# CS 4649/7649
# Robot Intelligence: Planning

## Dynamic Programming Revisited

**Sungmoon Joo**

**School of Interactive Computing**
**College of Computing**
**Georgia Institute of Technology**



Some of the slides are from http://www.robots.ox.ac.uk/~az/lectures/opt

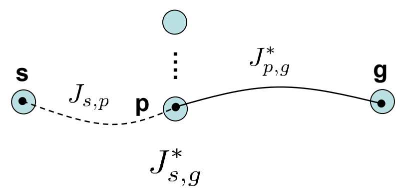
# Dynamic Programming

**DP**

- **Principle of Optimality, Complexity analysis**
- **Optimal Decision Making: Shortest Path(Dijkstra's Algorithm)**
- **Optimal Control: LQR**
- **Optimal Estimation: Kalman Filter**
- **Learning**

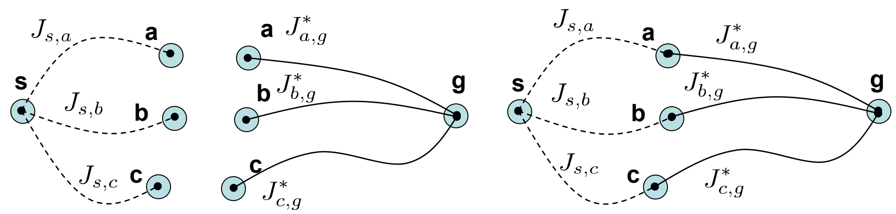
## Principle of Optimality

- Assertion: If s-p-g is the optimal path from s to g, then p-g is the optimal path from p to g
- **Bellman** has called the above property of an optimal policy the Principle of Optimality- "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"






## Principle of Optimality




**Paths resulting from all allowable Decisions at s**

**Optimal paths from a, b, and c to g**

**Candidates for optimal paths from s to g**

$$C^*_{s,a,g} = J_{s,a} + J^*_{a,g}$$
$$C^*_{s,b,g} = J_{s,b} + J^*_{b,g}$$
$$C^*_{s,c,g} = J_{s,c} + J^*_{c,g}$$

Minimum of these costs must be the one associated with the optimal decision at point s

$$C^*_{s,g} = \min_{x \in \{a,b,c\}} \{J_{s,x} + J^*_{x,g}\}$$

Dynamic Programming extends the above decision making concept to sequence of decisions which together define an optimal policy and trajectory

## Dynamic Programming

- Applies to problems where the cost function can be:
  - decomposed into a sequence (ordering) of stages, and
  - each stage depends on only a fixed number of previous stages
- The cost function need not be convex

* Also called the "Viterbi" algorithm



## Discrete vs Continuous DP

● Discrete dynamic programming problems: Number of stages is finite

● When the number of stages tends to infinity then it is called an infinite-stage problem

● Continuous problems are used to solve continuous decision problem

● The classical method of solving continuous decision problems is by the calculus of variations

# Discrete vs Continuous DP

- However, the analytical solutions, using calculus of variations is applicable only for very simple problems
- The infinite-stage dynamic programming approach provides a very efficient numerical approximation procedure for solving continuous decision problems
- For discrete dynamic programming model, the objective function value is the sum of individual stage outputs
- For a continuous dynamic programming model, summation is replaced by integrals of the returns from individual stages
- Such models are useful when infinite number of decisions have to be made in finite time interval



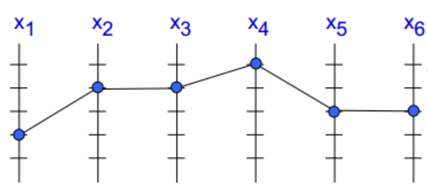
# Dynamic Programming

Consider a cost function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi_i(x_{i-1}, x_i)$$

where $x_i$ can take one of h values

e.g. h=5, n=6




find shortest path

$$f(\mathbf{x}) = \left\{ \begin{array}{l} m_1(x_1) + m_2(x_2) + m_3(x_3) + m_4(x_4) + m_5(x_5) + m_6(x_6) \\ \phi(x_1, x_2) + \phi(x_2, x_3) + \phi(x_3, x_4) + \phi(x_4, x_5) + \phi(x_5, x_6) \end{array} \right.$$

Complexity of minimization:

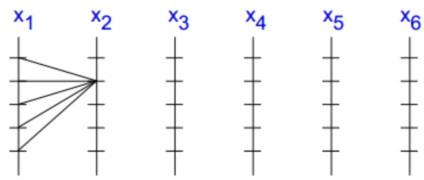
- exhaustive search $O(h^n)$
- dynamic programming $O(nh^2)$

$$x_i = \{x_{i,1}, \ldots, x_{i,h}\},\ i = 1, \ldots, n$$

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**

# Dynamic Programming

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$




Key idea: the optimization can be broken down into n sub-optimizations

**Step 1**: For each value of $x_2$ determine the best value of $x_1$

- Compute

$$\begin{aligned} S_2(x_2) &= \min_{x_1}\{m_2(x_2) + m_1(x_1) + \phi(x_1, x_2)\} \\ &= m_2(x_2) + \min_{x_1}\{m_1(x_1) + \phi(x_1, x_2)\} \end{aligned}$$

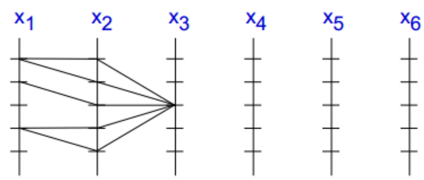
- Record the value of $x_1$ for which $S_2(x_2)$ is a minimum

To compute this minimum for all $x_2$ involves $O(h^2)$ operations

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**



# Dynamic Programming




**Step 2**: For each value of $x_3$ determine the best value of $x_2$ **and** $x_1$

- Compute

$$S_3(x_3) = m_3(x_3) + \min_{x_2}\{S_2(x_2) + \phi(x_2, x_3)\}$$

- Record the value of $x_2$ for which $S_3(x_3)$ is a minimum

Again, to compute this minimum for all $x_3$ involves $O(h^2)$ operations
Note $S_k(x_k)$ encodes the lowest cost partial sum for all nodes up to $k$ which have the value $x_k$ at node $k$, i.e.

$$S_k(x_k) = \min_{x_1, x_2, \ldots, x_{k-1}} \sum_{i=1}^{k} m_i(x_i) + \sum_{i=2}^{k} \phi(x_{i-1}, x_i)$$

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**

## Dynamic Programming

**Algorithm**

- Initialize $S_1(x_1) = m_1(x_1)$
- For $k = 2 : n$

$$S_k(x_k) = m_k(x_k) + \min_{x_{k-1}} \{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\}$$
$$b_k(x_k) = \arg\min_{x_{k-1}} \{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\}$$

- Terminate

$$x_n^* = \arg\min_{x_n} S_n(x_n)$$

- Backtrack (optimal trajectory)

$$x_{i-1} = b_i(x_i)$$

Complexity $O(nh^2)$

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**



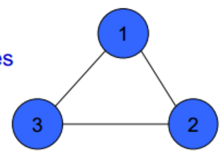
## DP on Graphs

- Graph $(V, E)$
- Vertices $v_i$ for $i = 1, \ldots, n$
- Edges $e_{ij}$ connect $v_i$ to other vertices $v_j$

$$f(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_{e_{ij} \in E} \phi(v_i, v_j)$$

e.g.

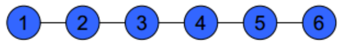
- 3 vertices, each can take one of h values
- 3 edges



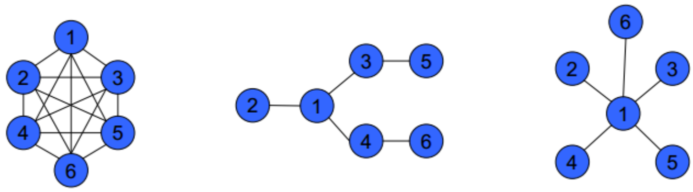

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**

# DP on Graphs

So far have considered chains




Can dynamic programming be applied to these configurations?
(i.e. to reduce the optimization complexity from $O(h^n)$ to $O(nh^2)$)

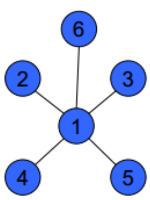


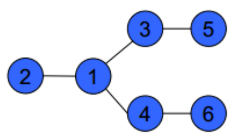

Fully connected | Tree structure | Star structure

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**



# DP on Graphs



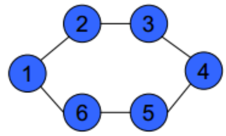




- for a value of the central node determine the best value of each of the other nodes in turn $O(nh)$
- repeat for all values of the central node $O(h)$
- final complexity $O(nh^2)$

- order the nodes by their depth from the root, where d is max depth
- start with nodes at depth d-1, and compute best value for all (child) nodes at depth d, $O(h^2)$
- decrease depth and repeat, $O(n)$
- final complexity $O(nh^2)$

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**

# DP on Graphs

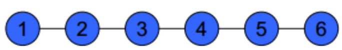


- select one node and choose its value
- for the other nodes, the graph is then equivalent to an open chain and can be optimized with $O(nh^2)$ complexity
- repeat for all values of the selected node and choose lowest overall cost from these
- final complexity $O(nh^3)$

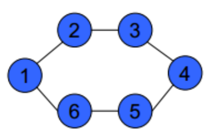
**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**



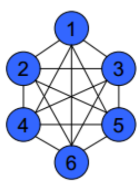
# DP on Graphs



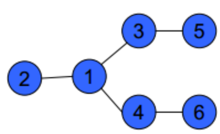
Open chain $O(nh^2)$



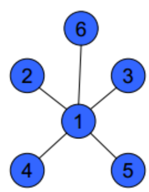
Closed chain $O(nh^3)$



Fully connected



Tree structure



Star structure

$O(h^n)$ $O(nh^2)$ $O(nh^2)$

**Source: http://www.robots.ox.ac.uk/~az/lectures/opt**

## Characteristics of DP solution

[1] Absolute Minimum
- The solution obtained from DP is the absolute(or global) minimum

[2] DP vs Exhaustive Search
- DP makes the direct search feasible b/c, instead of exhaustive search, we consider only those that satisfy a necessary condition – "the principle of optimality
- The number of calculations required by exhaustive search (direct enumeration) increases exponentially with the number of stages, while those of DP increase linearly

**Exhaustive Search: $O(h^n)$**
**DP: $O(nh^2)$ ~ $O(nh^3)$**



## Curse of Dimensionality

- Limitation of dynamic programming: Dimensionality restriction
- The number of samples we have to visit increases rapidly as the number of variables and stages increase, even though we use DP.

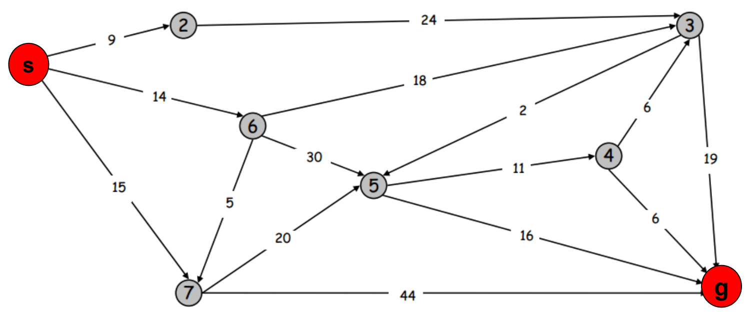
*The interpretation of Curse of Dimensionality is not unique.

## Shortest Path Problem

Given: a weighted directed graph, with a single source s

Goal: Find the shortest path from s to t

Length of path = Σ Length of arcs






## Dijkstra's Algorithm('59)

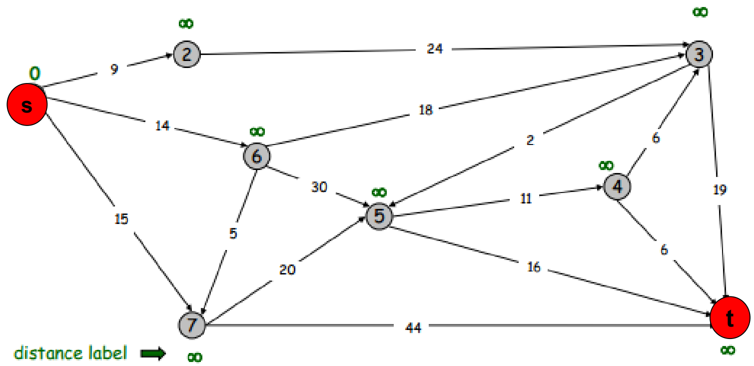
Given: a directed graph with non-negative edge costs

Find: the shortest path from s to every other vertex

- pick the unvisited vertex with the lowest distance to s
- calculate the distance through it to each unvisited neighbor, and update the neighbor's distance if smaller
- mark vertex as visited once all neighbors explored

# Dijkstra's Algorithm('59)

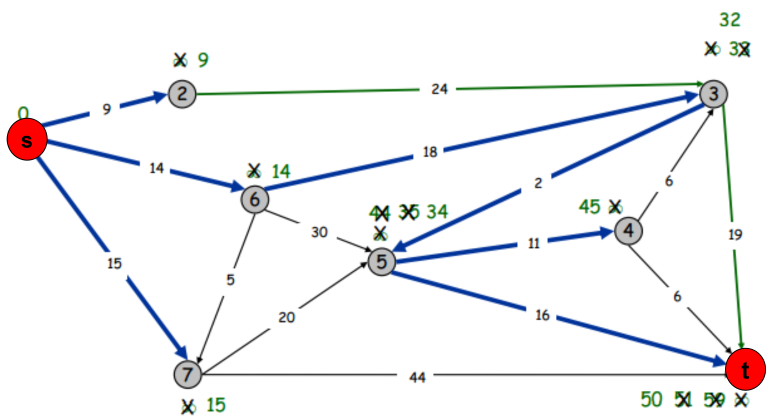
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }






# Dijkstra's Algorithm('59)

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



## Continuous LQR

- Linear Dynamic System: $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$
- Quadratic Performance Criterion: $C(\mathbf{x},\mathbf{u}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u}$
- Gain Matrix: $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}$ $\quad \mathbf{u} = \mathbf{K}\mathbf{x} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{x}$
- Algebraic Riccati Equation:

$$\boxed{0 = \mathbf{Q} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P}}$$



## Continuous LQR

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)\ ,\ x(t_0) = x_0\ ,\ t_0 \le t \le t_f$$

$$J = \frac{1}{2}x^T(t_f)Sx(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\{x^T(t)Q(t)x(t) + u^T(t)R(t)u(t)\}\,dt\ ,\ Q(t) = C^T(t)C(t)$$

$$u^*(t) = -R^{-1}(t)B^T(t)P(t)x(t)$$

$$J^*(x(t_0)) = \frac{1}{2}x^T(t_0)P(t_0)x(t_0)$$

$$-\dot{P}(t) = A^T(t)P(t) + P(t)A(t) - P(t)B(t)R^{-1}(t)B^T(t)P(t) + C^T(t)C(t)$$

$$P(t_f) = S$$

**Stationary case(P = constant) was discussed in the previous lecture**

# Discrete LQR

$$x(k+1) = A(k)x(k) + B(k)u(k)\ ,\ x(0) = x_0\ ,\ k = 0, \ldots, N-1$$

$$J = \frac{1}{2}x^T(N)Sx(N) + \frac{1}{2}\sum_{k=0}^{N-1}\left\{x^T(k)Q(k)x(k) + u^T(k)R(k)u(k)\right\}\ ,\ Q(k) = C^T(k)C(k)$$

$$J^*(x(k)) = \min_{u(k)}\left\{\frac{1}{2}x^T(k)Q(k)x(k) + \frac{1}{2}u^T(k)R(k)u(k) + J^*(x(k+1))\right\}$$

$$J^*(x(N)) = \frac{1}{2}x^T(N)Sx(N)$$

$$\begin{aligned} J^*(x(N-1)) &= \min_{u(N-1)}\left\{\frac{1}{2}x^T(N-1)Q(N-1)x(N-1) + \frac{1}{2}u^T(N-1)R(k)u(N-1) + J^*(x(N))\right\} \\ &= \min_{u(N-1)}\left\{\frac{1}{2}x^T(N-1)Q(N-1)x(N-1) + \frac{1}{2}u^T(N-1)R(k)u(N-1)\right. \\ &\qquad \left. + \frac{1}{2}x^T(N)Sx(N)\right\} \end{aligned}$$



# Discrete LQR

$$\begin{aligned} J^*(x(N-1)) &= \min_{u(N-1)}\left\{\frac{1}{2}x^T(N-1)Q(N-1)x(N-1) + \frac{1}{2}u^T(N-1)R(k)u(N-1) + J^*(x(N))\right\} \\ &= \min_{u(N-1)}\left\{\frac{1}{2}x^T(N-1)Q(N-1)x(N-1) + \frac{1}{2}u^T(N-1)R(k)u(N-1)\right. \\ &\qquad \left. + \frac{1}{2}x^T(N)Sx(N)\right\} \end{aligned}$$

$$\Big\downarrow \quad \textbf{Exercise!}$$

$$u^*(N-1) = -\left[R(N-1) + B^T(N-1)SB(N-1)\right]^{-1} B^T(N-1)SA(N-1)x(N-1)$$

**Feedback controller!!**

# Discrete LQR

$\downarrow$ **Exercise!**

$$J^*(x(N-1)) = \frac{1}{2}x^T(N-1)\Big\{Q(N-1) + A^T(N-1)SA(N-1) - A^T(N-1)SB(N-1)$$
$$\left[R(N-1) + B^T(N-1)SB(N-1)\right]^{-1} B^T(N-1)SA(N-1)\Big\}x(N-1)$$
$$= \frac{1}{2}x^T(N-1)P(N-1)x(N-1)$$

$$u^*(k) = -\left[R(k) + B^T(k)P(k+1)B(k)\right]^{-1} B^T(k)P(k+1)A(k)x(k)$$

$$J^*(x(k)) = \frac{1}{2}x^T(k)P(k+1)x(k)$$

$$P(k) = A^T(k)P(k+1)A(k) + Q(k)$$
$$- A^T(k)P(k+1)B(k)\left[R(k) + B^T(k)P(k+1)B(k)\right]^{-1} B^T(k)P(k+1)A(k)$$
$$P(N) = S$$