ECE276B: Planning & Learning in Robotics
Lecture 4: Deterministic Shortest Path

Instructor:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Zhichao Li: zhl355@eng.ucsd.edu

Ehsan Zobeidi: ezobeidi@eng.ucsd.edu

Ibrahim Akbar: iakbar@eng.ucsd.edu

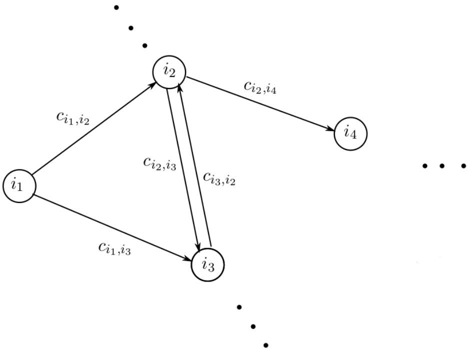
UC San Diego

JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

The Deterministic Shortest Path (DSP) Problem

- Consider a graph with a finite vertex space $\mathcal{V}$ and a weighted edge space $\mathcal{C} := \{(i, j, c_{ij}) \in \mathcal{V} \times \mathcal{V} \times \mathbb{R} \cup \{\infty\}\}$ where c_{ij} denotes the arc length or cost from vertex i to vertex j.



- **Objective**: find the shortest path from a start node s to an end node τ

- It turns out that the DSP problem is equivalent to the standard finite-horizon finite-space deterministic optimal control problem

The Deterministic Shortest Path (DSP) Problem

- **Path**: an ordered list $Q := (i_1, i_2, \ldots, i_q)$ of nodes $i_k \in \mathcal{V}$.
- **Set of all paths from $s \in \mathcal{V}$ to $\tau \in \mathcal{V}$**: $\mathbb{Q}_{s,\tau}$.
- **Path Length**: sum of the arc lengths over the path: $J^Q = \sum_{t=1}^{q-1} c_{t,t+1}$.
- **Objective**: find a path $Q^* = \underset{Q \in \mathbb{Q}_{s,\tau}}{\arg\min}\, J^Q$ that has the smallest length from node $s \in \mathcal{V}$ to node $\tau \in \mathcal{V}$
- **Assumption**: For all $i \in \mathcal{V}$ and for all $Q \in \mathbb{Q}_{i,i}$, $J^Q \geq 0$, i.e., there are no negative cycles in the graph and $c_{i,i} = 0$ for all $i \in \mathcal{X}$.
- Solving DSP problems:
  - map to a deterministic finite-state system and apply (backward) DP
  - label correcting methods (variants of a "forward" DP algorithm)

Deterministic Finite State (DFS) Optimal Control Problem

- Deterministic problem: closed-loop control does not offer any advantage over open-loop control
- Consider the standard problem with no disturbances w_t and finite state space $\mathcal{X}$. Given $x_0 \in \mathcal{X}$ the goal is to construct an optimal control sequence $u_{0:T-1}$ such that:

$$\begin{aligned} \min_{u_{0:T-1}} \quad & \mathfrak{q}(x_T) + \sum_{t=0}^{T-1} \ell_t(x_t, u_t) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t), \ t = 0, \ldots, T-1 \\ & x_t \in \mathcal{X}, \ u_t \in \mathcal{U}(x_t), \end{aligned}$$

- This problem can be solved via the Dynamic Programming algorithm

Equivalence of DFS and DSP Problems (DFS to DSP)

- We can construct a graph representation of the DFS problem.

- Every state $x_t \in \mathcal{X}$ at time t is represented by a node in the graph:
$$\mathcal{V} := \left(\bigcup_{t=0}^{T} \{(t, x_t) \mid x_t \in \mathcal{X}\} \right) \cup \{\tau\}$$

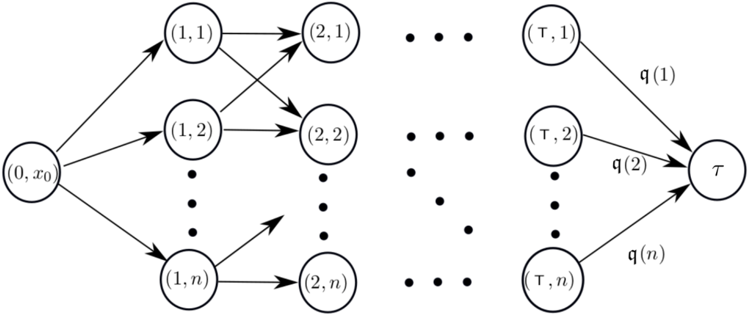
- The given x_0 is the starting node $s := (0, x_0)$.

- An artificial terminal node τ is added with arc lengths to τ equal to the terminal costs of the DFS.

- The arc length between any two nodes is the (smallest) stage cost between them and is ∞ if there is no control that links them:

$$\mathcal{C} := \left\{ ((t, x_t), (t+1, x_{t+1}), c) \middle| c = \min_{\substack{u \in \mathcal{U}(x_t) \\ \text{s.t. } x_{t+1} = f(x_t, u)}} \ell_t(x_t, u) \right\} \bigcup \{((T, x_T), \tau, \mathfrak{q}(x_T))\}$$

Equivalence of DFS and DSP Problems (DFS to DSP)



Equivalence of DFS and DSP Problems (DSP to DFS)

- Consider an DSP problem with vertex space $\mathcal{V}$, weighted edge space $\mathcal{C}$, start node $s \in \mathcal{V}$ and terminal node $\tau \in \mathcal{V}$.
- Due to the assumption of no cycles with negative cost, the optimal path need not have more than $|\mathcal{V}|$ elements
- We can formulate the DSP problem as a DFS with $T := |\mathcal{V}| - 1$ stages:
  - State space: $\mathcal{X}_0 := \{s\}$, $\mathcal{X}_T := \{\tau\}$, $\mathcal{X}_t := \mathcal{V} \setminus \{\tau\}$ for $t = 1, \ldots, T-1$
  - Control space: $\mathcal{U}_{T-1} := \{\tau\}$ and $\mathcal{U}_t := \mathcal{V} \setminus \{\tau\}$ for $t = 0, \ldots, T-2$
  - Dynamics: $x_{t+1} = u_t$ for $u_t \in \mathcal{U}_t$, $t = 0, \ldots, T-1$
  - Costs: $\mathfrak{q}(\tau) := 0$ and $\ell_t(x_t, u_t) = c_{x_t, u_t}$ for $t = 0, \ldots, T-1$

Dynamic Programming Applied to DSP

- Due to the equivalence, the DSP can be solved via the DP algorithm
- $V_t(i)$ is the optimal cost of getting from node i to node τ in $T - t$ steps
- Upon termination, $V_0(s) = J^{Q^*}$
- The algorithm can be terminated early if $V_t(i) = V_{t+1}(i)$, $\forall i \in \mathcal{V} \setminus \{\tau\}$

Algorithm 1 Deterministic Shortest Path via Dynamic Programming

```
Input: node set 𝒱, start s ∈ 𝒱, goal τ ∈ 𝒱, and costs c_ij for i, j ∈ 𝒱
T = |𝒱| − 1
V_T(τ) = 0
V_{T−1}(i) = c_{i,τ},   ∀i ∈ 𝒱 \ {τ}
π_{T−1}(i) = τ,   ∀i ∈ 𝒱 \ {τ}
for t = (T − 2), ..., 0 do
    Q_t(i, j) = c_{i,j} + V_{t+1}(j),   ∀i, j ∈ 𝒱 \ {τ}
    V_t(i) = min_{j∈𝒱\{τ}} Q_t(i, j),   ∀i ∈ 𝒱 \ {τ}
    π_t(i) = arg min_{j∈𝒱\{τ}} Q_t(i, j),   ∀i ∈ 𝒱 \ {τ}
    if V_t(i) = V_{t+1}(i), ∀i ∈ 𝒱 \ {τ} then
        break
```

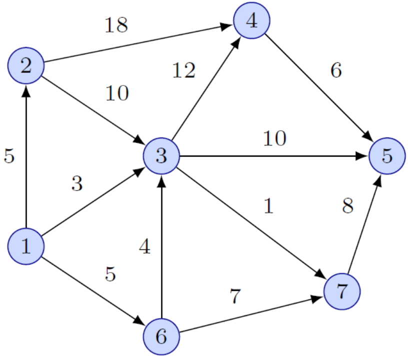
Forward DP Algorithm

- The DSP problem is symmetric: an optimal path from s to τ is also a shortest path from τ to s, where all arc directions are flipped.
- This view leads to a "forward Dynamic Programming" algorithm.
- $V_t^F(j)$ is the **optimal cost-to-arrive** to node j from node s in t moves

Algorithm 2 Deterministic Shortest Path via Forward Dynamic Programming

Input: node set $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs c_{ij} for $i, j \in \mathcal{V}$
$T = |\mathcal{V}| - 1$
$V_0^F(s) = 0$
$V_1^F(j) = c_{s,j}, \quad \forall j \in \mathcal{V} \setminus \{s\}$
for $t = 2, \ldots, T$ **do**
    $V_t^F(j) = \min_{i \in \mathcal{V} \setminus \{s\}} \left(c_{i,j} + V_{t-1}^F(i)\right), \quad \forall j \in \mathcal{V} \setminus \{s\}$
    if $V_t^F(i) = V_{t-1}^F(i), \forall i \in \mathcal{V} \setminus \{s\}$ **then**
        break

Forward DP Algorithm (Example)



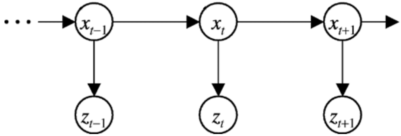
- $s = 1$ and $\tau = 5$
- $T = |\mathcal{V}| - 1 = 6$

	1	2	3	4	5	6	7
V_0^F	0	∞	∞	∞	∞	∞	∞
V_1^F	0	5	3	∞	∞	5	∞
V_2^F	0	5	3	15	13	5	4
V_3^F	0	5	3	15	12	5	4
V_4^F	0	5	3	15	12	5	4

- Since $V_t^F(i) = V_{t-1}^F(i), \forall i \in \mathcal{V}$ at time $t = 4$, the algorithm can terminate early, i.e., without computing $V_5^F(i)$ and $V_6^F(i)$

Hidden Markov Model

- **States**: $x_t \in \mathcal{X} := \{1, \ldots, N\}$
- **Observations**: $z_t \in \mathcal{Z} := \{1, \ldots, M\}$



- **Markov Assumptions**:
  - Given x_t, x_{t+1} is independent of everything else, e.g., $x_{0:t-2}$ and $z_{1:t}$
  - Given x_t, z_t is independent of everything else, e.g., $x_{0:t-2}$ and $z_{1:t-1}$
- **Model Parameters**:
  - **Prior**: probability mass function $p_{0|0} \in [0,1]^N$ with $p_{0|0}(i) := \mathbb{P}(x_0 = i)$
  - **Motion model**: matrix $A \in \mathbb{R}^{N \times N}$ with $A(i,j) = \mathbb{P}(x_{t+1} = j \mid x_t = i)$
  - **Observation model** matrix $B \in \mathbb{R}^{N \times M}$ with $B(i,j) = \mathbb{P}(z_t = j \mid x_t = i)$
- **Predicted pmf**: $p_{t+1|t}(i) := \mathbb{P}(x_{t+1} = i \mid z_{1:t} = \bar{z}_{1:t})$
- **Updated pmf**: $p_{t+1|t+1}(i) := \mathbb{P}(x_{t+1} = i \mid z_{1:t+1} = \bar{z}_{1:t+1})$

Bayes Filter Applied to HMM

- **Prediction step**: given $p_{t|t}$ compute $p_{t+1|t}$:

$$\begin{aligned}
p_{t+1|t}(j) &= \mathbb{P}(x_{t+1} = j \mid z_{1:t} = \bar{z}_{1:t}) \\
&\overset{\text{Total prob.}}{=\!=\!=} \sum_{i=1}^{N} \mathbb{P}(x_{t+1} = j, x_t = i \mid z_{1:t} = \bar{z}_{1:t}) \\
&\overset{\text{Cond. prob.}}{=\!=\!=} \sum_{i=1}^{N} \mathbb{P}(x_{t+1} = j \mid x_t = i, z_{1:t} = \bar{z}_{1:t})\mathbb{P}(x_t = i \mid z_{1:t} = \bar{z}_{1:t}) \\
&\overset{\text{Markov}}{=\!=\!=} \sum_{i=1}^{N} A(i,j) p_{t|t}(i)
\end{aligned}$$

- **Concise prediction step**: $\boxed{p_{t+1|t} = A^T p_{t|t}}$

Bayes Filter Applied to HMM

- **Update step**: given $p_{t+1|t}$ compute $p_{t+1|t+1}$:

$$
\begin{aligned}
p_{t+1|t+1}(j) =& \mathbb{P}(x_{t+1} = j \mid z_{1:t+1} = \bar{z}_{1:t+1}) \\
\overset{\text{Bayes}}{\underset{\text{Rule}}{=\!=}}& \frac{\mathbb{P}(z_{t+1} = \bar{z}_{t+1} \mid x_{t+1} = j, z_{1:t} = \bar{z}_{1:t})\mathbb{P}(x_{t+1} = j \mid z_{1:t} = \bar{z}_{1:t})}{\mathbb{P}(z_{t+1} = \bar{z}_{t+1} \mid z_{1:t} = \bar{z}_{1:t})} \\
=& \frac{B(j, \bar{z}_{t+1})p_{t+1|t}(j)}{\sum_{i=1}^{N} B(i, \bar{z}_{t+1})p_{t+1|t}(i)}
\end{aligned}
$$

- **Concise update step**: let $b_{t+1} := \begin{bmatrix} B(1, \bar{z}_{t+1}) & \cdots & B(N, \bar{z}_{t+1}) \end{bmatrix}^T$:

$$
\boxed{p_{t+1|t+1} = \frac{\mathbf{diag}(b_{t+1})p_{t+1|t}}{b_{t+1}^T p_{t+1|t}}}
$$

The Three Main HMM Problems

P1 Given an observation sequence $\bar{z}_{1:T}$ and model parameters $\theta := (p_{0|0}, A, B)$, how do we efficiently compute the likelihood $p_\theta(\bar{z}_{1:T})$ of the observation sequence?

- **Forward Procedure** (Filtering) and **Forward-Backward Procedure** (Smoothing)

P2 Given an observation sequence $\bar{z}_{1:T}$ and model parameters $\theta := (p_{0|0}, A, B)$, how do we choose a corresponding state sequence $x^*_{0:T}$ which best explains the observations, i.e., maximizes $p(x_{0:T}, \bar{z}_{1:T})$?

- **Viterbi Decoding**

P3 How do we adjust the model parameters $\theta := (p_{0|0}, A, B)$ to maximize $p_\theta(\bar{z}_{1:T})$?

- **Maximum Likelihood Estimation** (MLE) and **Baum-Welch Algorithm** (Expectation Maximization)

Viterbi Decoding

- **Viterbi Decoding**: computes the most-likely state sequence corresponding to the observations:

$$x^*_{0:T} = \arg\max_{x_{0:T}} p(x_{0:T}, \bar{z}_{1:T})$$

- The Viterbi algorithm keeps track of two variables:
  - Likelihood of the observed sequence with the most likely state assignment up to $t-1$:

$$\delta_t(i) := \max_{x_{0:t-1}} p(x_{0:t-1}, x_t = i, \bar{z}_{1:t})$$

  - State from the previous time that leads to the maximum for the current state at time t:

$$\psi_t(i) := \arg\max_{x_{t-1}} \max_{x_{0:t-2}} p(x_{0:t-1}, x_t = i, \bar{z}_{1:t})$$

Viterbi Decoding

- **Initialize**: $\delta_0(i) = p_{0|0}(i)$ and $\psi_0(i) = 0$
- **Forward Pass** for $t = 1, \ldots, T$

$$\begin{aligned}\delta_t(j) &= \max_i \mathbb{P}(z_t = \bar{z}_t \mid x_t = j)\mathbb{P}(x_t = j \mid x_{t-1} = i)\delta_{t-1}(i) \\ &= \max_i B(j, \bar{z}_t)A(i,j)\delta_{t-1}(i) \\ \psi_t(j) &= \arg\max_i \mathbb{P}(z_t = \bar{z}_t \mid x_t = j)\mathbb{P}(x_t = j \mid x_{t-1} = i)\delta_{t-1}(i) \\ &= \arg\max_i B(j, \bar{z}_t)A(i,j)\delta_{t-1}(i)\end{aligned}$$

$$\boxed{p(x^*_{0:T}, \bar{z}_{1:T}) = \max_i \delta_T(i)} \qquad \boxed{x^*_T = \arg\max_i \delta_T(i)}$$

- **Backward Pass** for $t = T-1, \ldots, 0$:

$$\boxed{x^*_t = \psi_{t+1}(x^*_{t+1})}$$

Viterbi Decoding

- By the conditioning rule, $p(x_{0:T}, \bar{z}_{1:T}) = p(x_{0:T} \mid \bar{z}_{1:T}) p(\bar{z}_{1:T})$. Since $p(\bar{z}_{1:T})$ is fixed and positive, maximizing $p(x_{0:T} \mid \bar{z}_{1:T})$ is equivalent to maximizing $p(x_{0:T}, \bar{z}_{1:T})$
- **Joint probability density function**:

$$p(x_{0:T}, z_{1:T}) = \underbrace{p_{0|0}(x_0)}_{\text{prior}} \prod_{t=1}^{T} \underbrace{A(x_{t-1}, x_t)}_{\text{motion model}} \prod_{t=1}^{T} \underbrace{B(x_t, z_t)}_{\text{observation model}}$$

- **Idea**: we can express $\max_{x_{0:T}} p(x_{0:T}, z_{1:T})$ as a shortest path problem:

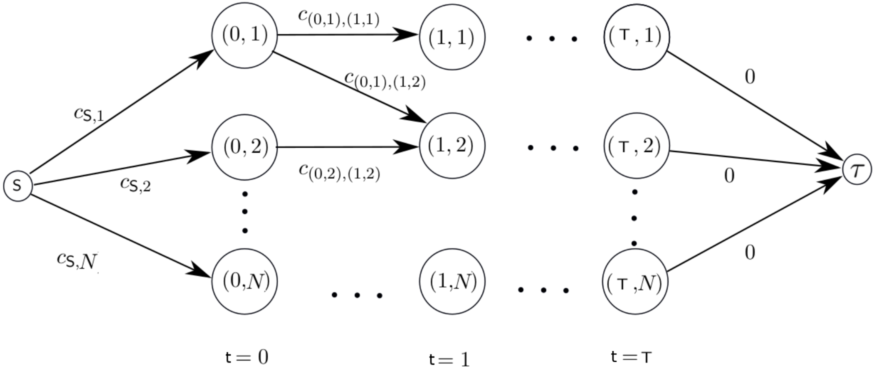
$$\min_{x_{0:T}} \left(c_{s,(0,x_0)} + \sum_{t=1}^{T} c_{(t-1,x_{t-1}),(t,x_t)} \right)$$

  where:

$$\begin{aligned} c_{s,(0,x_0)} &:= -\log\left(p_{0|0}(x_0)\right) \\ c_{(t-1,x_{t-1}),(t,x_t)} &:= -\log\left(A(x_{t-1}, x_t) B(x_t, z_t)\right) \end{aligned}$$

Viterbi Decoding

- Construct a graph of state-time pairs with artificial starting node s and terminal node τ



- Computing the shortest path via the forward DP algorithm leads to the forward pass of the Viterbi algorithm!

Label Correcting Methods for the SP Problem

- The DP algorithm computes the shortest paths from *all* nodes to the goal. Often many nodes are not part of the shortest path from s to τ
- The **label correcting (LC) algorithm** is a general algorithm for SP problems that does not necessarily visit every node of the graph
- LC algorithms prioritize the visited nodes using the **cost-to-arrive** values
- **Key Ideas**:
  - **Label** g_i: keeps (an estimate of) the lowest cost from s to each visited node $i \in \mathcal{V}$
  - Each time g_i is reduced, the labels g_j of the **children** of i can be corrected: $g_j = g_i + c_{ij}$
  - **OPEN**: set of nodes that can potentially be part of the shortest path to τ

Label Correcting Algorithm

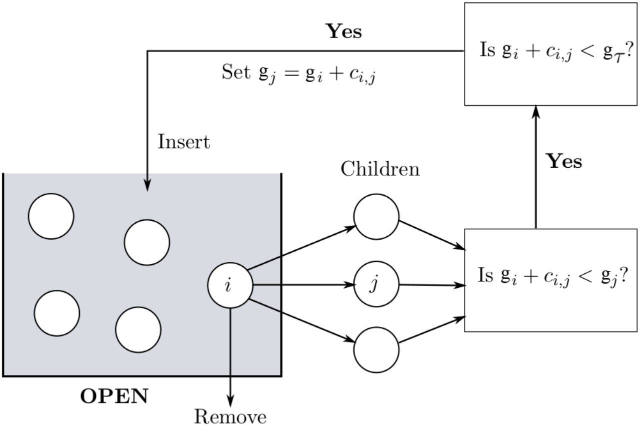
Algorithm 3 Label Correcting Algorithm

OPEN $\leftarrow \{s\}$, $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
while OPEN is not empty **do**
  Remove i from OPEN
  for $j \in$ Children(i) **do**
    if $(g_i + c_{ij}) < g_j$ **and** $(g_i + c_{ij}) < g_\tau$ **then** $\triangleright$ Only when $c_{ij} \geq 0$ for all $i, j \in \mathcal{V}$
      $g_j \leftarrow (g_i + c_{ij})$
      Parent(j) $\leftarrow i$
      if $j \neq \tau$ **then**
        OPEN $\leftarrow$ OPEN $\cup \{j\}$

Theorem

If there exists at least one finite cost path from s to τ, then the Label Correcting (LC) algorithm terminates with $g_\tau = J^{Q^*}$ (the shortest path from s to τ). Otherwise, the LC algorithm terminates with $g_\tau = \infty$.

Label Correcting Algorithm



Label Correcting Algorithm Proof

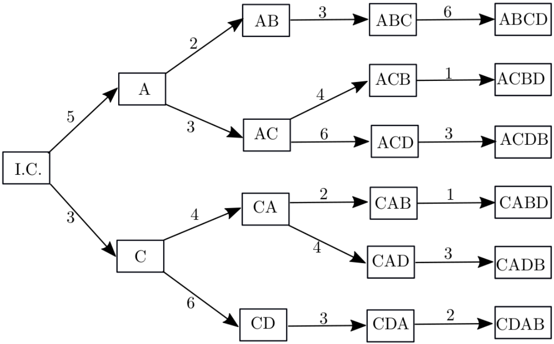
1. **Claim**: The LC algorithm terminates in a finite number of steps
   - Each time a node j enters OPEN, its label is decreased and becomes equal to the length of some path from s to j.
   - The number of distinct paths from s to j whose length is smaller than any given number is finite (**no negative cycles assumption**)
   - There can only be a finite number of label reductions for each node j
   - Since the LC algorithm removes nodes from OPEN in line 3, the algorithm will eventually terminate
2. **Claim**: The LC algorithm terminates with $g_\tau = \infty$ if there is no finite cost path from s to τ
   - A node $i \in \mathcal{V}$ is in OPEN only if there is a finite cost path from s to i
   - If there is no finite cost path from s to τ, then for any node i in OPEN $c_{i,\tau} = \infty$; otherwise there would be a finite cost path from s to τ
   - Since $c_{i,\tau} = \infty$ for every i in OPEN, line 5 ensures that g_τ is never updated and remains ∞

Label Correcting Algorithm Proof

3. **Claim**: The LC algorithm terminates with $g_\tau = J^{Q^*}$ if there is at least one finite cost path from s to τ
   - Let $Q^* = (s, i_1, i_2, \ldots, i_{q-2}, \tau) \in \mathbb{Q}_{s,\tau}$ be a shortest path from s to τ with length J^{Q^*}
   - By the principle of optimality $Q^*_m := (s, i_1, \ldots, i_m)$ is the shortest path from s to i_m with length $J^{Q^*_m}$ for any $m = 1, \ldots, q-2$
   - Suppose that $g_\tau > J^{Q^*}$ (proof by contradiction)
   - Since g_τ only decreases in the algorithm and every cost is nonnegative, $g_\tau > J^{Q^*_m}$ for all $m = 2, \ldots, q-2$
   - Thus, i_{q-2} does not enter OPEN with $g_{i_{q-2}} = J^{Q^*_{q-2}}$ since if it did, then the next time i_{q-2} is removed from OPEN, g_τ would be updated to J_{Q^*}
   - Similarly, i_{q-3} will not enter OPEN with $g_{i_{q-3}} = J^{Q^*_{q-3}}$. Continuing this way, i_1 will not enter open with $g_{i_1} = J^{Q^*_1} = c_{s,i_1}$ but this happens at the first iteration of the algorithm, which is a contradiction!

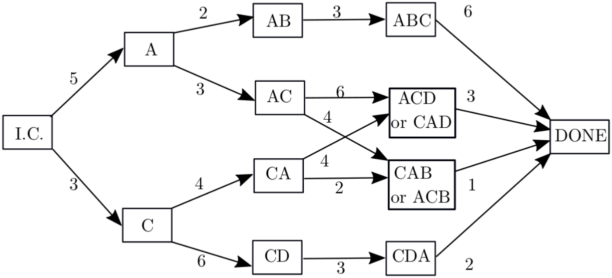
Example: Deterministic Scheduling Problem

- Consider a deterministic scheduling problem where 4 operations A, B, C, D are used to produce a product
- Rules: Operation A must occur before B, and C before D
- Cost: there is a transition cost between each two operations:



Example: Deterministic Scheduling Problem

- The state transition diagram of the scheduling problem can be simplified in order to reduce the number of nodes

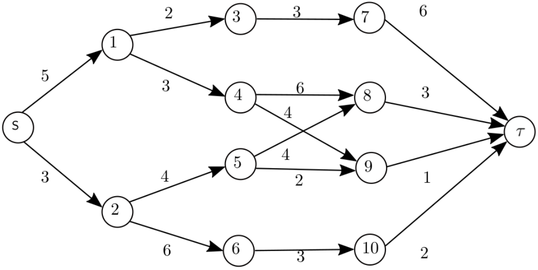


- This results in a DFS problem with $T = 4$, $\mathcal{X}_0 = \{\text{I.C.}\}$, $\mathcal{X}_1 = \{\text{A}, \text{C}\}$, $\mathcal{X}_2 = \{\text{AB}, \text{AC}, \text{CA}, \text{CD}\}$, $\mathcal{X}_3 = \{\text{ABC}, \text{ACD or CAD}, \text{CAB or ACB}, \text{CDA}\}$, $\mathcal{X}_T = \{DONE\}$

- We can map the DFS problem to a DSP problem

Example: Deterministic Scheduling Problem

- We can map the DFS problem to a SP problem and apply the LC algorithm
- Keeping track of the parents when a child node is added OPEN, it can be determined that a shortest path is $(s, 2, 5, 9, \tau)$ with total cost 10, which corresponds to $(C, CA, CAB, CABD)$ in the original problem



Iteration	Remove	OPEN	g_s	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_τ
0	–	s	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	s	1, 2	0	5	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	2	1, 5, 6	0	5	3	∞	∞	7	9	∞	∞	∞	∞	∞
3	6	1, 5, 10	0	5	3	∞	∞	7	9	∞	∞	∞	12	∞
4	10	1, 5	0	5	3	∞	∞	7	9	∞	∞	∞	12	14
5	5	1, 8, 9	0	5	3	∞	∞	7	9	∞	11	9	12	14
6	9	1, 8	0	5	3	∞	∞	7	9	∞	11	9	12	10
7	8	1	0	5	3	∞	∞	7	9	∞	11	9	12	10
8	1	3, 4	0	5	3	7	8	7	9	∞	11	9	12	10
9	4	3	0	5	3	7	8	7	9	∞	11	9	12	10
10	3	–	0	5	3	7	8	7	9	∞	11	9	12	10

Specific Label Correcting Methods

- There is freedom in selecting the node to be removed from OPEN at each iteration, which gives rise to several different methods:
  - **Breadth-first search** (BFS) (**Bellman-Ford Algorithm**): "first-in, first-out" policy with OPEN implemented as a **queue**.
  - **Depth-first search** (DFS): "last-in, first-out" policy with OPEN implemented as a **stack**; often saves memory
  - **Best-first search** (**Dijkstra's Algorithm**): the node with minimum label $i^* = \arg\min_{j \in OPEN} g_j$ is removed, which guarantees that *a node will enter OPEN at most once*. OPEN is implemented as a **priority queue**.
  - **D'Esopo-Pape method**: removes nodes at the top of OPEN. If a node has been in OPEN before it is inserted at the top; otherwise at the bottom.
  - **Small-label-first** (SLF): removes nodes at the top of OPEN. If $d_i \leq d_{TOP}$ node i is inserted at the top; otherwise at the bottom.
  - **Large-label-last** (LLL): the top node is compared with the average of OPEN and if it is larger, it is placed at the bottom of OPEN; otherwise it is removed.

A* Algorithm

- The **A* algorithm** is a modification to the LC algorithm in which the requirement for admission to OPEN is strengthened:

$$\text{from} \quad g_i + c_{ij} < g_\tau \quad \text{to} \quad g_i + c_{ij} + h_j < g_\tau$$

  where h_j is a positive lower bound on the optimal cost to get from node j to τ, known as **heuristic**.

- The more stringent criterion can reduce the number of iterations required by the LC algorithm

- The heuristic is constructed depending on special knowledge about the problem. The more accurately h_j estimates the optimal cost from j to τ, the more efficient the A* algorithm becomes!