STOP, otherwise go to Step 2

In-Class Assignment

Use Algorithm 3.2.1 to determine the schedule that minimizes h_{max} for the following example:

jobs	1	2	3
p_j	2	3	5
$h_j(C_j)$	$1+C_1$	$1.2\,C_2$	$2+0.8\,C_3$

A Special Case: 1 || L_{max}

Algorithm 3.2.1 can be used to prove the earliest due date first (EDD) rule is optimal for 1 || L_{max}

Proof sketch by **induction** using DP formulation:

- At stage n it is optimal to choose the job with maximum due date L_{max}
- Assume it is optimal to choose maximum due date job at stages n,n-1,…,k +1
- Prove it is optimal to choose maximum due date at stage k

Total tardiness $1||\Sigma T_j$

- Due to its practical importance $1||\Sigma T_j$ it has been used in many applications
- Until 1990 it was not clear if it was easy or (NP-) hard
- It can be solved using **dynamic programming** or **integer programming**

An Important Result

Arrange all jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$ and let job k be such that $p_k = \max(p_1, \ldots, p_n)$

<u>Lemma (3.4.1 Pinedo):</u> There is an integer δ, $0 \leq \delta \leq n-k$, such that there is an optimal sequence S in which job k is **preceded** by all jobs j with $j \leq k+\delta$ and **followed** by all jobs j with $j > k+\delta$.

Proof: See Pinedo

We can use this to decompose the scheduling problem into 3 parts

Consequences of Lemma 3.4.3.

There is an optimal sequence that processes the jobs in this order:

1. jobs 1,2,…,k-1,k+1,…,k+δ in some order
2. job k
3. jobs k+δ+1, k+δ+2, …, k+n in some order

How do we find the optimal sequence?

DP to the rescue!

Algorithm to Min Total Tardiness

Definitions:

- Completion time of job k: $C_k(\delta)=\Sigma_{j\leq k+\delta}p_j$

- J(j,l,k) is the **set of jobs** in {j,j+1,…,l} that have a processing time less than job k

- k' is the job in J(j,l,k) with the next largest processing time, i.e., $p_{k'}=\max\{p_j : j \in J(j,l,k)\}$

- V(J(j,l,k),t) is the **minimum total tardiness** of the jobs in J(j,l,k) for the optimal sequence starting at time t

DP Algorithm for $1||\Sigma\, T_j$

Iteratively apply the following recursive equations:

Optimality Equations:

$$V(J(j,l,k),t) = \min_{\delta}\{V(J(j,k'+\delta,k'),t)$$

$$+\max\{0, C_{k'}(\delta) - d_{k'}\} + V(J(k'+\delta+1,l,k'), C_{k'}(\delta))\}$$

Initial conditions:

$$V(\emptyset,t) = 0, \qquad V(\{j\},t) = \max\{0, t + p_j - d_j\}$$

Example

Use the DP Algorithm to solve the following instance of $1||\Sigma T_j$

jobs	1	2	3	4	5
p_j	121	79	147	83	130
d_j	260	266	266	336	337

Check ctools for complete solution

Complexity of DP Algorithm

- There are at most n^3 subsets J(j,l,k) and at most Σp_j time points

- Each equation requires O(n) operations so the overall complexity is $O(n^4 \Sigma p_j)$

- The running time is polynomial in n, however due to Σp_j the algorithm is called **pseudopolynomial** or NP Hard in the **ordinary sense**

Next Time

We will wrap up chapter 3: read sections 3.6 and 3.7

Next Thursday we will begin on Chapter 4