

CIE 6022 Dynamic Programming

Binghao He

March 1, 2021

Contents

Lecture 1. Introduction and Basic DP Algorithm	2
Basic Strucutre of Stochastic DP	2
Example 1. Inventory Control	4
Example 2. Scheduling	4
Example 3. Two-game Chess	5
Principle of Optimality	5
Demonstration for the Principle of Optimality (Inventory Control)	5
DP Algorithm	6
State Augmentation	6
Lecture 2. Deterministic Discrete-Time System and Shortest Path Problems	8
Deterministic Finite-State DP Problems	8
Backward and Forward DP Algorithm	8
Note on Forward DP algorithms	9
Generic Shortest Path Problems	9
Example for Generic Shortest Path Problem	10
Critical Path Analysis	10
Estimation/Hidden Markov Models	10
Viterbi Algorithm	11
Advantage of Forward Alogrithm (for Viterbi Algorithm)	12
General Shortest Path Algorithm	12
Traveling Salesman Problem	12
Knapsack Problem	13
Label Correcting Methods	14
Specific Label Correcting Methods	14
Label Correcting Variations	14
Appendix.1 Value of Information (Two-game Chess)	16

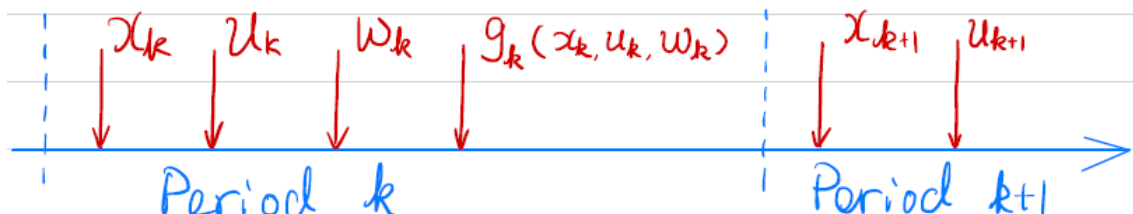
Lecture 1. Introduction and Basic DP Algorithm

Basic Structure of Stochastic DP

- 2 principle features of DP
 - Discrete-time dynamic system
 - Cost function is additive over time
- 6 key elements of DP formulation
 - **Stage:** k
 - * Finite horizon. Total number of horizon: N .
 - **State:** x_k
 - **Control:** $u_k \in U_k(x_k)$ (or called **Decision, Action**)
 - **Disturbance:** w_k
 - **"Memoryless" System Dynamics:** $x_{k+1} = f_k(x_k, u_k, w_k)$
 - * Alternative system equation: $P(x_{k+1}|x_k, u_k)$
 - **"Additive" Stage Cost:** $g_k(x_k, u_k, w_k)$
 - * Terminal Stage Cost: $g_N(x_N)$
 - * Total cost: $\mathbb{E}_{w_0, w_{k-1}} \{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\}$
- Additional Assumptions
 - The feasible set of u_k , $U_k(x_k)$, depends at most x_k and not on prior x or u .
 - The probability distribution of w_k may depends on x_k, u_k but not past values w_0, \dots, w_{k-1} .
 - * $P(w_k|x_k, u_k)$

This two assumptions ensures the optimization of u_k only requires the information of x_k and u_k , which is "Memoryless". Past values of x or w should be useless for future optimization.

- Sequence of events envisioned in period k :



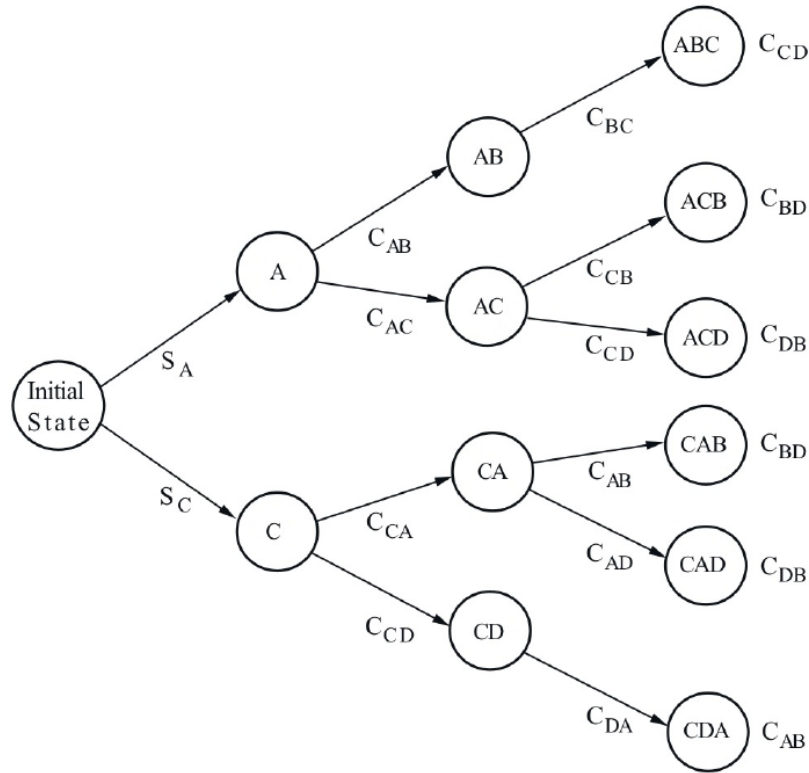
- x_k occurs according to system dynamics
 - * $x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1})$
- u_k is selected with knowledge of x_k
 - * $u_k \in U_k(x_k)$
- w_k is random and generated according to a distribution
 - * $w_k \sim P(w_k | x_k, u_k)$
- $g_k(x_k, u_k, w_k)$ is incurred and added to previous costs
- **Basic problem**
 - **Policies** $\pi = \{\mu_0, \dots, \mu_{N-1}\}$
 - * μ_k maps states x_k into controls $u_k = \mu_k(x_k)$
 - * $\mu_k(x_k) \in U_k(x_k)$ for all x_k
 - **Expected cost** of policy π starting at x_0 is
 - * $J_\pi(x_0) = \mathbb{E}\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\}$
 - **Optimal cost function**
 - * $J^*(x_0) = \min_\pi J_\pi(x_0)$
 - **Optimal policy** π^*
 - * $J_{\pi^*}(x_0) = J^*(x_0)$
- **2 aims**
 - **Optimal policy**
 - * $u_k = \mu_k^*(x_k)$ for all $k = 1, \dots, N-1$
 - * $\pi^* = (\mu_0^*, \mu_1^*, \dots, \mu_{N-1}^*)$
 - **Value function**
 - * $J_N(x_N) = g_N(x_N)$
 - * $J_k(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}_{w_k}\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\}, \forall k = 0, 1, \dots, N-1$

Example 1. Inventory Control

- **Stage:** Ordering period k
- **State:** Inventory level at period k , x_k
- **Control:** Stock ordered at period k , $u_k \geq 0$
- **Disturbance:** Demand at period k , w_k
- **System Dynamics:** $x_{k+1} = f_k(x_k, u_k, w_k) = x_k + u_k - w_k$
- **Stage Cost:** $g_k(x_k, u_k, w_k) = \underbrace{k \cdot \mathbb{1}_{[u_k > 0]}}_{\text{Set up}} + \underbrace{c \cdot u_k}_{\text{Ordering fee}} + \underbrace{h(x_k + u_k - w_k)^+}_{\text{Holding cost}} + \underbrace{p(x_k + u_k - w_k)^-}_{\text{Shortage penalty}}$
- **Terminal Cost:** $g_N(x_N) = 0$

Example 2. Scheduling

- Find the optimal sequence of operations A, B, C, D
- A must precede B , C must precede D
- Given startup cost S_A, S_C and setup transition cost C_{nm} from operation m to n
- This is a **Deterministic Finite-State Problem**.



Example 3. Two-game Chess

- Find two-game chess match strategy
- **State:** Match score
- **Control:** Timid play *or* Bold play
- **System Dynamics:**
 - Timid play: draws with $P_d > 0$ and loses with $1 - P_d$
 - Bold play: wins with $P_w \leq \frac{1}{2}$ and loses with $1 - P_w$
- **Value of information** = $J_{\text{closed-loop}}^* - J_{\text{open-loop}}^*$
 - Open-loop policy: play always Bold
 - Close-loop policy: Play Timid if and only if you are ahead

See the example in Appendix.1

Principle of Optimality

- Consider the ”tail subproblem”. Minimize the ”cost-to-go” from i to N
 - $\mathbb{E}\{g_N(x_N) + \sum_{k=1}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\}$
- Consider the ”tail policy”
 - $\{\mu_i^*, \mu_{i+1}^*, \dots, \mu_{N-1}^*\}$
- **Principle of optimality:** The tail policy is optimal for the tail subproblem.
- DP first solve ALL tail subproblems of final stage
- At the generic step, DP solves ALL tail subproblems of a given time length, using the solution of the subproblems of shorter length

Demonstration for the Principle of Optimality (Inventory Control)

- $J_N(x_N) = 0$
- Tail subproblem of length 1 ($u_{N-1} = \mu_{N-1}^*(x_{N-1})$)
$$\begin{aligned} J_{N-1}(x_{N-1}) &= \min_{u_{N-1} \geq 0} \mathbb{E}_{w_{N-1}} \{cu_{N-1} + r(x_{N-1}) + u_{N-1} - w_{N-1} + J_N(x_{N-1} + u_{N-1} - w_{N-1})\} \\ &= \min_{u_{N-1} \geq 0} \mathbb{E}_{w_{N-1}} \{cu_{N-1} + r(x_{N-1}) + u_{N-1} - w_{N-1}\} \end{aligned}$$
- Tail subproblem of length $N - k$ ($u_k = \mu_k^*(x_k)$)
$$J_k(x_k) = \min_{u_k \geq 0} \mathbb{E}_{w_k} \{cu_k + r(x_k) + u_k - w_k + J_{k+1}(x_k + u_k - w_k)\}$$
- $J_0(x_0)$ is the optimal cost of initial state x_0

DP Algorithm

Original Problem:

$$\begin{aligned} J^*(x_0) = \min_{x \in \pi} J_\pi(x_0) &= \min_{x \in \pi} \mathbb{E} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\} \\ \text{s.t. } x_{k+1} &= f_k(x_k, \mu_k(x_k), w_k) \\ \text{where } \pi &= \{\mu_0, \mu_1, \dots, \mu_{N-1}\} \end{aligned}$$

DP algorithm:

- Start with $J_N(x_N) = g_N(x_N)$ and go backwards using

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}_{w_k} \{ g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \} \quad , k = 0, 1, \dots, N-1$$

This means that $u_k = \mu_k^*(x_k)$.

- Then $J_0(x_0)$, generated at the last setp, is equal to the optimal cost $J^*(x_0)$.
Also, the policy $\pi^* = \{\mu_0^*, \dots, \mu_{N-1}^*\}$.

Some remark:

- Ideally, the DP algorithm can be used to obtain closed-form expressions for J_k , the cost-to-go function at k , or an optimal policy
- Even if models may rely on over-simplified assumption, they may provide valuable insights about the structures of the optimal solution of more complex models and form the basis for suboptimal control scheme
- Unfortunately, an analytical solution is not possible in many practical cases, and one has to resort to numerical execution of the DP algorithm
- DP is the only general approach for sequential optimization under uncertainty, and even when it is computationally prohibitive, it can serve as the basis for more practicval suboptimal approaches

[Example, Proof ...]

State Augmentation

- When assumptions of the basic problem are violated (e.g., disturbances are correlated, cost is nonadditive, etc) reformulate/augment the state
- DP algorithm still applies, but the problem gets bigger
- **Example:**
Time lags

$$\begin{aligned} x_{k+1} &= f_k(x_k, x_{k-1}, u_k, w_k) \\ x_1 &= f_0(x_0, u_0, w_0) \end{aligned}$$

- Introduce additional state variable $y_k = x_{k-1}$. New system takes the form

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, w_k) \\ x_k \end{pmatrix}$$

View $\tilde{x}_k = (x_k, y_k)$ as the new state.

- DP algorithm for the formulated problem:

$$J_k(x_k, x_{k-1}) = \min_{u_k \in U_k(x_k)} \mathbb{E}_{w_k} \{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, x_{k-1}, u_k, w_k), x_k)\}$$

• **Example:**

Correlated disturbances

$$w_k = \lambda w_k + \xi_k$$

- Let $y_k = w_{k-1}$

$$\tilde{x}_{k+1} = \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, u_k, \lambda y_k + \xi_k) \\ \lambda y_k + \xi_k \end{pmatrix}$$

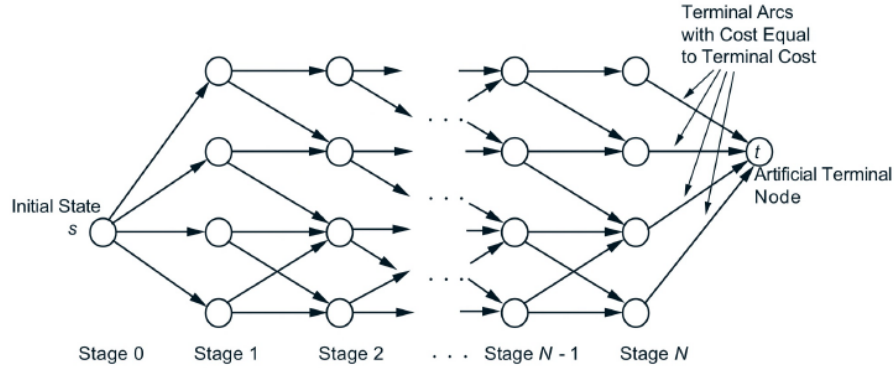
View $\tilde{x}_k = (x_k, y_k)$ as the new state.

- DP algorithm for the formulated problem:

$$J_k(x_k, y_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}_{\xi_k} \{g_k(x_k, u_k, \lambda y_k + \xi_k) + J_{k+1}(f_k(x_k, u_k, \lambda y_k + \xi_k), \lambda y_k + \xi_k)\}$$

Lecture 2. Deterministic Discrete-Time System and Shortest Path Problems

Deterministic Finite-State DP Problems



- Two conditions required for the transformation to the shortest path problem
 - Deterministic
 - Finite-state

A deterministic finite-state problem is equivalent to finding a shortest path from s (initial state) to t (terminal state)

- States: Nodes
- Controls: Arcs
- Control sequences (Open-loop): paths from initial state to terminal state
- a_{ij}^k : cost of transition from state $i \in S_k$ to state $j \in S_{k+1}$ at stage k ("length" of the arc)
- a_{it}^N : terminal cost of state $i \in S_N$
- Cost of control sequence: cost of the corresponding path ("length" of the path)

Backward and Forward DP Algorithm

- DP algorithm
 - $J_N(i) = a_{it}^N, \quad i \in S_N$
 - $J_k(i) = \min_{j \in S_{k+1}} [a_{ij}^k + J_{k+1}(j)], \quad i \in S_k, \quad k = 0, \dots, N-1$

The optimal cost is $J_0(s)$ and is equal to the length of the shortest path from s to t

- Observation: An optimal path $s \rightarrow t$ is also an optimal path $t \rightarrow s$ in a "reverse" shortest path problem where the direction of each arc is reversed and its length is left unchanged
- In this situation: forward DP algorithm = backward DP algorithm

- For the reverse problem:

$$- \tilde{J}_N(i) = a_{it}^0, \quad i \in S_1$$

$$- \tilde{J}_k(i) = \min_{j \in S_{N-k}} [a_{ij}^{N-k} + \tilde{J}_{k+1}(j)], \quad i \in S_{N-k+1}, \quad k = 1, \dots, N$$

$$\text{The optimal cost is } \tilde{J}_0(t) = \min_{i \in S_N} [a_{it}^N + \tilde{J}_1(i)]$$

- View $\tilde{J}_k(j)$ as optimal-to-arrive to state j from initial state s

Note on Forward DP algorithms

- There is no Forward DP Algorithm for **stochastic** problems
- Mathematically, for stochastic problems, we cannot restrict ourselves to open loop sequences, so the shortest path viewpoint fails
- Conceptually, in the presence of uncertainty the concept of "optimal cost-to-arrive" at a state s_k does not make sense. It may be impossible to guarantee (with prob. 1) that any given state can be reached.
- By contrast, even in stochastic problems, the concept "optimal cost-to-go" from any state s_k makes clear sense.

Generic Shortest Path Problems

- $\{1, 2, \dots, N, t\}$: Nodes of a graph (t : the destination)
- a_{ij} : cost of moving from node i to node j
- Find a shortest (minimum cost) path from each node i to node t
- Assumption: all cycles have nonnegative length. Then an optimal path need not take more than N moves.
Some arcs are allowed to have negative length
- We formulate the problem as one where we require exactly N moves but allows degenerate moves from a node i to itself with cost $a_{ii} = 0$

$$- J_k(i): \text{optimal cost of getting from } i \text{ to } t \text{ in } N - k \text{ moves}$$

$$- J_0(i): \text{cost of the optimal path from } i \text{ to } t$$

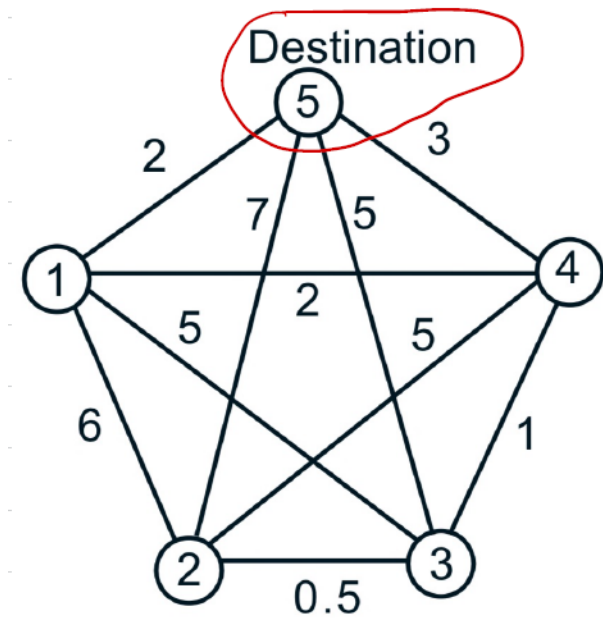
- DP algorithm:

$$- J_k(i) = \min_{j=1, \dots, N} [a_{ij} + J_{k+1}(j)], \quad k = 0, \dots, N - 2$$

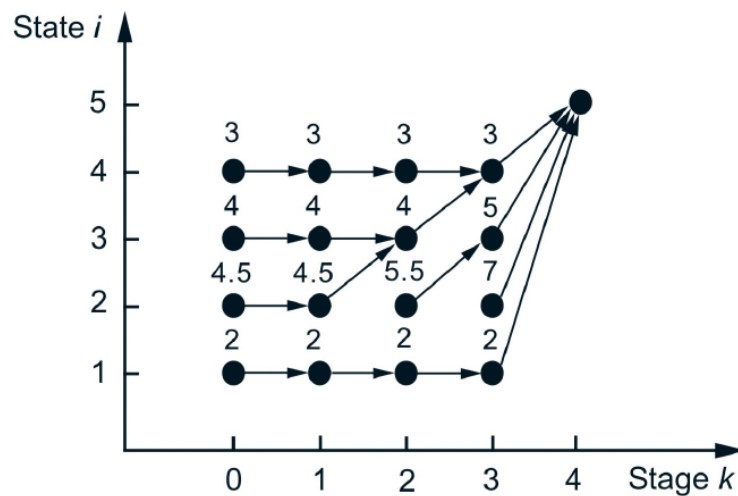
$$- J_{N-1}(i) = a_{it}, \quad i = 1, 2, \dots, N$$

Example for Generic Shortest Path Problem

At most 4 moves to the destination



Convert to equivalent staged-shortest-path-problem



Bellman-Ford Algorithm

Critical Path Analysis

[To be completed ...]

Estimation/Hidden Markov Models

- Markov chain with transition probabilities p_{ij}

- State transitions are hidden from view
- For each transition, we get an (independent) observation
- $r(z; i, j)$: Prob. the observation take value z when the state transition is from i to j
- **Trajectory estimation problem:** Given the observation sequence $Z_N = \{z_1, z_2, \dots, z_N\}$, what is the "most likely" state transition sequence $\hat{X}_N = \{\hat{x}_0, \hat{x}_1, \dots, \hat{x}_N\}$ [one that maximizes $p(X_N|Z_N)$ over all $X_N = \{x_0, x_1, \dots, x_N\}$]

Viterbi Algorithm

Assumption of independent observation: an observation only depends on its corresponding transition

- We have

$$p(X_N|Z_N) = \frac{p(X_N, Z_N)}{p(Z_N)}$$

where $p(X_N, Z_N)$ and $p(Z_N)$ are the unconditional probabilities of occurrence of (X_N, Z_N) and Z_N

- Maximizing $p(X_N|Z_N)$ is equivalent with maximizing $\ln(p(X_N, Z_N))$
- We have

$$\begin{aligned} p(X_N, Z_N) &= p(x_0, x_1, \dots, x_N, z_1, z_2, \dots, z_N) \\ &= \pi_{x_0} p(x_1, \dots, x_N, z_1, z_2, \dots, z_N | x_0) \\ &= \pi_{x_0} p(x_1, z_1 | x_0) p(x_2, \dots, x_N, z_2, \dots, z_N | x_0, x_1, z_1) \\ &= \pi_{x_0} p_{x_0 x_1} r(z_1; x_0, x_1) p(x_2, \dots, x_N, z_2, \dots, z_N | x_0, x_1, z_1)^\dagger \\ &= \dots \\ &= \pi_{x_0} \prod_{k=1}^N p_{x_{k-1} x_k} r(z_k; x_{k-1}, x_k) \end{aligned}$$

so the problem is equivalent to

$$\min -\ln(\pi_{x_0}) - \sum_{k=1}^N \ln(p_{x_{k-1} x_k} r(z_k; x_{k-1}, x_k))$$

over all possible sequences $\{x_0, x_1, \dots, x_N\}$

[†]: The calculation can be continued by writing

$$\begin{aligned} p(x_2, \dots, x_N, z_2, \dots, z_N | x_0, x_1, z_1) &= p(x_2, z_2 | x_0, x_1, z_1) p(x_3, \dots, x_N, z_3, \dots, z_N | x_0, x_1, z_1, x_2, z_2) \\ &= p_{x_1 x_2} r(z_2; x_1, x_2) p(x_3, \dots, x_N, z_3, \dots, z_N | x_0, x_1, z_1, x_2, z_2)^\ddagger \end{aligned}$$

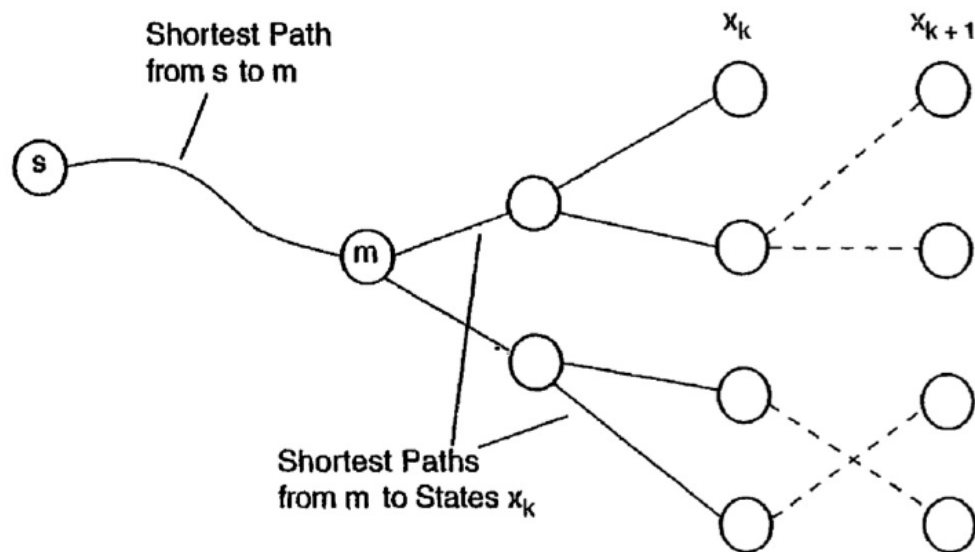
[‡]:

– $P(x_2 | x_0, x_1, z_1) = P(x_2, x_1)$: Markov Property

– $P(z_2 | x_1, x_2) = P(z_2 | x_0, x_1, x_2, z_1)$: Independenc pf observation

- This is a shortest path problem

Advantage of Forward Alogrithm (for Viterbi Algorithm)



Suppose that the shortest paths from s to all states x_k pass through a single node m . If an additional observation is received, the shortest paths from s to all states x_{k+1} will continue pass through m . Therefore, the portion of the state sequence up to node m can be safely estimated because additional observations will not change the initial portion of the shortest paths from s up to m .

Thereofre, we can estimate a portion of state sequence **without waiting** to receive the entire observation sequence Z_N for a number of practical schemes.

This is useful if Z_N is a long sequence.

General Shortest Path Algorithm

- There are many nonDP shortest path algorithms. They can all be used to solve deterministic finite-state problems
- They may be preferable than DP if they avoid calculating the optimal cost-to-go of **EVERY** state
- This is essential for problems with **HUGE** state spaces. Such problems arise for example in combinatorial optimization
- Example: Traveling Salesman Problem (TSP)

Traveling Salesman Problem

: Find the shortest route that starts from Home City o , visits every other city $1, 2, \dots, N$ exactly once and returns to Home City o

- State: (i, S) . Now at city i , and still have to visit cities in S and return to home city o
- Control: j . Which city should be visited next.
- System Dynamic: $(i, s) \xrightarrow{\text{Control: } j} (k, S \setminus \{j\})$
- Stage cost: d_{ij} . Distance from i to j .
 - $J_N(i, \emptyset) = d_{i,o}$
 - $J_k(i, S) = \min_{i \in S} \{d_{i,j} + J_{k+1}(j, S \setminus \{j\})\}^\dagger$
 - $J_0(0, \{1, 2, \dots, N\})$ gives the shortest tour.
- Methods comparison
 - Brute Force: $N!$
 - DP: $N^2 \cdot 2^N$. much less than $N!$

† : Take N evaluations for each pair of (i, S) . There are N cities and 2^N subsets S and N stages.
Total: $N \cdot N \cdot 2^N$

Knapsack Problem

$$\begin{aligned} \max \quad & \sum_{i=1}^N v_i \cdot u_i \\ \text{s.t.} \quad & \sum_{i=1}^N w_i \cdot u_i \leq W \text{ (Integer)} \\ & u_i \in \{0, 1\}, \quad \forall i = 1, \dots, N \end{aligned}$$

- Stage k : Items $1, \dots, k$ have been considered
- Stage x_k : The remaining capacity after considering items $1, \dots, k - 1$. $x_k \in \{0, 1, 2, \dots, W\}$
- Control u_k : Whether to take item k . $u_k \in \{0, 1\}$
- Stage Cost: $v_k \cdot u_k$
- System Dynamics: $x_{k+1} = x_k - w_k \cdot u_k$
- DP Algorithm:
 - $J_k(x_k) = \begin{cases} J_{k+1}(x_k) & \text{if } x_k < w_k \\ \max\{J_{k+1}(x_k), v_k + J_{k+1}(x_k - w_k)\} & \text{if } x_k \geq w_k \end{cases}, \quad 1 \leq k \leq N - 1$
 - $J_N(x_N) = \begin{cases} 0 & \text{if } x_k < w_k \\ v_N & \text{if } x_k \geq w_k \end{cases}$
- Computational Complexity: $O(N \cdot W)$, Pseudo-Polynomial.
For each stage k and each state x_k , at most 2 operations. There are N stages and $W + 1$ possible states for every stage.
 \Rightarrow Total complexity: $O(N \cdot W)$

Label Correcting Methods

- Given: origin s , destination t , lengths $a_{ij} \geq 0$
- Idea is to progressively discover shorter paths from the origin s to every other node i
- Notation
 - d_i (label of i): length of the shortest path found (initially $d_s = 0$, $d_i = \infty$ for $i \neq s$)
 - UPPER: the label d_t of the destination
 - OPEN list: contains nodes that are currently active in the sense that they are candidates for further examination (initially $\text{OPEN} = \{s\}$)

Label Correcting Algorithm

- Step 1 (Node Removal): Remove a node i from OPEN and for each child j of i , do Step 2
- Step 2 (Node Insertion Test): If $d_i + a_{ij} < \min\{d_j, \text{UPPER}\}$, set $d_j = d_i + a_{ij}$ and set i to be the parent of j . In addition, if $j \neq t$, place j in OPEN if it is not already in OPEN, while if $j = t$, set UPPER to the new value $d_i + a_{it}$ of d_t
- Step 3 (Termination Test): If OPEN is empty, terminate; else go to Step 1

Specific Label Correcting Methods

- Breadth-first search
 - Bellman-Ford Algorithm. Works for the cases with negative weights.
 - Computational complexity: $O(|E| \cdot |V|) = O(|V|^3)$
- Depth-first search
- Best-first search
 - Dijkstra's Algorithm. Requires the assumption of nonnegative weights.
 - Computational complexity: $O(|E| + |V|^2) = O(|V|^2)$

Label Correcting Variations

- A* Algorithm can speed up the computational substantially by placing a node j in OPEN in Step 2 only when

$$d_i + a_{ij} + h_j < \text{UPPER}$$

where h_j is an underestimate of the true shortest distance from j to the destination. In this way, fewer nodes will potentially be placed in OPEN before termination.

Since h_j is an underestimate, if $d_i + a_{ij} + h_j \geq \text{UPPER}$, node j need not enter OPEN.

- Another way to sharpen the test $d_i + a_{ij} < \text{UPPER}$ for admission of node j into the OPEN list is to try to reduce UPPER by obtaining an upper bound m_j of the shortest distance from j to t . Then, if $d_j + m_j < \text{UPPER}$, we can reduce UPPER to $d_j + m_j$, thereby making the test in Step 2 more stringent.

Appendix.1 Value of Information (Two-game Chess)

Set $P_w = 0.45$, $P_d = 0.9$

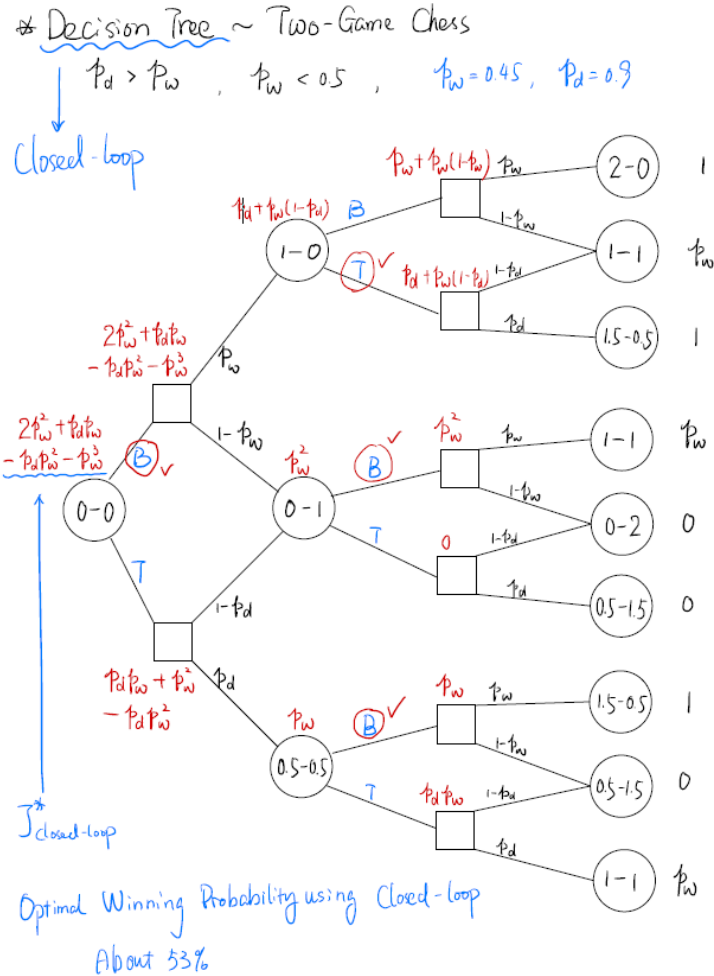
Open-loop

There are 4 possible choices

Policy	Possible cases to win	Possibility to win
$\{B, B\}$	$\{w, w\}, \{l, w, w\}, \{w, l, w\}$	$P_w^2 + 2 \cdot P_w^2(1 - P_w)$
$\{B, T\}$	$\{w, d\}, \{w, l, w\}$	$P_w P_d + P_w^2(1 - P_d)$
$\{T, B\}$	$\{d, w\}, \{l, w, w\}$	$P_d P_w + (1 - P_d) \cdot P_w^2$
$\{T, T\}$	$\{d, d, w\}$	$P_d^2 P_w$

$$J_{\text{open-loop}}^* = \max\{P_w^2 + 2 \cdot P_w^2(1 - P_w), P_w P_d + P_w^2(1 - P_d), P_d P_w + (1 - P_d) \cdot P_w^2, P_d^2 P_w\} = 42.5\%$$

Closed-loop



$$J_{\text{closed-loop}}^* = 53\%$$

Value of information

$$J_{\text{closed-loop}}^* - J_{\text{open-loop}}^* = 10.5\%$$