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

Lecturer:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Tianyu Wang: tiw161@eng.ucsd.edu

Yongxi Lu: yol070@eng.ucsd.edu

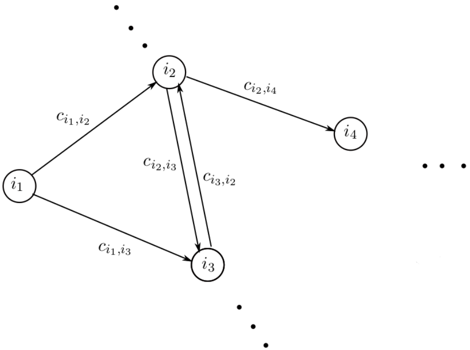
UC San Diego

JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

The Shortest Path (SP) 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 SP problem is equivalent to the standard finite-horizon finite-space deterministic optimal control problem

The Shortest Path (SP) 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 SP 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:

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

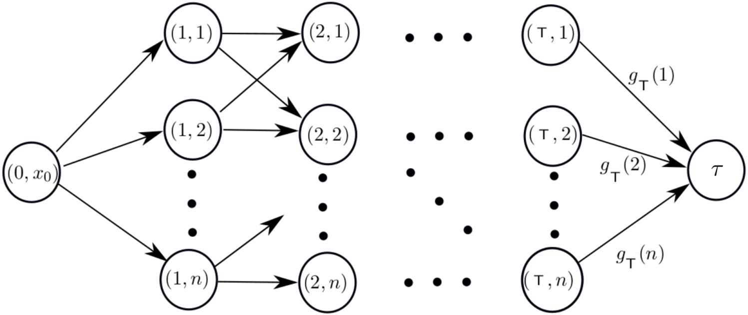
- This problem can be solved via the Dynamic Programming algorithm

Equivalence of DFS and SP Problems (DFS to SP)

- 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:

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

Equivalence of DFS and SP Problems (DFS to SP)



Equivalence of DFS and SP Problems (SP to DFS)

- Consider an SP 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 SP 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: $g_T(\tau) := 0$ and $g_t(x_t, u_t) = c_{x_t, u_t}$ for $t = 0, \ldots, T-1$

Dynamic Programming Applied to DFS and SP

- Due to the equivalence, the DFS/SP can be solved via the DP algorithm

$$
\begin{aligned}
V_T(\tau) &= g_T(\tau) = 0, \\
V_{T-1}(i) &= \min_{u \in \mathcal{U}_t} \left(g_t(i,u) + V_{t+1}(u)\right) = c_{i,\tau}, \quad \forall i \in \mathcal{V} \setminus \{\tau\} \\
V_t(i) &= \min_{u \in \mathcal{U}_t} \left(g_t(i,u) + V_{t+1}(u)\right) = \min_{j \in \mathcal{V}\setminus\{\tau\}} \left(c_{i,j} + V_{t+1}(j)\right), \; i \in \mathcal{V} \setminus \{\tau\}, t = T-2, \ldots, 0
\end{aligned}
$$

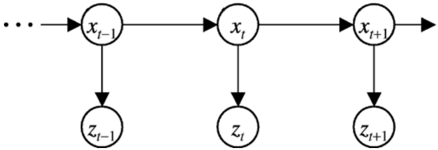
- Remarks:
  - $V_t(i)$ is the optimal cost of getting from node i to node τ in $T - t$ steps
  - $V_0(s) = J^{Q^*}$
  - The algorithm can be terminated early if $V_t(i) = V_{t+1}(i)$, $\forall i \in \mathcal{V} \setminus \{\tau\}$
  - The SP problem is symmetric: an optimal path from s to τ is also a shortest path from τ to s, where all arc directions are flipped. This view leads to a "forward Dynamic Programming" algorithm.
  - There is no analog of forward DP for stochastic problems!

- **Forward DP Algorithm**: $V_t^F(j)$ is the optimal **cost-to-arrive** to node j from node s in t moves. Starting with $V_0^F(s) = 0$, iterate:

$$
\begin{aligned}
V_1^F(j) &= c_{s,j}, \quad \forall j \in \mathcal{V} \setminus \{s\} \\
V_t^F(j) &= \min_{i \in \mathcal{V}\setminus\{s\}} \left(c_{i,j} + V_{t-1}^F(i)\right), \quad j \in \mathcal{V} \setminus \{s\}, t = 2, \ldots, T
\end{aligned}
$$

Hidden Markov Models and the Viterbi Algorithm

- Remember the HMM model from ECE-276A:
- States: $x_t \in \mathcal{X} := \{1, \ldots, N\}$
- Prior: $p_{0|0} \in [0,1]^N$ with $p_{0|0}(i) := \mathbb{P}(x_0 = i)$
- Motion model: $P \in \mathbb{R}^{N \times N}$ with $P(i,j) = \mathbb{P}(x_{t+1} = i \mid x_t = j)$
- Observations: $z_t \in \mathcal{Z} := \{1, \ldots, M\}$
- Observation model: $O \in \mathbb{R}^{M \times N}$ with $O(i,j) = \mathbb{P}(z_t = i \mid x_t = j)$



- One of the three basic HMM problems concerns the most likely sequence of states $x_{0:T}$:
  - Given an observation sequence $z_{0:T}$ and model parameters $(p_{0|0}, P, O)$, how do we choose a corresponding state sequence $x_{0:T}$ which best "explains" the observations?

Viterbi Decoding

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

Likelihood of the observed sequence with the most likely state assignment up to $t-1$

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

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

- **Initialize**:
$$\delta_0(i) = p(z_0 \mid x_0 = i)p(x_0 = i) = O(z_0, i)p_{0|0}(i)$$
$$\psi_0(i) = 0$$
- **Forward Pass** for $t = 1, \dots, T$

$$\delta_t(i) = \max_j p(z_t \mid x_t = i)p_a(x_t = i \mid x_{t-1} = j)\delta_{t-1}(j) = \max_j O(z_t, i)P(i,j)\delta_{t-1}(j)$$

$$\psi_t(i) = \arg\max_j p(z_t \mid x_t = i)p_a(x_t = i \mid x_{t-1} = j)\delta_{t-1}(j) = \arg\max_j O(z_t, i)P(i,j)\delta_{t-1}(j)$$

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

- **Backward Pass** for $t = T-1, \dots, 0$: $\boxed{x^*_t = \psi_{t+1}(x^*_{t+1})}$

Viterbi Decoding

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

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

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

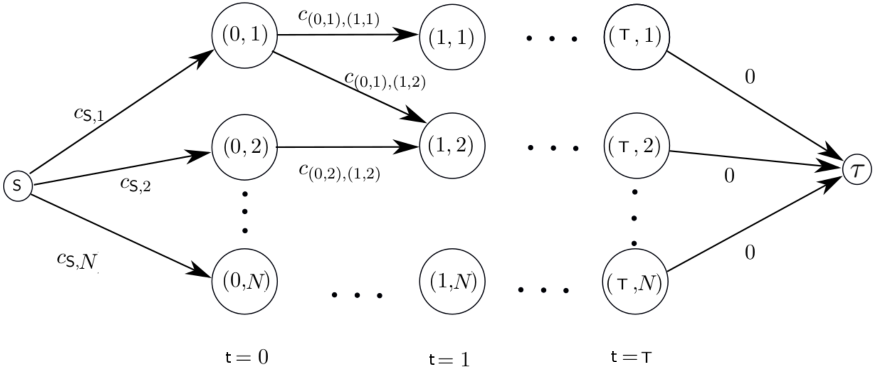
$$\max_{x_{0:T}} \left(c_{s,(0,x_0)} + \sum_{t=1}^{T} c_{(t-1,x_{t-1}),(t,x_t)} \right)$$

where:

$$c_{s,(0,x_0)} := -\log\left(p_{0|0}(x_0) O(z_0, x_0)\right)$$
$$c_{(t-1,x_{t-1}),(t,x_t)} := -\log\left(P(x_t, x_{t-1}) O(z_t, x_t)\right)$$

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** d_i: keeps (an estimate of) the lowest cost from s to each visited node $i \in \mathcal{V}$
  - Each time d_i is reduced, the labels d_j of the **children** of i can be corrected: $d_j = d_i + c_{ij}$
  - **OPEN**: set of nodes that can potentially be part of the shortest path to τ

Label Correcting Algorithm

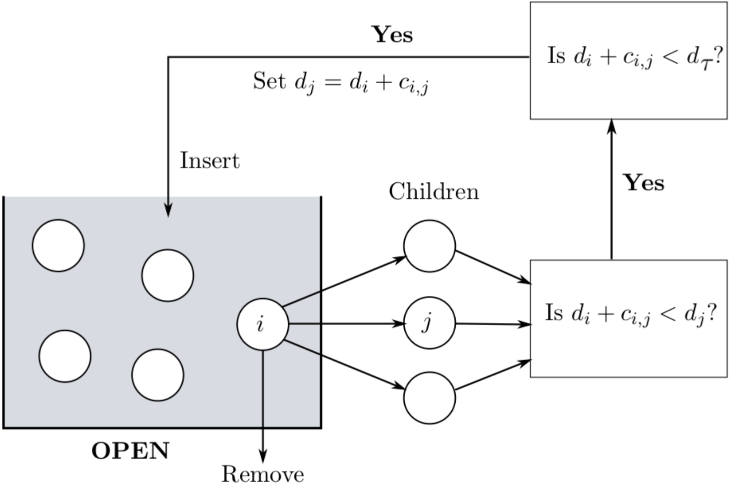
Algorithm 1 Label Correcting Algorithm

```
OPEN ← {s}, d_s = 0, d_i = ∞ for all i ∈ 𝒱 \ {s}
while OPEN is not empty do
    Remove i from OPEN
    for j ∈ Children(i) do
        if (d_i + c_ij) < d_j and (d_i + c_ij) < d_τ then
            d_j ← (d_i + c_ij)
            Parent(j) ← i
            if j ≠ τ then
                OPEN ← OPEN ∪{j}
```

Theorem

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

Label Correcting Algorithm



Label Correcting Algorithm Proof

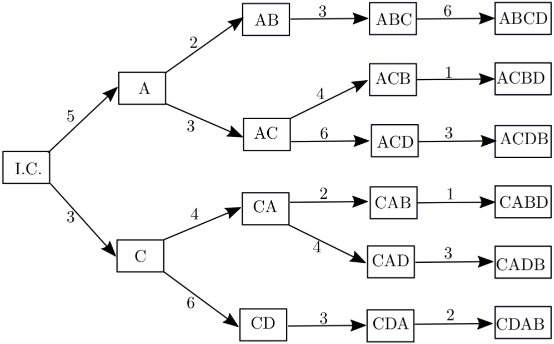
1. **Claim**: The LC algorithm terminates in a finite number of steps
   - Each time a node j enters OPEN, its label is decreased and becomes equal to the length of some path from s to j.
   - The number of distinct paths from s to j whose length is smaller than any given number is finite (**no negative 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 $d_\tau = \infty$ 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 d_τ is never updated and remains ∞

Label Correcting Algorithm Proof

3. **Claim**: The LC algorithm terminates with $d_\tau = J^{Q^*}$ 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 $d_\tau > J^{Q^*}$ (proof by contradiction)
   - Since d_τ only decreases in the algorithm and every cost is nonnegative, $d_\tau > J^{Q^*_m}$ for all $m = 2, \ldots, q-2$
   - Thus, i_{q-2} does not enter OPEN with $d_{i_{q-2}} = J^{Q^*_{q-2}}$ since if it did, then the next time i_{q-2} is removed from OPEN, d_τ would be updated to J_{Q^*}
   - Similarly, i_{q-3} will not enter OPEN with $d_{i_{q-3}} = J^{Q^*_{q-3}}$. Continuing this way, i_1 will not enter open with $d_{i_1} = J^{Q^*_1} = c_{s,i_1}$ but this happens at the first iteration of the algorithm, which is a contradiction!

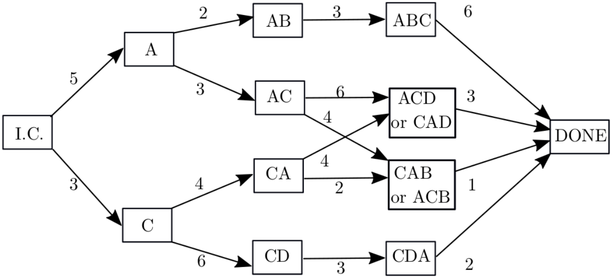
Example: Deterministic Scheduling Problem

- 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

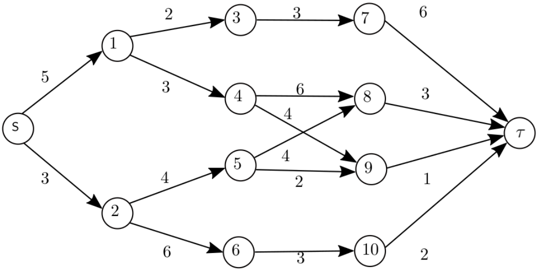


- This results in a DFS problem with $T = 4$, $\mathcal{X}_0 = \{\text{I.C.}\}$, $\mathcal{X}_1 = \{\text{A}, \text{C}\}$, $\mathcal{X}_2 = \{\text{AB}, \text{AC}, \text{CA}, \text{CD}\}$, $\mathcal{X}_3 = \{\text{ABC}, \text{ACD or CAD}, \text{CAB or ACB}, \text{CDA}\}$, $\mathcal{X}_T = \{DONE\}$

- We can map the DFS problem to a SP 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	d_S	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_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 $i^* = \arg\min_{j \in OPEN} d_j$ 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:

$$\text{from} \quad d_i + c_{ij} < d_\tau \quad \text{to} \quad d_i + c_{ij} + h_j < d_\tau$$

  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!