Optimization Theory
MT 610

2011/12 Semester I

Dynamic Programming

©2012 St. John's University of Tanzania

Dynamic Programming

- An approach to making sequential, interrelated decisions in an optimal way
  - *Multistage Decision Problems / Sequential Decision Problems*
- Method is *recursive*
  - Adding information to a stack at each step
  - Stopping when certain conditions are met
  - Removing the information in the proper sequence
- Optimize part of the problem, then use that solution to optimize a slightly larger problem. Keep increasing the size of the problem until it encompasses the original problem. (Ex: *Dykstra's shortest path algorithm*)

©2012 St. John's University of Tanzania

Characteristics

- *Stages*
- *States* at each Stage
- *Decision* at each Stage
  - Decision updates the State for the next Stage
  - Optimum decision for remaining Stages is *independent* of decisions at previous Stages
- *Recursive relationship* between value of decision at current Stage and the value of optimum decisions at earlier stages
- Often stages are sequenced in time, hence the name dynamic programming. Optimizing the answer to the “What next?” question

©2012 St. John's University of Tanzania

Recursion

- Shortest path example
  - Shortest path to node i =
    minimum { Shortest path to solved nodes j +
    Distance from j to I}
  - Shortest path on both sides
- New optimum derived from old optimum along with some local value
- Recursive relation can be addition, multiplication or even something more abstract and general

©2012 St. John's University of Tanzania

Formulating the Solution

- What are the *stages* in the solution?
- How is the *state* defined at a stage?
- What kind of *decision* must you make at a stage?
- How does the decision *update* the state for the next stage?
- What is the *recursive* value relationship between the optimum decision at a stage and a previous optimum decision?

©2012 St. John's University of Tanzania

More on Recursion

- Dynamic Programming most often involves *backward* recursion
  - Consider this starting at the last step in a decision process and working back to the initial decision
- Why backward?
  - Sometimes *must* be done that way
    - Employee scheduling: need a certain number at the last stage, so don't need to evaluate paths that won't get there

©2012 St. John's University of Tanzania

Example 1

- Equipment Replacement (Chinneck, 2010, Ch15, p3)
  - Objective function
    - cost of ownership = acquisition + maintenance – scrap value
  - Stages = time frames, overall & incremental
  - Decision = buy or keep at each stage
  - End stage = must have a functioning piece of equipment at the end of the overall time frame
- Specific case: Bicycle over five years
- Recursion is additive

©2012 St. John's University of Tanzania

Example 1 as Shortest Path

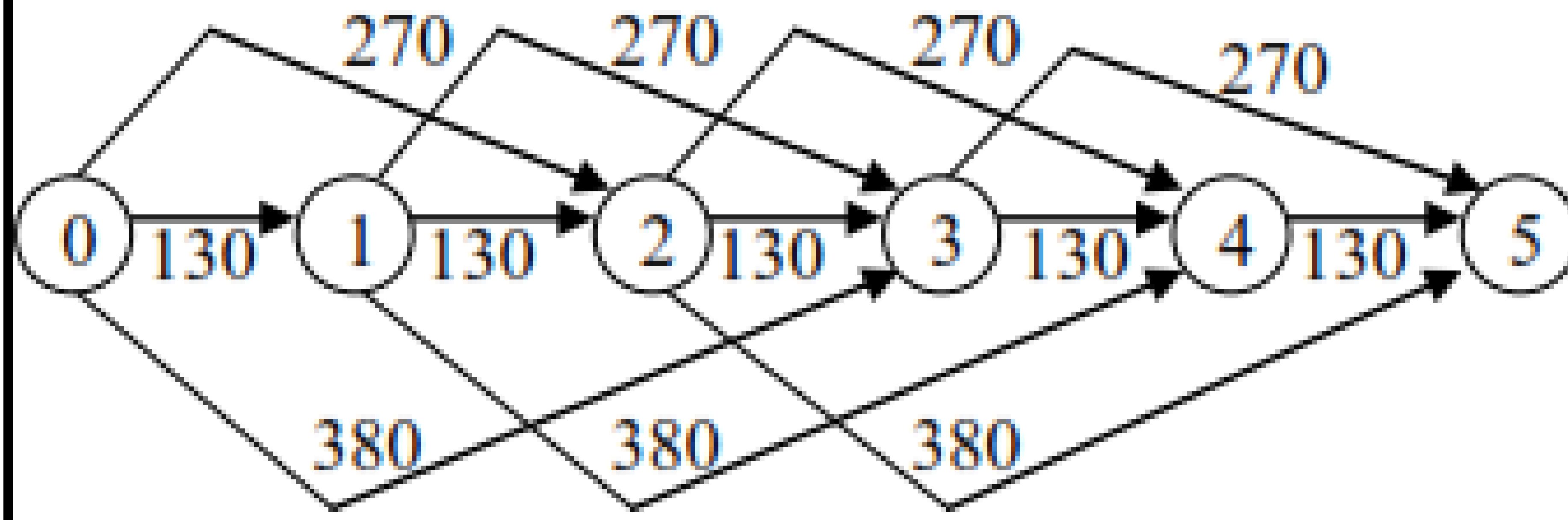

Figure 15.1: Converting the equipment replacement problem to a shortest route problem.

From Chinneck, 2010, p5.

©2012 St. John's University of Tanzania

Example 2

- Simultaneous Failure (Chinneck, 2010, Ch15, p9)
  - Objective function
    - Failure probability at any location = product of the failure probability at each location
  - Stages = locations
  - Decision = backups to assign to each stage
  - End stage = assign all the available backups
- Specific case: Hard drives
- Recursion is multiplicative

©2012 St. John's University of Tanzania

Example 2 Data

		Location		
		A	**B**	**C**
backup	**0**	0.20	0.30	0.40
drives	**1**	0.10	0.20	0.25
assigned	**2**	0.05	0.10	0.15

$$f_t(d_t) = min_{x_t}\left[p_t(x_t) \times f_{t+1}(d_t - x_t)\right]$$

d_C	$x_C=0$	$x_C=1$	$x_C=2$	$f_C(d_C)$
0	**0.4**	-	-	0.4
1	0.4	**0.25**	-	0.25
2	0.4	0.25	0.15	**0.15**

d_B	$x_B=0$	$x_B=1$	$x_B=2$	$f_B(d_B)$	$d_C=d_B-x_B$
0	**0.12**	-	-	0.12	0
1	**0.075**	0.080	-	0.075	1
2	0.045	0.050	**0.040**	0.040	0

d_A	$x_A=0$	$x_A=1$	$x_A=2$	$f_A(d_A)$	$d_B=d_A-x_A$
2	0.0080	0.0075	**0.0060**	0.0060	0

Solution:
Assign all the backups to location A, which has the worst failure rate.

©2012 St. John's University of Tanzania

Efficiency

- Seemed tedious in our examples
- Efficient compared to brute force
  - Think of all the initial choices that would *not* lead to the correct final conclusion
- Example: 5 nodes to get to 6 stages, with each stage fully connected
  - 5^5= 3125 possible paths x 5 ops/ea = 15,625
  - Dykstra's algorithm = 105 operations
  - 105/15,525 = 0.7% of the work!

©2012 St. John's University of Tanzania

References / Further Reading

- Chinneck, John W. *Practical Optimization: A Gentle Introduction*. Ontario: Carleton University, 2011, Chapter 15.
  - Found at www.sce.carleton.ca/faculty/chinneck/po.html
- Rao, Singiresu S. *Engineering Optimization: Theory and Practice*, 3rd Ed. New Dehli: New Age International (P) Ltd, 2010, Chapter 9, pp 515-556.

©2012 St. John's University of Tanzania