LEC007 Inventory Management II

VG441 SS2021

Cong Shi
Industrial & Operations Engineering
University of Michigan

Wagner-Whitin Model

INPUT:

- Deterministic demand (non-stationary) over T periods

$$(d_1, d_2, d_3, \dots, d_{T-1}, d_T)$$

- No stockout is allowed
- Lead time L (setting $L = 0$ WLOG)
- Fixed cost $K > 0$ per order
- Purchase cost c per unit (setting $c = 0$ WLOG)
- Inventory hold cost $h > 0$ per unit per unit of time

without loss of generality

OUTPUT:

The optimal ordering strategy

Mixed Integer Linear Program (MILP)

套路① 得出 function, contraints
② minimize/maximize

Decision Variables

q_t = the number of units ordered in period
$y_t = 1$ if we order in period t, 0 otherwise
x_t = the inventory level at the end of period, with $x_0 \equiv 0$ initially empty

fixedcosts holding costs

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T} (Ky_t + hx_t) & \\
\text{subject to} & x_t = x_{t-1} + q_t - d_t & \forall t = 1, \ldots, T \\
& q_t \leq My_t & \forall t = 1, \ldots, T \\
& x_t \geq 0 & \forall t = 1, \ldots, T \\
& q_t \geq 0 & \forall t = 1, \ldots, T \\
& y_t \in \{0, 1\} & \forall t = 1, \ldots, T
\end{array}
$$

linking constr
M: huge number
non-negative constraints

Inventory-balance constraint

Linking constraint ("a common trick")

flow balance constraints

ZIO Property

Dynamic programming

structure of problem

- It is optimal to place orders only in time periods in which the inventory level is zero.

- This suggests that each order is of a size equal to the total demand in an integer number of subsequent periods, i.e., in period t, we either order d_t ,

  or $d_t + d_{t+1}$,

  or $d_t + d_{t+1} + d_{t+2}$,

  and so on.

$d_1=100$ $d_2=50$ $d_5=10$ $d_6=5$

就是说,最好一次性order多一点,
把后面几次的也加起来

Dynamic Programming

recursive relation of value function

- Define θ_t to be the optimal cost in periods $[t, T]$ if we place optimal orders over $[t, T]$.

holding cost

smaller subproblem

$$\theta_t = \min_{t<s\le T+1}\left\{\boxed{K + h\sum_{i=t}^{s-1}(i-t)d_i} + \theta_s\right\}$$

$d_t,\ d_t + d_{t+1},\ d_t + d_{t+1} + d_{t+2}, \ldots$

Cost of covering demands of periods $t, t+1, \ldots, s-1$

trivial problem

- Boundary condition:

$$\theta_{T+1} \equiv 0$$

1 t T

a subproblem $[t, T]$

θ_t = opt. value of subproblem

Dynamic Programming

- Backward induction

$$\theta_t = \min_{t < s \leq T+1} \left\{ K + h \sum_{i=t}^{s-1} (i - t) d_i + \theta_s \right\}. \qquad (3.39)$$

Algorithm 3.1 Wagner–Whitin algorithm

$\theta_{T+1} \leftarrow 0$ ▷ Initialization
for $t = T, \ldots, 1$ **do** ▷ Main loop
    $\theta_t \leftarrow$ right-hand side of (3.39) ▷ Minimization over s
    $s(t) \leftarrow$ argmin in right-hand side of (3.39)
end for
return θ_t, $s(t)$ for all $t = 1, \ldots, T$

Numerical Example

$K = 500, h = 2$ per period. The demands are 90, 120, 80, and 70.

$$\theta_5 = 0$$

$$\theta_4 = K + h\,(0 \cdot d_4) + \theta_5$$ 不断找更好的

order 90+120 in 1

$$= 500 \quad [s(4) = 5]$$

$$\theta_3 = \min\left\{K + h\,(0 \cdot d_3) + \theta_4, K + h\,(0 \cdot d_3 + 1 \cdot d_4) + \theta_5\right\}$$

3

$$= \min\{1000, 640\}$$

$$= 640 \quad [s(3) = 5]$$

80+70 in 3

$$\theta_2 = \min\left\{K + h\,(0 \cdot d_2) + \theta_3, K + h\,(0 \cdot d_2 + 1 \cdot d_3) + \theta_4\right.$$

$$\left.K + h\,(0 \cdot d_2 + 1 \cdot d_3 + 2 \cdot d_4) + \theta_5\right\}$$

$$= \min\{1140, 1160, 940\}$$

5

$$= 940 \quad [s(2) = 5]$$

$$\theta_1 = \min\left\{K + h\,(0 \cdot d_1) + \theta_2, K + h\,(0 \cdot d_1 + 1 \cdot d_2) + \theta_3\right.$$

$$K + h\,(0 \cdot d_1 + 1 \cdot d_2 + 2 \cdot d_3) + \theta_4$$

$$\left.K + h\,(0 \cdot d_1 + 1 \cdot d_2 + 2 \cdot d_3 + 3 \cdot d_4) + \theta_5\right\}$$

$$= \min\{1440, 1380, 1560, 1480\}$$

$$= 1380 \quad [s(1) = 3]$$ keep track ☆

condition: 90 – 120 – 80 – 70 ← 从这里思考

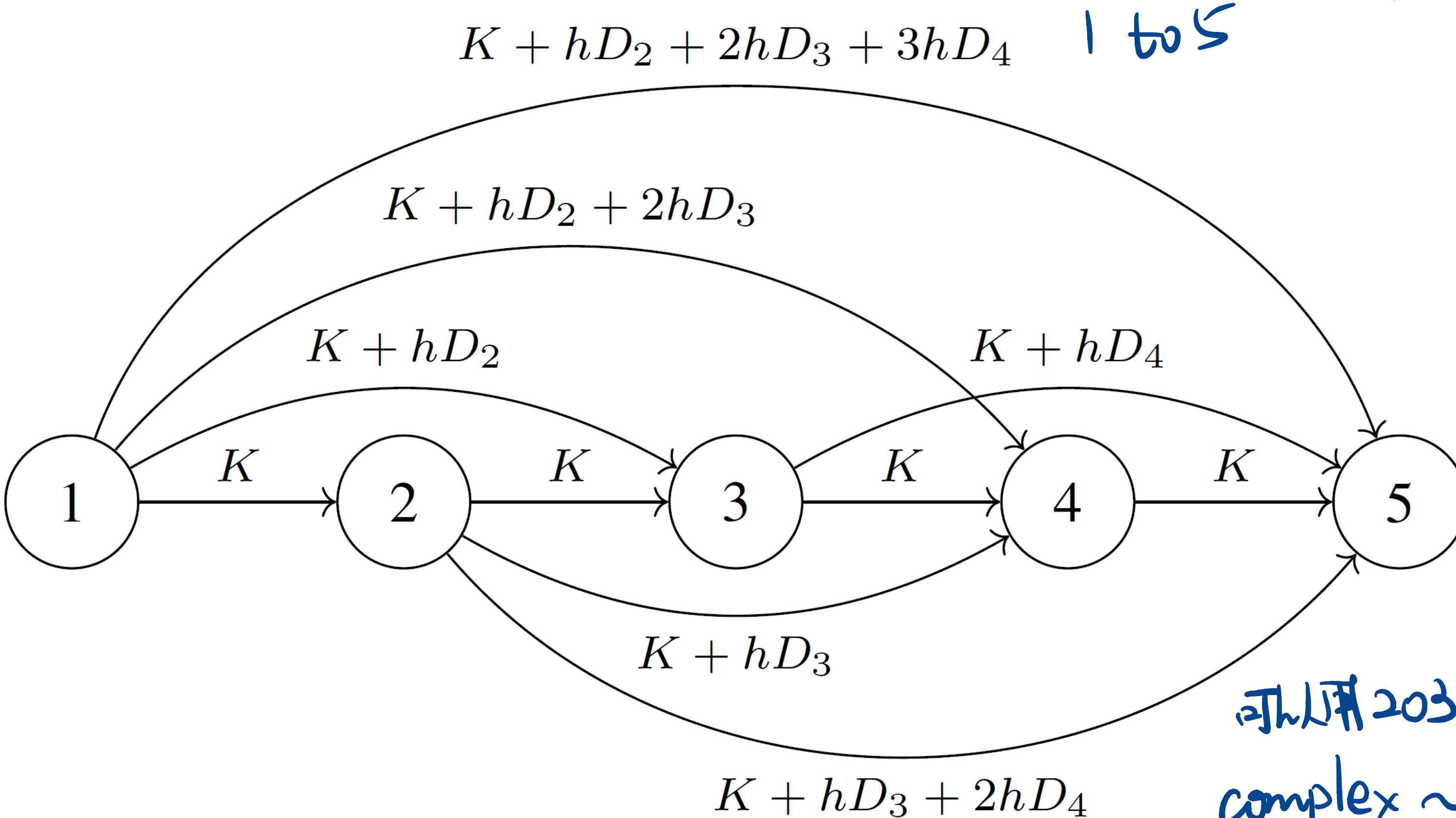
θ_3: subproblem over [3,4] order

80 + 70

80

If you think about this…

寻找shortest path from 1 to 5



可以用203来解决

complex ~ O(nm)

not fast enough

Figure 3.12 Wagner–Whitin network.

Shortest Path Problem

Input:

- Directed Graph $G(V, E)$ with $|V| = n, |E| = m$
- Each edge $e \in E$ has non-negative length $l_e \geq 0$
- Source vertex s

Output:

For each $v \in V$, compute

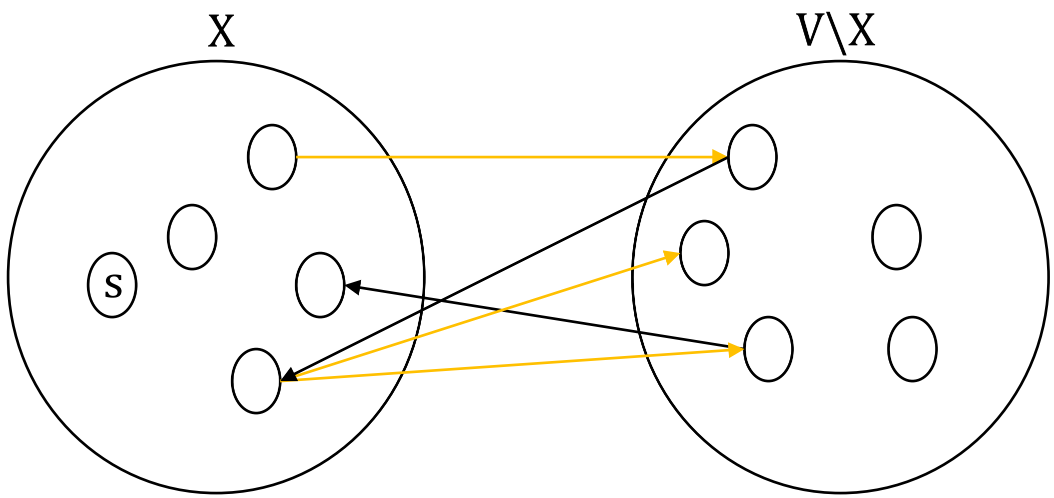
$L(v)$= length of a shortest s-v path in G

Fastest algorithm is called Dijkstra's Algorithm

Caveat: length/weight/travel time $l_e \geq 0$!

Dijkstra's Algorithm

- Initialize $X = \{s\}, A[s] = 0, B[s] = \emptyset$
- Main loop
  - While $X \neq V$

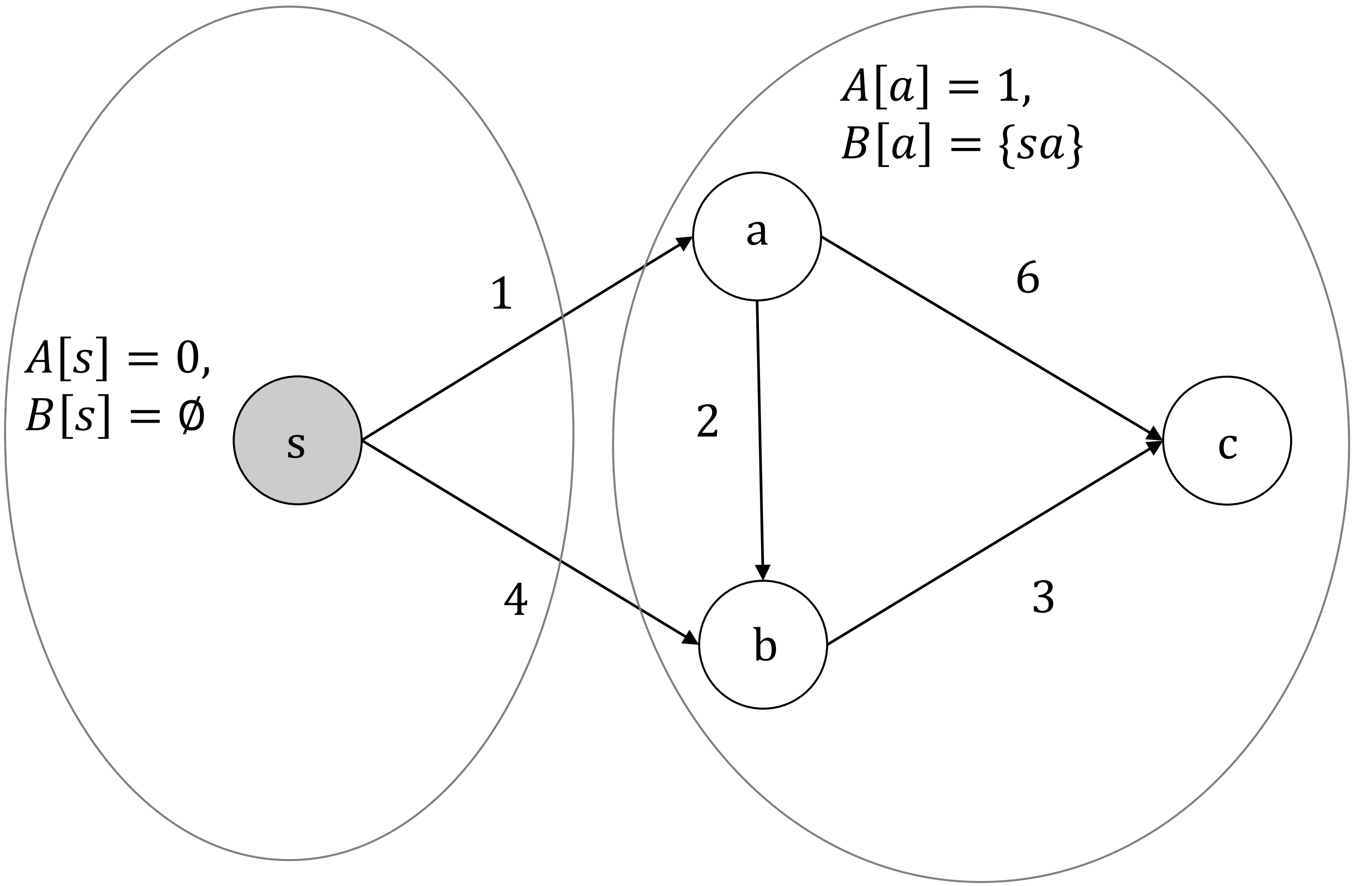


    - Of all edges $(v, w) \in E$ with $v \in$ X and $w \notin X$, pick the one that minimizes $A[v] + l_{vw}$ (Dijkstra's greedy criterion)
    - Call the minimizing edge (v^*, w^*) and add vertex w^* to X
    - Set $A[w^*] = A[v^*] + l_{v^*w^*}$
    - Set $B[w^*] = B[v^*] \cup (v^*, w^*)$

An Example

$A[v] + l_{vw}$ (Dijkstra's greedy criterion)

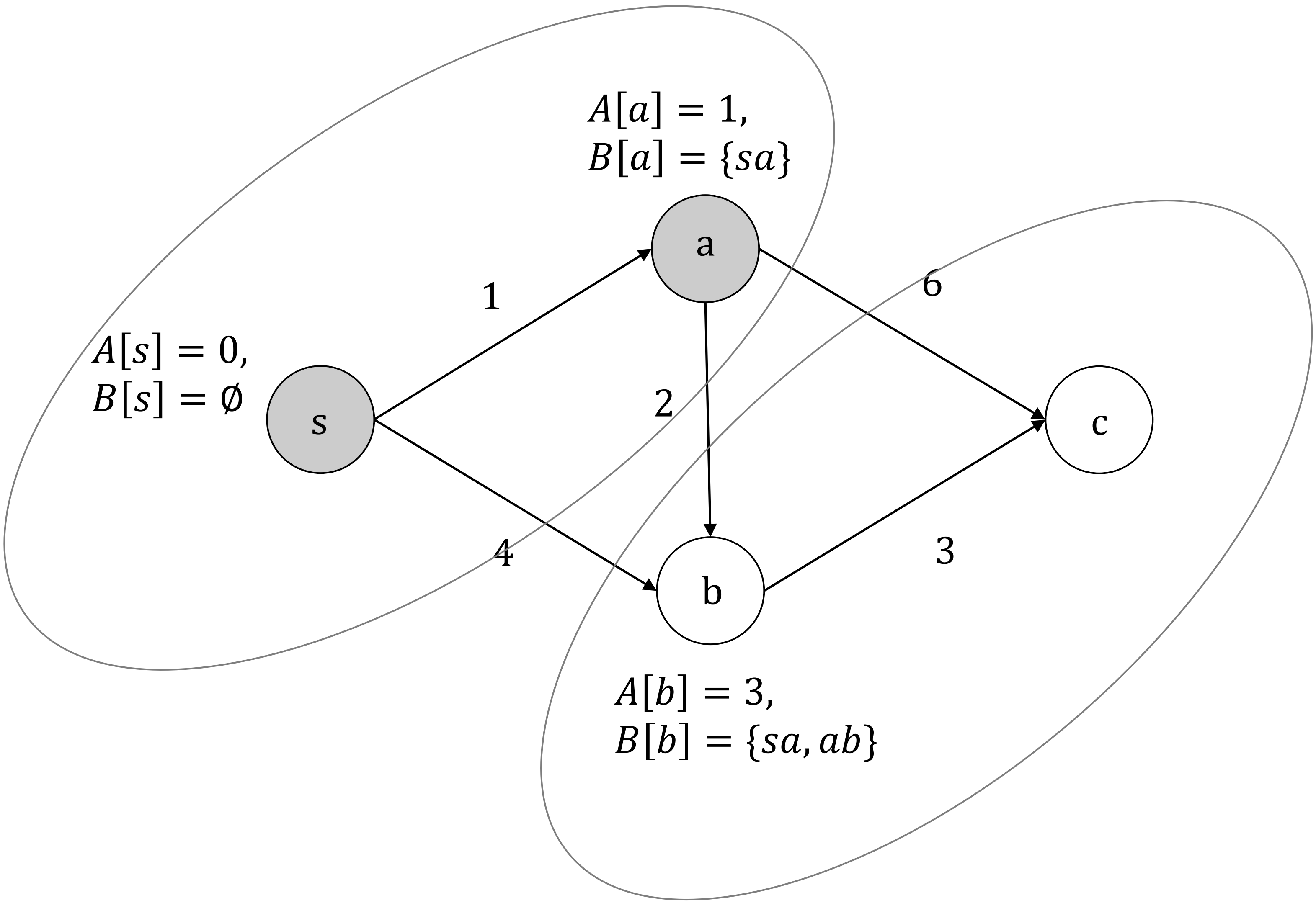
$min\ (A[s] + l_{sa}, A[s] + l_{sb}) = \min\ (0 + 1, 0 + 4) = 1$



An Example

$A[v] + l_{vw}$ (Dijkstra's greedy criterion)

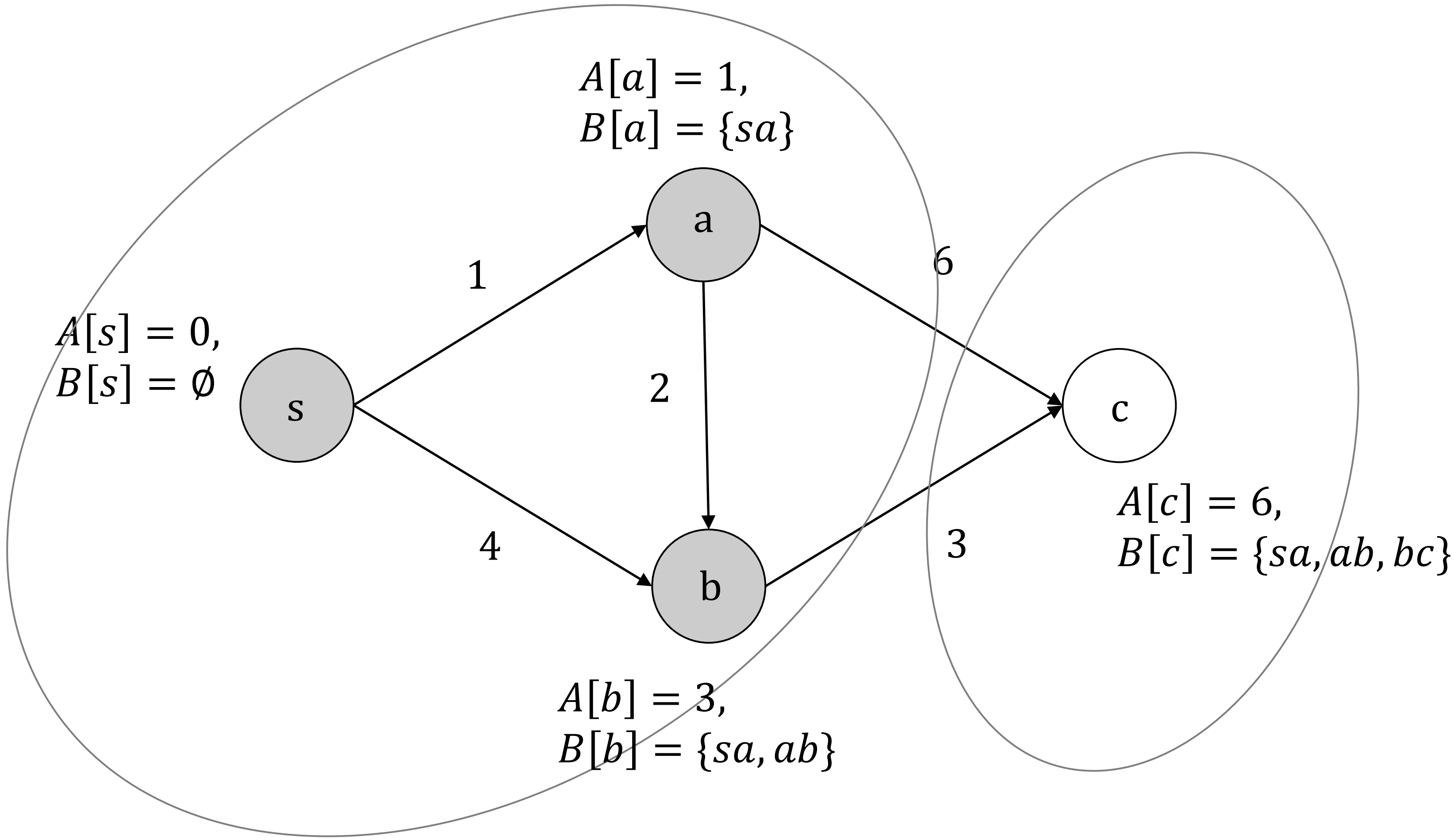
$\min\,(A[s] + l_{sb}, A[a] + l_{ab}, A[a] + l_{ac}) = \min\,(0 + 4, 1 + 2, 1 + 6) = 3$



An Example

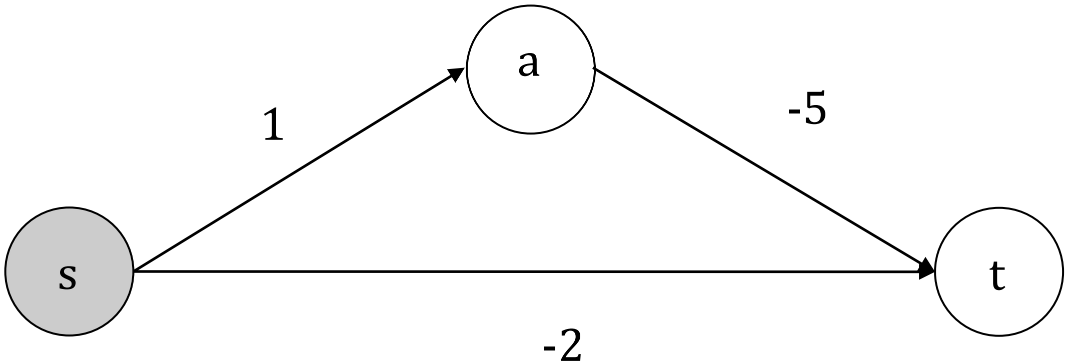
$A[v] + l_{vw}$ (Dijkstra's greedy criterion)

$\min\,(A[a] + l_{ac}, A[b] + l_{bc}) = \min\,(1 + 6, 3 + 3) = 6$



Non-Example

- Dijkstra is incorrect on this G



- Use dynamic programming in this case

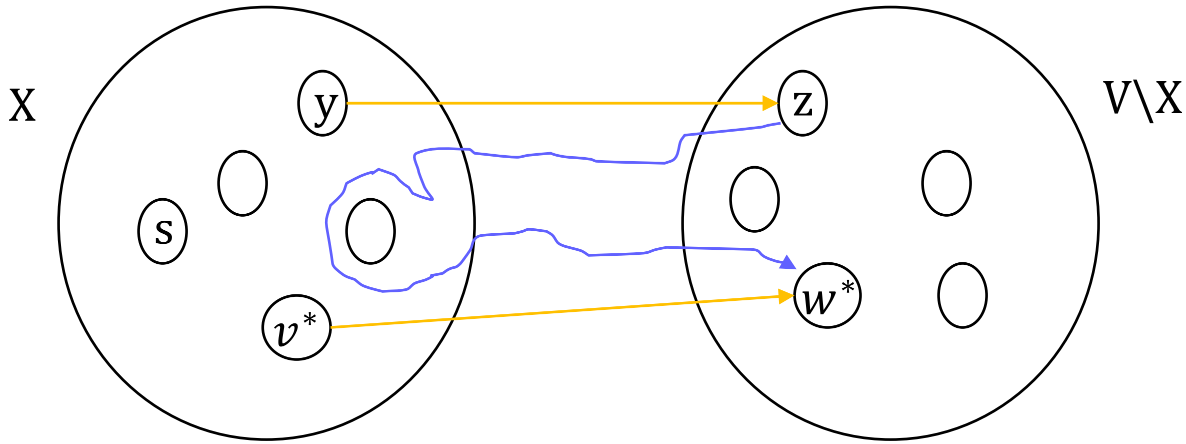
Proof of Correctness

Claim: $A[v] = L[v]$ where A is the output of Dijkstra and L is true shortest distance

Proof: By induction, base case $A[s] = L[s] = 0$ is true.

Inductive Hypothesis (I.H.): all previous iterations are corrrect, i.e.,

$$\forall v \in \mathrm{X}, A[v] = L[v] \text{ and } B[v] \text{ gives the shortest path}$$



In the current iteration, Dijkstra have chosen v^*w^*, we have $A[w^*] = A[v^*] + l_{v^*w^*}$

Now let P be any s $\rightarrow w^*$ path and it must "cross the frontier"

Length of P $\geq L(y) + l_{yz} + 0 = A(y) + l_{yz} + 0$, note that $L(y) = A(y)$ by I.H.

Also, by Dijkstra's greedy criterion,

$$\text{Our length} = A[v^*] + l_{v^*w^*} \leq A(y) + l_{yz} \leq \text{Length of P}$$

General Graph Search priority queue

- Let q be an abstract queue object
  - add(node), which adds a node into q
  - popFirst(), which pops the first node from q

- General graph search BFS.

  While q is not empty:

  - $v \leftarrow q.popFirst()$
  - For all neighbors u of v such that $u \notin q$:
    - add(u)

General Graph Search

- General graph search

  While q is not empty:

  - $v \leftarrow q.popFirst()$
  - For all neighbors u of v such that $u \notin q$:
    - add(u)

- If q is a standard LIFO stack, then DFS
- If q is a standard FIFO queue, then BFS
- If q is a priority queue, then Dijkstra
- If q is a priority queue with a heuristic, then A*

Priority Queue Implementation

- $A[s] \leftarrow 0$, and $A[v] \leftarrow \infty$ for all $v \in V \backslash \{s\}$
- $q.add(s)$
- While q is not empty:
  - $v \leftarrow q.popFirst()$

    Extract min-q
  - For all neighbors u of v such that $A[v] + l_{vu} \leq A[u]$
    - $A[u] \leftarrow A[v] + l_{vu}$
    - *q.update(u, A[u])*

      Decrease-key(q)

Runtime: If using binary minheap, $O\big((|E| + |V|)\log|V|\big)$
If using Fibonacci minheap, $O(|E| + |V|\log|V|)$

Summary

- Wagner-Whitin model
- Dynamic Programming
- Shortest Path (Dijkstra's Algorithm)

- Next Up: Stochastic Inventory Model