ECE276B: Planning & Learning in Robotics
Lecture 5: Deterministic Shortest Path

Nikolay Atanasov

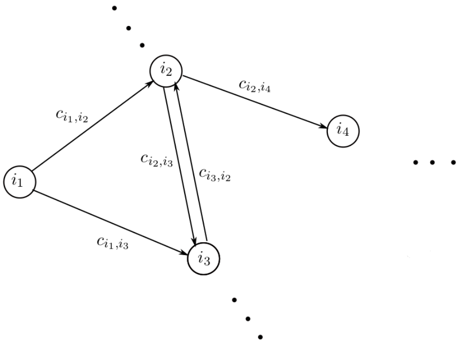
natanasov@ucsd.edu

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Outline

Deterministic Shortest Path (DSP) Problem

- Consider a graph with vertex set $\mathcal{V}$, edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and edge weights $\mathcal{C} := \{c_{ij} \in \mathbb{R} \cup \{\infty\} \mid (i, j) \in \mathcal{E}\}$ where c_{ij} denotes the cost of transition from vertex i to vertex j



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

Deterministic Shortest Path (DSP) Problem

- **Path**: a sequence $i_{1:q} := (i_1, i_2, \ldots, i_q)$ of nodes $i_k \in \mathcal{V}$
- **Path length**: sum of edge weights along the path: $J^{i_{1:q}} = \sum_{k=1}^{q-1} c_{i_k, i_{k+1}}$
- **All paths from $s \in \mathcal{V}$ to $\tau \in \mathcal{V}$**: $\mathcal{P}_{s,\tau} := \{i_{1:q} \mid i_k \in \mathcal{V}, i_1 = s, i_q = \tau\}$
- **Objective**: find a path that has the min length from node s to node τ:

$$\mathbf{dist}(s, \tau) = \min_{i_{1:q} \in \mathcal{P}_{s,\tau}} J^{i_{1:q}} \qquad\qquad i^*_{1:q} \in \arg\min_{i_{1:q} \in \mathcal{P}_{s,\tau}} J^{i_{1:q}}$$

- **Assumption**: There are no negative cycles in the graph, i.e., $J^{i_{1:q}} \geq 0$, for all $i_{1:q} \in \mathcal{P}_{i,i}$ and all $i \in \mathcal{V}$
- Solving DSP problems:
  - The finite-state DSP problem is equivalent to a finite-horizon finite-state deterministic optimal control (DOC) problem
  - Apply dynamic programming or label correcting (variant of a “forward” DPA) to the equivalent DOC problem

Deterministic Optimal Control (DOC) Problem

- **DOC Problem**:
  - optimal control problem with no disturbances, $\mathbf{w}_t \equiv 0$
  - closed-loop control does not offer any advantage over open-loop control
- Given $\mathbf{x}_0 \in \mathcal{X}$, construct an optimal control sequence $\mathbf{u}_{0:T-1}$ such that:

$$
\begin{aligned}
\min_{\mathbf{u}_{0:T-1}} \quad & \mathfrak{q}(\mathbf{x}_T) + \sum_{t=0}^{T-1} \ell(\mathbf{x}_t, \mathbf{u}_t) \\
\text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \; t = 0, \ldots, T-1 \\
& \mathbf{x}_t \in \mathcal{X}, \; \mathbf{u}_t \in \mathcal{U}
\end{aligned}
$$

- The DOC problem can be solved via Dynamic Programming

Equivalence of DOC and DSP Problems (DOC to DSP)

- Construct a graph representation of the DOC problem
- **Start node**: $s := (0, \mathbf{x}_0)$ given state $\mathbf{x}_0 \in \mathcal{X}$ at time 0
- **Vertex set**: represent every state $\mathbf{x} \in \mathcal{X}$ at time t by node $i := (t, \mathbf{x})$:

$$\mathcal{V} := \{s\} \cup \left(\bigcup_{t=1}^{T} \{(t, \mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\} \right) \cup \{\tau\}$$

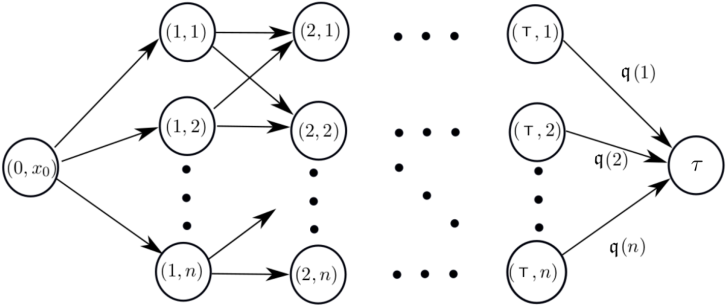
- **End node**: an artificial node τ with arc length $c_{i,\tau}$ from node $i = (t, \mathbf{x})$ to τ equal to the terminal cost $\mathfrak{q}(\mathbf{x})$ of the DOC problem

Equivalence of DOC and DSP Problems (DOC to DSP)

- The edge weight between two nodes $i = (t, \mathbf{x})$ and $j = (t', \mathbf{x}')$ is finite, $c_{ij} < \infty$, only if $t' = t + 1$ and $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$ for some $\mathbf{u} \in \mathcal{U}$.

- The edge weight between two nodes $i = (t, \mathbf{x})$ and $j = (t + 1, \mathbf{x}')$ is the smallest stage cost between $\mathbf{x}$ and $\mathbf{x}'$:

$$\mathcal{C} := \left\{ c_{(t,\mathbf{x}),(t+1,\mathbf{x}')} = \min_{\substack{\mathbf{u} \in \mathcal{U}(\mathbf{x}) \\ \text{s.t. } \mathbf{x}' = f(\mathbf{x},\mathbf{u})}} \ell(\mathbf{x}, \mathbf{u}) \right\} \bigcup \left\{ c_{(T,\mathbf{x}),\tau} = \mathfrak{q}(\mathbf{x}) \right\}$$



Equivalence of DOC and DSP Problems (DSP to DOC)

- Consider a DSP problem with vertices $\mathcal{V}$, edges $\mathcal{E}$, edge weights $\mathcal{C}$, start node $s \in \mathcal{V}$ and terminal node $\tau \in \mathcal{V}$
- **No negative cycles assumption**: an optimal path need not have more than $|\mathcal{V}|$ elements
- We can formulate the DSP problem as DOC with $T := |\mathcal{V}| - 1$ stages:
  - **State space** $\mathcal{X} = \mathcal{V}$ and **control space**: $\mathcal{U} = \mathcal{V}$
  - **Motion model**: $x_{t+1} = f(x_t, u_t) := \begin{cases} x_t & \text{if } x_t = \tau \\ u_t & \text{otherwise} \end{cases}$
  - **Stage cost** and **terminal cost**:

$$\ell(x, u) := \begin{cases} 0 & \text{if } x = \tau \\ c_{x,u} & \text{otherwise} \end{cases} \qquad \mathfrak{q}(x) := \begin{cases} 0 & \text{if } x = \tau \\ \infty & \text{otherwise} \end{cases}$$

Dynamic Programming Applied to DSP

- Due to the DOC equivalence, a DSP problem can be solved via dynamic programming

---

Algorithm Deterministic Shortest Path via Dynamic Programming

---

Input: vertices $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs c_{ij} for $i, j \in \mathcal{V}$
$T = |\mathcal{V}| - 1$
$V_T(\tau) = V_{T-1}(\tau) = \ldots = V_0(\tau) = 0$
$V_T(i) = \infty, \quad \forall i \in \mathcal{V} \setminus \{\tau\}$
$V_{T-1}(i) = c_{i,\tau}, \quad \forall i \in \mathcal{V} \setminus \{\tau\}$
$\pi_{T-1}(i) = \tau, \quad \forall i \in \mathcal{V} \setminus \{\tau\}$
for $t = (T-2), \ldots, 0$ **do**
    $Q_t(i,j) = c_{i,j} + V_{t+1}(j), \quad \forall i \in \mathcal{V} \setminus \{\tau\}, j \in \mathcal{V}$
    $V_t(i) = \min_{j \in \mathcal{V}} Q_t(i,j), \quad \forall i \in \mathcal{V} \setminus \{\tau\}$
    $\pi_t(i) \in \arg\min_{j \in \mathcal{V}} Q_t(i,j), \quad \forall i \in \mathcal{V} \setminus \{\tau\}$
    if $V_t(i) = V_{t+1}(i), \forall i \in \mathcal{V} \setminus \{\tau\}$ **then**
        break

- $V_t(i)$ is the **optimal cost-to-go** from node i to node τ in at most $T - t$ steps
- Upon termination, $V_0(s) = J^{i^*_{1:q}} = \textbf{dist}(s, \tau)$
- The algorithm can be terminated early if $V_t(i) = V_{t+1}(i), \forall i \in \mathcal{V} \setminus \{\tau\}$

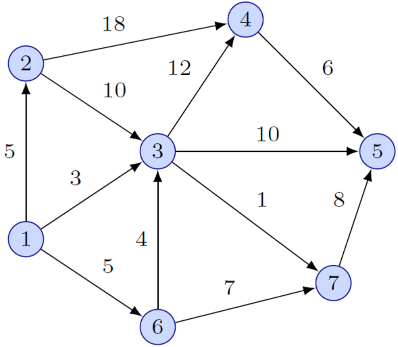
Forward Dynamic Programming Applied to DSP

- The DSP problem is symmetric: a shortest path from s to τ is also a shortest path from τ to s with all arc directions 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 at most t steps

Algorithm Deterministic Shortest Path via Forward Dynamic Programming

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

Example: Forward DP Algorithm



- $s = 1$ 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 $V_6^F(i)$

Outline

Deterministic Shortest Path

Label Correcting Algorithm

Label Correcting Methods for the DSP Problem

- The (backward) Dynamic Programming algorithm applied to the DSP problem computes the shortest paths from *all* nodes to the goal τ
- The forward Dynamic Programming algorithm computes the shortest paths from the start s to *all* nodes
- Often many nodes are not part of the shortest path from s to τ
- **Label correcting (LC)** algorithms for the DSP problem do not necessarily visit every node of the graph
- LC algorithms prioritize visited nodes i using the **cost-to-arrive** $V_t^F(i)$
- **Key ideas** in LC algorithms:
  - **Label** g_i: estimate of optimal cost-to-arrive from s to each visited $i \in \mathcal{V}$
  - **Label correction**: each time g_i is reduced, the labels g_j of the **children** of i are corrected: $g_j = g_i + c_{ij}$
  - **OPEN List**: set of nodes that can potentially be part of the shortest path to τ

Label Correcting Algorithm

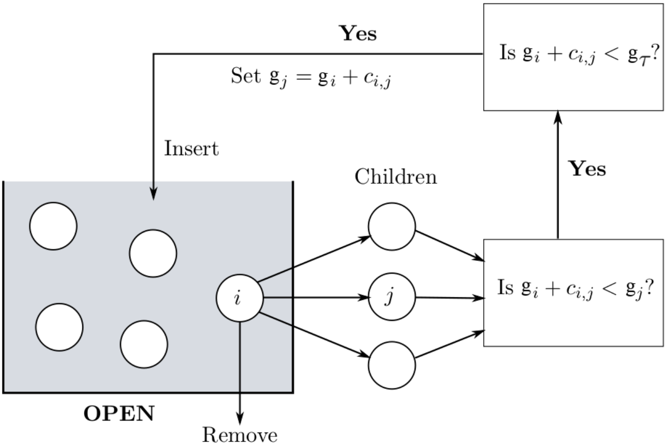
Algorithm Label Correcting Algorithm

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** ▷ Only when $c_{ij} \geq 0$ for all $i, j \in \mathcal{V}$
      $g_j = g_i + c_{ij}$
      Parent$(j) = i$
      if $j \neq \tau$ **then**
        OPEN $=$ OPEN $\cup \{j\}$

Theorem

Consider a finite-state deterministic shortest path problem. If there exists at least one finite cost path from s to τ, then the Label Correcting algorithm terminates with $g_\tau = \textbf{dist}(s, \tau)$, the shortest path length from s to τ. Otherwise, the algorithm terminates with $g_\tau = \infty$.

Label Correcting Algorithm



Label Correcting Algorithm Proof

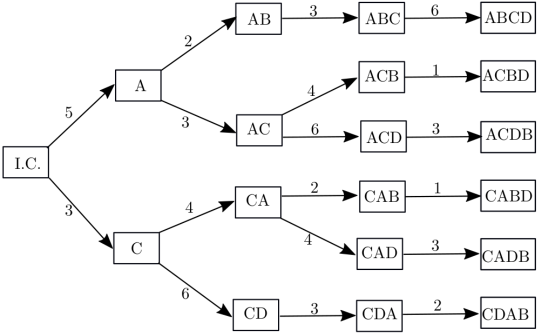
1. **Claim**: The LC algorithm terminates in a finite number of steps
   - Each time a node j enters OPEN, its label is decreased and becomes equal to the length of some path from s to j.
   - The number of distinct paths from s to j whose length is smaller than any given number is finite (**no negative 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**: Assume $c_{ij} \geq 0$ (special case). The LC algorithm terminates with $g_\tau = \textbf{dist}(s,\tau)$ if there is at least one finite cost path from s to τ.
   - Let $i^*_{1:q} \in \mathcal{P}_{s,\tau}$ be a shortest path from s to τ with $i^*_1 = s$, $i^*_q = \tau$, and length $J^{i^*_{1:q}} = \textbf{dist}(s,\tau)$.
   - By the principle of optimality, $i^*_{1:m}$ is a shortest path from s to i^*_m with length $J^{i^*_{1:m}} = \textbf{dist}(s, i^*_m)$ for any $m = 1, \ldots, q-1$.
   - Suppose that $g_\tau > J^{i^*_{1:q}} = \textbf{dist}(s,\tau)$ (proof by contradiction).
   - Since g_τ only decreases in the algorithm and every cost is nonnegative, $g_\tau > J^{i^*_{1:m}} = \textbf{dist}(s, i^*_m)$ for all $m = 2, \ldots, q-1$.
   - Thus, i^*_{q-1} does not enter OPEN with $g_{i^*_{q-1}} = J^{i^*_{1:q-1}} = \textbf{dist}(s, i^*_{q-1})$ since if it did, then the next time i^*_{q-1} is removed from OPEN, g_τ would be updated to $J^{i^*_{1:q}} = \textbf{dist}(s, i^*_{q-1})$.
   - Similarly, i^*_{q-2} does not enter OPEN with $g_{i^*_{q-2}} = J^{i^*_{1:q-2}} = \textbf{dist}(s, i^*_{q-2})$.
   - Continuing this way, i^*_2 will not enter OPEN with $g_{i^*_2} = J^{i^*_{1:2}} = c_{s,i^*_2}$ but this happens at the first iteration of the algorithm, which is a contradiction.

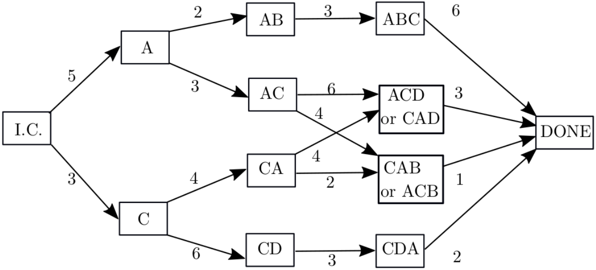
Example: Deterministic Scheduling Problem

- 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

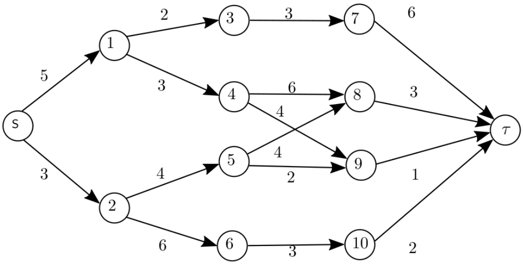


- This results in a DOC problem with $T = 4$ and $\mathcal{X} = \{$I.C., A, C, AB, AC,CA, CD, ABC, ACD or CAD, CAB or ACB, CDA, DONE$\}$

- The DOC problem can be converted into a DSP problem

Example: Deterministic Scheduling Problem

- We can map the DOC problem to a DSP problem and apply the LC algorithm



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

- Keeping track of the parents when a child node is added to OPEN, we can determine a shortest path $(s, 2, 5, 9, \tau)$ with total cost 10, which corresponds to $(C, CA, CAB, CABD)$ in the original problem

Label Correcting Algorithm Variations

- The freedom to select which node to remove from OPEN at each iteration gives rise to several different label correcting 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**: 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 $g_i \leq g_{TOP}$ 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 for special case $c_{ij} \geq 0$ in which the requirement for admission to OPEN is strengthened:

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

  where h_j is a non-negative lower bound on the optimal cost-to-go $\textbf{dist}(j, \tau)$ from node j to τ, known as a **heuristic function**

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

- A heuristic function is constructed using special knowledge about the problem. The more accurately h_j estimates the optimal cost-to-go $\textbf{dist}(j, \tau)$ from j to τ, the more efficient the A* algorithm becomes.