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 $\tau$

- It turns out that the DSP problem is equivalent to a finite-horizon deterministic finite-state (DFS) optimal control problem.
The Deterministic Shortest Path (DSP) Problem

- **Path**: a sequence $i_{1:q} := (i_1, i_2, \ldots, i_q)$ of nodes $i_k \in \mathcal{V}$.

- **All paths from $s \in \mathcal{V}$ to $\tau \in \mathcal{V}$**: $\Pi_{s, \tau} := \{i_{1:q} \mid i_k \in \mathcal{V}, i_1 = s, i_q = \tau\}$.

- **Path length**: sum of the arc lengths over the path: $J_{i_{1:q}} = \sum_{k=1}^{q-1} c_{i_k, i_{k+1}}$.

- **Objective**: find a path $i_{1:q}^* = \arg\min_{i_{1:q} \in \Pi_{s, \tau}} J_{i_{1:q}}$ 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 $i_{1:q} \in \Pi_{i,i}$, $J_{i_{1:q}} \geq 0$, i.e., there are no negative cycles in the graph.

- **Solving DSP problems**: map to a deterministic finite-state problem and apply the (backward) DPA, label correcting methods (variants of a “forward” DPA).
Deterministic Finite State (DFS) Optimal Control Problem

- **DFS**: the optimal control problem with no disturbances, $w_t \equiv 0$, and finite state space, $|\mathcal{X}| < \infty$

- Deterministic problem: closed-loop control does not offer any advantage over open-loop control

- Given $x_0 \in \mathcal{X}$, construct an optimal control sequence $u_0: T-1$ such that:

$$
\min_{u_0: T-1} \quad q(x_T) + \sum_{t=0}^{T-1} \ell(x_t, u_t)
\quad \text{s.t. } x_{t+1} = f(x_t, u_t), \ t = 0, \ldots, T - 1
\quad x_t \in \mathcal{X}, \ u_t \in \mathcal{U}(x_t),
$$

- The DFS problem can be solved via the Dynamic Programming algorithm
We can construct a graph representation of the DFS problem.

**Start node:** $s := (0, x_0)$ given state $x_0 \in \mathcal{X}$ at time 0

Every state $x \in \mathcal{X}$ at time $t$ is represented by a node $i := (t, x)$:

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

**End node:** an artificial node $\tau$ with arc length $c_{i,\tau}$ from node $i = (t, x)$ to $\tau$ equal to the terminal cost $q(x)$ of the DFS.
Equivalence of DFS and DSP Problems (DFS to DSP)

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

- The arc length between two nodes $i = (t, x)$ and $j = (t + 1, x')$ is the smallest stage cost between $x$ and $x'$:

$$
C := \left\{ ((t, x), (t + 1, x'), c) \middle| c = \min_{u \in \mathcal{U}(x)} \ell(x, u) \right\} \cup \left\{ (((T, x), \tau, q(x))) \right\}
$$
Consider a DSP problem with vertex space $\mathcal{V}$, weighted edge space $\mathcal{C}$, start node $s \in \mathcal{V}$ and terminal node $\tau \in \mathcal{V}$

**No negative cycles assumption:** 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:** $q(\tau) := 0$ and $\ell(x_t, u_t) = c_{x_t, u_t}$ for $t = 0, \ldots, T - 1$
Dynamic Programming Applied to DSP

- Due to the equivalence, a DSP problem can be solved via the DPA
- $V_t(i)$ is the optimal cost of getting from node $i$ to node $\tau$ in $T - t$ steps
- Upon termination, $V_0(s) = J_{1:q}^{i*}$
- 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

1: **Input**: node set $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs $c_{ij}$ for $i, j \in \mathcal{V}$
2: $T = |\mathcal{V}| - 1$
3: $V_T(\tau) = 0$
4: $V_{T-1}(i) = c_{i,\tau}$, $\forall i \in \mathcal{V} \setminus \{\tau\}$
5: $\pi_{T-1}(i) = \tau$, $\forall i \in \mathcal{V} \setminus \{\tau\}$
6: **for** $t = (T - 2), \ldots, 0$ **do**
7: $Q_t(i, j) = c_{i,j} + V_{t+1}(j)$, $\forall i, j \in \mathcal{V} \setminus \{\tau\}$
8: $V_t(i) = \min_{j \in \mathcal{V} \setminus \{\tau\}} Q_t(i, j)$, $\forall i \in \mathcal{V} \setminus \{\tau\}$
9: $\pi_t(i) = \arg\min_{j \in \mathcal{V} \setminus \{\tau\}} Q_t(i, j)$, $\forall i \in \mathcal{V} \setminus \{\tau\}$
10: **if** $V_t(i) = V_{t+1}(i)$, $\forall i \in \mathcal{V} \setminus \{\tau\}$ **then**
11: break
Forward DP Algorithm

- The DSP problem is symmetric: a shortest path from $s$ to $\tau$ is also a shortest path from $\tau$ to $s$, where all arc directions are flipped.

- This view leads to a **forward Dynamic Programming algorithm**.

- $V^F_t(j)$ is the **optimal cost-to-arrive** to node $j$ from node $s$ in $t$ moves

---

**Algorithm 2** Deterministic Shortest Path via Forward Dynamic Programming

1: **Input**: node set $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs $c_{ij}$ for $i,j \in \mathcal{V}$
2: $T = |\mathcal{V}| - 1$
3: $V^F_0(s) = 0$
4: $V^F_1(j) = c_{s,j}$, $\forall j \in \mathcal{V} \setminus \{s\}$
5: for $t = 2, \ldots, T$ do
6: $V^F_t(j) = \min_{i \in \mathcal{V} \setminus \{s\}} (c_{i,j} + V^F_{t-1}(i))$, $\forall j \in \mathcal{V} \setminus \{s\}$
7: if $V^F_t(i) = V^F_{t-1}(i)$, $\forall i \in \mathcal{V} \setminus \{s\}$ then
8: break
Example: Forward DP Algorithm

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_0^F$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$V_1^F$</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$V_2^F$</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>13</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$V_3^F$</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>12</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$V_4^F$</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>12</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- 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 $V_6^F(i)$
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 $\tau$.

- **Label correcting (LC) algorithms** for the DSP problem do not necessarily visit every node of the graph.

- LC algorithms prioritize the visited nodes using the cost-to-arrive values.

- **Key Ideas:**
  - **Label** $g_i$: 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 $\tau$. 


Label Correcting Algorithm

Algorithm 3 Label Correcting Algorithm

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

Theorem

If there exists at least one finite cost path from $s$ to $\tau$, then the Label Correcting (LC) algorithm terminates with $g_\tau = J^{i_1:q}$ (the shortest path from $s$ to $\tau$). Otherwise, the LC algorithm terminates with $g_\tau = \infty$. 
Label Correcting Algorithm

Is $g_i + c_{i,j} < g_j$?

Is $g_i + c_{i,j} < g_T$?

Set $g_j = g_i + c_{i,j}$

Yes

Insert

Open

Remove

Children
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 $\tau$
   ▶ A node $i \in V$ 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 $\tau$, then for any node $i$ in OPEN $c_{i,\tau} = \infty$; otherwise there would be a finite cost path from $s$ to $\tau$
   ▶ Since $c_{i,\tau} = \infty$ for every $i$ in OPEN, line 5 ensures that $g_\tau$ is never updated and remains $\infty$
3. **Claim:** The LC algorithm terminates with $g_{\tau} = J^{i_1^*:q}$ if there is at least one finite cost path from $s$ to $\tau$

- Let $i_1^*:q \in I^s,\tau$ be a shortest path from $s$ to $\tau$ with $i_1^* = s$, $i_q^* = \tau$, and length $J^{i_1^*:q}$

- By the principle of optimality, $i_1^*:m$ is a shortest path from $s$ to $i_m^*$ with length $J^{i_1^*:m}$ for any $m = 1, \ldots, q - 1$

- Suppose that $g_{\tau} > J^{i_1^*:q}$ (proof by contradiction)

- Since $g_{\tau}$ only decreases in the algorithm and every cost is nonnegative, $g_{\tau} > J^{i_1^*:m}$ for all $m = 2, \ldots, q - 1$

- Thus, $i_{q-1}^*$ does not enter OPEN with $g_{i_{q-1}^*} = J^{i_1^*:q-1}$ since if it did, then the next time $i_{q-1}^*$ is removed from OPEN, $g_{\tau}$ would be updated to $J^{i_1^*:q}$

- Similarly, $i_{q-2}^*$ will not enter OPEN with $g_{i_{q-2}^*} = J^{i_1^*:q-2}$. 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 DFS problem with $T = 4$, $X_0 = \{\text{I.C.}\}$, $X_1 = \{A, C\}$, $X_2 = \{AB, AC, CA, CD\}$, $X_3 = \{ABC, ACD \text{ or CAD}, CAB \text{ or ACB}, CDA\}$, $X_T = \{DONE\}$.

- We can map the DFS problem to a DSP problem.
Example: Deterministic Scheduling Problem

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

- Keeping track of the parents when a child node is added to 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.
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 \( i^* = \arg \min_{j \in \text{OPEN}} g_j \) 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 \( g_i \leq g_{\text{TOP}} \) 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:

\[
g_i + c_{ij} < g_\tau \quad \text{to} \quad g_i + c_{ij} + h_j < g_\tau
\]

where \( h_j \) is a positive lower bound on the optimal cost to get from node \( j \) to \( \tau \), 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 \( \tau \), the more efficient the A* algorithm becomes!