Reinforcement Learning **Spring Semester, 2018/9**

Lecture 2: March 6, 2019

Lecturer: Yishay Mansour *Scribe: ym*

In this chapter we introduce the dynamic system viewpoint of the optimal planning problem. We restrict the discussion here to deterministic (rather than stochastic) systems, and consider the finite-horizon decision problem and its recursive solution via finite-horizon Dynamic Programming.

2.1 Discrete Dynamic Systems

We consider a discrete-time dynamic system, of the form:

$$x_{k+1} = f_k(x_k, u_k), \quad k = 0, 1, 2, \ldots, N-1$$

where

- k is the time index.
- $x_k \in X_k$ is the state variable at time k, and X_k is the set of possible states at time k.
- $u_k \in U_k$ is the control variable at time k, and U_k is the set of possible control values (or actions) at time k.
- $f_k : X_k \times U_k \to X_{k+1}$ is the state transition function, which defines the *state dynamics* at time k.
- $N > 0$ is the *time horizon* of the system. It can be finite or infinite.

Remark 2.1. *More generally, the set U_k of available actions may depend on the state at time k, namely: $u_k \in U_k(x_k) \subset U_k$.*

Remark 2.2. *The system is in general time-varying. It is called* time invariant *if f_k, X_k, U_k do not depend on the time k. In that case we write*

$$x_{k+1} = f(x_k, u_k), \quad k = 0, 1, 2, \ldots, N-1; \quad x_k \in X,\ u_k \in U(x_k).$$

Remark 2.3. *The state dynamics may be augmented by an output equation:*

$$y_k = h_k(x_k, u_k),$$

where y_k is the system output, or the observation. In most of this course we implicitly assume that $y_k = x_k$, namely, the current state x_k is fully observed.

Example 2.1. ***Linear Systems***

The best known example of a dynamic system is that of a linear time-invariant system, where:

$$x_{k+1} = Ax_k + Bu_k$$

with $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$. Here the state and action spaces are evidently continuous (as opposed to discrete).

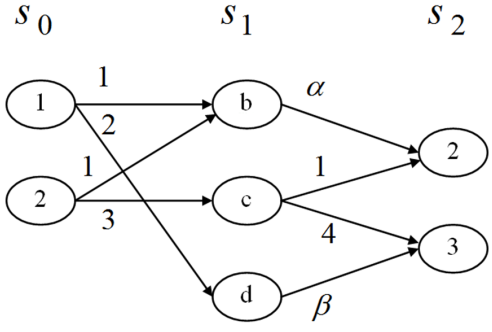
Example 2.2. ***Finite models***

Our emphasis here will be on finite state and action *models. A finite state space contains a finite number of points: $X_k = \{1, 2, \ldots, n_k\}$. Similarly, a finite action space implies a finite number of control values at each stage:*

$$U_k(x) = \{1, 2, \ldots, m_k(x)\}, \quad x \in X_k$$

Notation for finite models: When the state and action spaces are finite, it is common to denote the state by s_k (instead of x_k) and the actions by a_k (instead of u_k). That is, the system equations are written as: $s_{k+1} = f_k(s_k, a_k)$, $k = 0, 1, 2, \ldots, N-1$ with $s_k \in S_k$, $a_k \in A_k(x_k) \subset A_k$. **We will adhere to that notation in the following**.

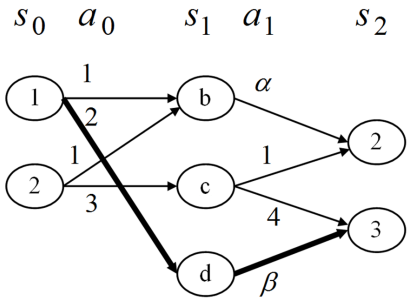
Graphical description: Finite models (over finite time horizons) can be represented by a corresponding decision graph:



Here:

- $N = 2$, $S_0 = \{1, 2\}$, $S_1 = \{b, c.d\}$, $S_2 = \{2, 3\}$,
- $A_0(1) = \{1, 2\}$, $A_0(2) = \{1, 3\}$, $A_1(b) = \{\alpha\}$, $A_1(c) = \{1, 4\}$, $A_1(d) = \{\beta\}$
- $f_0(1, 1) = b$, $f_0(1, 2) = d$, $f_0(2, 1) = b$, $f_0(2, 3) = c$, $f_1(b, \alpha) = 2$, etc.

Definition 2.1. ***Feasible Path***
A feasible path for the specified system is a sequence $(s_0, a_0, \ldots, s_{N-1}, a_{N-1}, s_N)$ *of states and actions, such that* $s_k \in A_k(s_k)$ *and* $s_{k+1} = f_k(s_k, a_k)$.



A feasible path

2.2 The Finite Horizon Decision Problem

We proceed to define our first and simplest planning problem. For that we need to specify a *performance objective* for our model, and the notion of *control policies*.

2.2.1 Costs and Rewards

The cumulative cost: Let $h_N = (s_0, a_0, \ldots, s_{N-1}, a_{N-1}, s_N)$ denote an N-stage path for the system. To each feasible path h_N we wish to assign some cost $C_N = C_N(h_N)$.

The standard definition of the cost C_N is through the following *cumulative cost functional*:

$$C_N(h_N) = \sum_{k=0}^{N-1} c_k(s_k, a_k) + c_N(s_N)$$

Here:

- $c_k(s_k, a_k)$ is the *instantaneous* cost or *single-stage* cost at stage k, and c_k is the instantaneous cost function.
- $c_N(s_N)$ is the *terminal* cost, and c_N is the terminal cost function.

Note:

- The cost functional defined above is *additive* in time. Other cost functionals are possible, for example the max cost, but additive cost is by far the most common and useful.
- We shall refer to C_N as the *cumulative N-stage cost*, or just the *cumulative cost*.

Our objective is to *minimize* the cumulative cost C_N, by a proper choice of actions. We will define that goal more formally below.

Cost versus reward formulation: It is often more natural to consider *maximizing* reward rather than minimizing cost. In that case, we define the cumulative N-stage return function:

$$R_N(h_N) = \sum_{k=0}^{N-1} r_k(s_k, a_k) + r_N(s_N)$$

Here and r_k is the running reward, and r_N is the terminal reward. Clearly, minimizing C_N is equivalent to maximizing R_N, if we set:

$$r_k(s, a) = -c_k(s, a) \text{ and } r_N(s) = -c_N(s).$$

2.2.2 Optimal Paths

Our first planning problem is the following *Path Optimization Problem*:

- For a given initial state s_0, find a feasible path $h_N = (s_0, a_0, \ldots, s_{N-1}, a_{N-1}, s_N)$ that minimizes the cost functional $C_N(h_N)$, over all feasible paths h_N.

Such a path is called an *optimal path* from s_0.

A more general notion than a path is that of a *control policy*, that specifies the action to be taken at each state. Control policies will play an important role in our Dynamic Programming algorithms, and are defined next.

2.2.3 Control Policies

In general we will consider a few classes of control policies. The two basic dimensions in which we will characterize the control policies is their dependence on the history, and their use of randomization.

- A general or **history-dependent** control policy $\pi = (\pi_t)_{t\in\mathbf{T}}$ is a mapping from each possible history $h_t = (s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t)$, $t \in \mathbf{T}$, to an action $a_t = \pi_t(h_t) \in A_t$. We denote the set of general policies by Π_H.

- A **Markov** control policy π is allowed to depend on the current state and time only: $a_t = \pi_t(s_t)$. We denote the set of Markov policies by Π_M.

- For stationary models, we may define **stationary** control policies that depend on the current state alone. A stationary policy is defined by a single mapping $\pi : S \to A$, so that $a_t = \pi(s_t)$ for all $t \in \mathbf{T}$. We denote the set of stationary policies by Π_S.

- Evidently, $\Pi_H \supset \Pi_M \supset \Pi_S$.

Randomized (Stochastic) Control policies

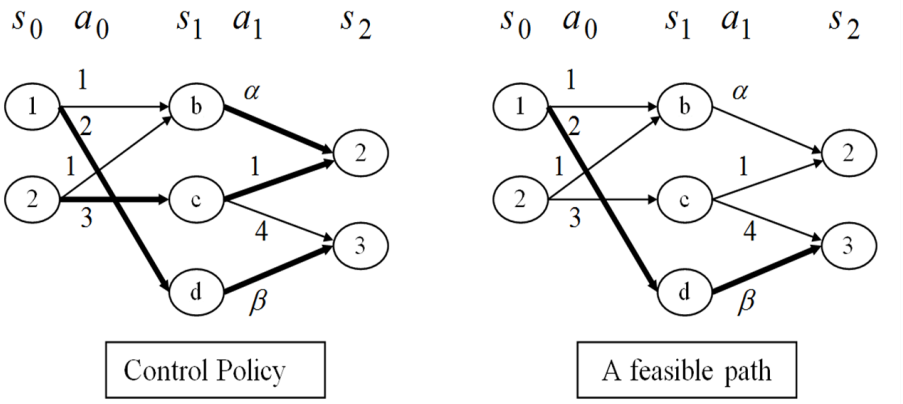
- The control policies defined above specify deterministically the action to be taken at each stage. In some cases we want to allow for a random choice of action.

- A general randomized (stochastic) control policy assigns to each possible history h_t a probability distribution $\pi_t(\cdot|h_t)$ over the action set A_t. That is, $prob\{a_t = a|h_t\} = \pi_t(a|h_t)$. We denote the set of general randomized policies by Π_{HS}.

- Similarly, we can define the set Π_{MS} of Markov randomized (stochastic) control policies, where $\pi_t(\cdot|h_t)$ is replaced by $\pi_t(\cdot|s_t)$, and the set Π_{SS} of stationary randomized (stochastic) control policies, where $\pi_t(\cdot|s_t)$ is replaced by $\pi(\cdot|s_t)$.

- Note that the set Π_{HS} includes all other policy sets as special cases.

Control policies and paths: As mentioned, a control policy specifies an action for each state, whereas a path specifies an action only for states along the path. This distinction is illustrated in the following figure.



Induced Path: A control policy π, together with an initial state s_0, specify a feasible path $h_N = (s_0, a_0, \ldots, s_{N-1}, a_{N-1}, s_N)$. This path may be computed recursively using $a_k = \pi_k(s_k)$ and $s_{k+1} = f_k(s_k, a_k)$, for $k = 0, 1, \ldots, N-1$.

Remark 2.4. *Suppose that for each state s_k, each action $a_k \in A_k(s_k)$ leads to a different state s_{k+1} (i.e., at most one edge connects any two states). We can then identify each action $a_k \in A_k(s)$ with the next state $s_{k+1} = f_k(s, a_k)$ it induces. In that case a path may be uniquely specified by the state sequence $(s_0, s_1, \ldots, s_N)$.*

2.2.4 Reduction between control policies classes

We first show a reduction from a general history dependent policies to Randomized Markovian policies. The main observation is that the only influence on the cost is the expected cost $E[c_t(s_t, a_t)]$. Namely, let $\rho_t(s, a) = E_{h_{t-1}}[\Pr[s_t = s, a_t = a]]$, where h_{t-1} is the history of the first $t-1$ time steps. Now we can rewrite the expected cost to go as,

$$E[J^\pi(s_0)] = E[\sum_{t=1}^{T-1} \sum_{a \in A_t, s \in S_t} c_t(s, a)\rho_t(s, a)].$$

This implies that we need to show that we can preserve the probabilities $\rho_t(s, a)$.

Theorem 2.1. *For any policy $\pi \in \Pi_{HS}$, there is a policy $\pi' \in \Pi_{MS}$, such that*

$$E[J^\pi(s_0)] = E[J^{\pi'}(s_0)]$$

Proof. Given the policy $\pi \in \Pi_{HS}$, we define $\pi' \in \Pi_{MS}$ as follows. For every state $s \in S_t$ we define

$$\pi'(a|s) = \Pr_{h_{t-1}}[a_t = a|s_t = s] = \frac{\rho_t(s,a)}{\sum_{a' \in A_t} \rho_t(s,a')}$$

By definition π' is Markovian (depends only on the time t and the realized state s). By construction $\rho_t(s,a)$ is identical for π and π', which implies that $E[J^{\pi}(s_0)] = E[J^{\pi'}(s_0)]$. □

Next we show that for any stochastic Markovian policy there is a deterministic Markovian policy with at most the same cost to go.

Theorem 2.2. *For any policy $\pi \in \Pi_{MS}$, there is a policy $\pi' \in \Pi_{MD}$, such that*

$$E[J^{\pi}(s_0)] \geq E[J^{\pi'}(s_0)]$$

Proof. By backward induction on the steps. The inductive claim is:
for any policy $\pi \in \Pi_{MS}$ which is deterministic in $[t+1, T]$, there is a policy $\pi' \in \Pi_{MS}$ which is deterministic in $[t, T]$ and $E^{\pi}[J(path)] \geq E^{\pi'}[J(path)]$.
Clearly the theorem follows for $t = 0$.

For the base of the induction we can take $t = T$, which holds trivially.

For the inductive step, assume that $\pi \in \Pi_{MS}$ is deterministic in $[t+1, T]$. For every $s_{t+1} \in S_{t+1}$ define

$$J(s_{t+1}) = J(path(s_{t+1}, \ldots, s_T)),$$

where $path(s_{t+1}, \ldots, s_T)$ is the deterministic path from s_{t+1}.

We define π' to be identical to π for all time steps $t' \neq t$. We define π' for each $s_t \in S_t$ as follows:

$$\pi'(a_t, s_t) = \arg\min_{a \in A_t} J(f_t(s_t, a)).$$

Recall that since we have a Deterministic Decision Process $f_t(s_t, a) \in S_{t+1}$ is the next state if we take action a in s_t. (Remark: For a stochastic f_t we would simply take the expectation.)

For the analysis, note that π and π' are identical until time t, so they generate exactly the same distribution over paths. At time t, π' is define to minimize the cost to go from s_t, given that we follow π from $t+1$ to T. Therefore the cost can only decrease. Formally,

$$\begin{aligned} E^{\pi}[J(s_t, \ldots, s_T)] =& E^{\pi} E_{a_t}[J(f_t(s_t, a_t), s_{t+1}, \ldots, s_T)] \\ \geq& E^{\pi} \min_{a_t \in A_t}[J(f_t(s_t, a_t), s_{t+1}, \ldots, s_T)] \\ =& E^{\pi'}[J(f_t(s_t, a_t), s_{t+1}, \ldots, s_T)] \end{aligned}$$

□

2.2.5 Optimal Control Policies

Definition 2.2. *A control policy π is called **optimal** if, for each initial state s_0, it induces an optimal path h_N from s_0.*

An alternative definition can be given in terms of policies only. For that purpose, let $h_N(\pi; s_0)$ denote the path induced by the policy π from s_0. For a given return functional $R_N(h_N)$, denote $R_N(\pi; s_0) = R_N(h_N(\pi; s_0))$ That is, $R_N(\pi; s_0)$ is the cumulative return for the path induced by π from s_0.

Definition 2.3. *A control policy π is called optimal if, for each initial state s_0, it holds that $R_N(\pi; s_0) \geq R_N(\tilde{\pi}; s_0)$ for any other policy $\tilde{\pi}$.*

Equivalence of the two definitions can be easily established (exercise). An optimal policy is often denoted by π^*.

The standard finite-horizon planning problem: Find a control policy π for the N-stage decision problem with the cumulative return (or cost) function.

The naive approach to finding an optimal policy: For finite models (i.e., finite state and action spaces), the number of feasible paths (or control policies) is finite. It is therefore possible, in principle, to enumerate all N-stage paths, compute the cumulative return for each one, and choose the one which gives the largest return. Let us evaluate the number of different paths and control policies. Suppose for simplicity that number of states at each stage is the same: $|X_k| = n$, and similarly the number of actions at each state is the same: $|A_k(x)| = m$ (with $m \leq n$) . The number of feasible N-stage paths for each initial state is seen to be m^N. The number of different policies is m^{nN}. For example, for a fairly small problem with $N = n = m = 10$, we obtain 10^{10} paths for each initial state (and 10^{11} overall), and 10^{100} control policies. Clearly it is not possible to enumerate them all.

Fortunately, Dynamic Programming offers a drastic simplification of the computational complexity for this problem.

2.3 Finite Horizon Dynamic Programming

The Dynamic Programming (DP) algorithm breaks down the N-stage decision problem into N sequential single-stage optimization problems. This results in dramatic

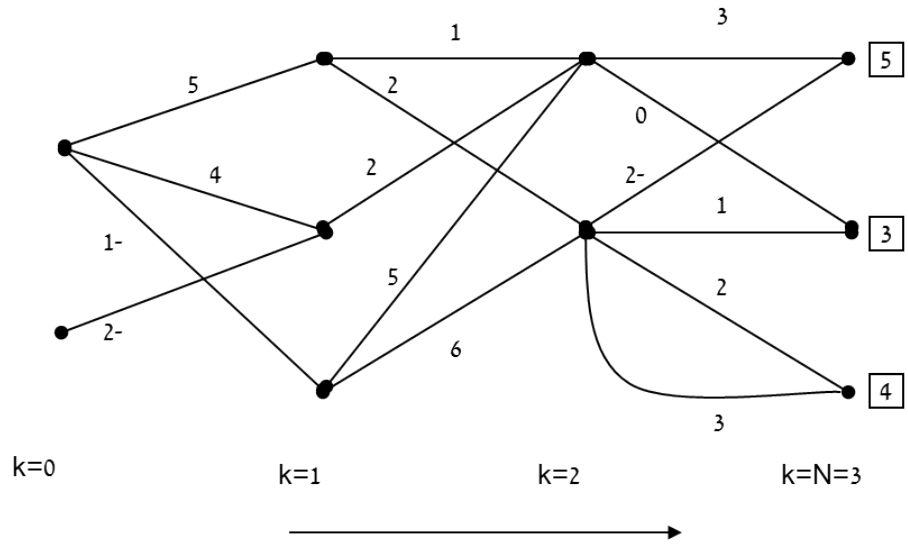
improvement in computation efficiency.

The DP technique for dynamic systems is based on a general observation called Bellman's Principle of Optimality. Essentially it states the following (for deterministic problems):

- **Any sub-path of an optimal path is itself an optimal path between its end point.**

Applying this principle recursively from the last stage backward, obtains the (backward) Dynamic Programming algorithm. Let us first illustrate the idea with following example.

Example 2.3. *Shortest path on a decision graph: Suppose we wish to find the shortest path (minimum cost path) from the initial node in N steps.*



The boxed values are the terminal costs at stage N, the other number are the link costs. Using backward recursion, we may obtain that the minimal path costs from the two initial states are 7 and 3, as well as the optimal paths and an optimal policy.

We can now describe the DP algorithm. Recall that we consider the dynamic system

$$s_{k+1} = f_k(s_k, a_k), \quad k = 0, 1, 2, \ldots, N-1$$
$$s_k \in S_k, \quad a_k \in A_k(s_k)$$

and we wish to maximize the cumulative return:

$$R_N = \sum_{k=0}^{N-1} r_k(s_k, a_k) + r_N(s_N)$$

The DP algorithm computes recursively a set of **value functions** $V_k : S_k \to \mathbb{R}$, where $V_k(s_k)$ is the value of an optimal sub-path $h_{k:N} = (s_k, a_k, \ldots, s_N)$ that starts at s_k.

Algorithm 2.1. ***Finite-horizon Dynamic Programming***

1. *Initialize the value function:* $V_N(s) = r_N(s)$, $s \in S_N$.

2. *Backward recursion: For* $k = N-1, \ldots, 0$, *compute*

$$V_k(s) = \max_{a \in A_k} \left\{ r_k(s,a) + V_{k+1}(f_k(s,a)) \right\}, \quad s \in S_k.$$

3. *Optimal policy: Choose any control policy* $\pi = (\pi_k)$ *that satisfies:*

$$\pi_k(s) \in \arg\max_{a \in A_k} \left\{ r_k(s,a) + V_{k+1}(f_k(s,a)) \right\}, \quad k = 0, \ldots, N-1.$$

Proposition 2.1. *The following holds for finite-horizon dynamic programming:*

1. *The control policy* π *computed above is an optimal control policy for the* N*-stage decision problem.*

2. $V_0(s)$ *is the optimal N-stage return from initial state* $s_0 = s$*:*

$$V_0(s) = \max_{\pi} R_N(\pi; s), \quad \forall s \in S_0.$$

Proof. We show that the computed policy is optimal and its value is V_k. We will establish the following inductive claim:

For any t *and any* s*, the path from* s *define by* π^* *is the minimum cost path of length* $N - t$*. The value of* $V_t(s)$ *is the min cost from* s*.*

The proof is by a backward induction. For the basis of the induction we have: $t = 0$, and the inductive claim follows from the initialization.

Assume the inductive claim holds for t prove for $t + 1$. For contradiction assume there is a lower cost path from s. Let the path generated by by π^* is $P = (s, s_{N-t}, \ldots, S_N)$ Let $P' = (s, s'_{N-t}, \ldots, s'_N)$ be an alternative path. Let $P'' = (s'_{N-t}, \ldots, s'_N)$ be the path from s'_{N-t} of π^*. From the inductive claim we have that $R(P') \leq R(P'')$. From the Backward recursion step, we have that $R(P'') \leq R(P)$. This implies that $R(P') \leq R(P)$, and the inductive claim holds. □

Let us make the following observations:

1. The algorithm involves visiting each state exactly once, proceeding backward in time. For each time instant (or stage) k, the value function $V_k(s)$ is computed for all states $s \in S_k$ before proceeding to stage $k-1$.

2. The recursive equation in part 2 of the algorithm, along with similar equations in the theory of DP, is called **Bellman's equation**.

3. Computational complexity: There is a total of nN states (excluding the final one), and in each we need m computations. Hence, the number of required calculations is mnN. For the example above with $m = n = N = 10$, we need $O(10^3)$ calculations.

4. A similar algorithm that proceeds forward in time (from $k = 0$ to $k = N$) can be devised. We note that this will not be possible for stochastic systems (i.e., the stochastic MDP model).

5. The celebrated **Viterbi algorithm** is an important instance of finite-horizon DP. The algorithm essentially finds the most likely sequence of states in a Markov chain (s_k) that is partially (or noisily) observed. The algorithm was introduced in 1967 for decoding convolution codes over noisy digital communication links. It has found extensive applications in communications, and is a basic computational tool in Hidden Markov Models (HMMs), a popular statistical model that is used extensively in speech recognition and bioinformatics, among other areas.

2.4 Shortest Paths

We can formulate a DDP problems similar to shortest path problems. Given a directed graph $G(V, E)$, there is a set of goal states S_G, and the goal is to reach one of the goal states. Formally, when we reach a goal state we stay there and have a zero cost. For such a DDP the optimal policy would be to compute a shortest path to one of the goal states.

There is a variety of algorithms for computing shortest paths.

1. Bellman-Ford: handles also negative weights, but assumes no negative cycle. Runs in time $O(|V| \cdot |E|)$.

2. Dijkstra: Assumes non-negative weights. Runs in time $O(|V| \log |E| + |E|)$

2.5 Average cost criteria

The average cost criteria considers the limit of the average costs. Formally:

$$J^{\pi}_{avg} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} c_t(s_t, a_t)$$

where the trajectory is generated using π. The aim is to minimize $E[J^{\pi}_{avg}]$. This implies that any finite prefix has no influence of the final average cost, since its influence vanishes as T goes to infinity.

For a deterministic stationary policy, the policy converges to a simple cycle, and the average cost is the average cost of the edges on the cycle. (Recall, we are considering only DDP.)

Given a directed graph $G(V, E)$, let Ω be the collection of all cycles in $G(V, E)$. For each cycle $\omega = (v_1, \ldots, v_k)$, we define $c(\omega) = \sum_{i=1}^{k} c(v_i, v_{i+1})$, where (v_i, v_{i+1}) is the i-th edge in the cycle ω. Let $\mu(\omega) = \frac{c(\omega}{k}$. The *min average cost cycle* is

$$\mu^* = \min_{\omega\in\Omega} \mu(\omega)$$

We show that the min average cost cycle is the optimal policy.

Theorem 2.3. *For any Deterministic Decision Process (DPP) the optimal average costs is μ^*, and an optimal policy is π_ω that cycles around a simple cycle of average cost μ^*.*

Proof. Let ω be a cycle of average cost μ^*. Let π_ω be a deterministic stationary policy that first reaches ω and then cycles in ω. Clearly $J^{\pi_\omega}_{avg} = \mu^*$.

We show that for any policy π (possibly in Π_{HS}) we have that $E[J^{\pi}_{avg}] \geq \mu^*$. For contradiction assume that there is a policy π' that has average cost $\mu^* - \varepsilon$. Consider a sufficiently long run of length T of π', and fix any realization θ of it. We will show that $c(\theta) \geq (T - n)\mu^*$, which implies that $E[J^{\pi}_{avg}] \geq \mu^* - n\mu^*/T$.

Given θ, consider the first simple cycle ω in θ. The average cost of ω is $\mu(\omega) \geq \mu^*$. Delete ω from θ, reducing the number of edges by $|\omega|$ and the cost by $\mu(\omega)|\omega|$. We continue the process until there is not remaining cycles, which implies that we have at most $|V| = n$ nodes remaining. This implies that the costs of ω was at least $(T - n)\mu^*$. This implies that the average cost of θ is at least $c(\theta) \geq (1 - \frac{n}{T})\mu^*$. For $\epsilon > \mu^* n/T$ we have a contradiction. □

Next we develop an algorithm for computing the minimum average cost cycle, which implies an optimal policy for DDP for average costs. The input is a directed graph $G(V, E)$ with cost $c : E \to \mathbb{R}$.

We first give a characterization of μ^*. Set a root $r \in V$. Let $F_k(v)$ be paths of length k from r to v. Let $d_k(v) = \min_{p \in F_k(v)} c(p)$, where if $F_k(v) = \emptyset$ then $d_k(v) = \infty$. The following theorem (due to [R. Karp, A characterization of the minimum cycle mean in digraph, Discrete mathematics, 1978]) gives a characterization of μ^*.

Theorem 2.4.

$$\mu^* = \min_{v \in S} \max_{0 \le k \le n-1} \frac{d_n(v) - d_k(v)}{n-k} ,$$

where we define $\infty - \infty$ *as* ∞.

Proof. We have two cases, $\mu^* = 0$ and $\mu^* > 0$. We assume that the graph has no negative cycle (we can guarantee this by adding a large number M to all the weights).

We start with $\mu^* = 0$. This implies that we have in $G(V, E)$ a cycle of weight zero, but no negative cycle. For the theorem it is sufficient to show that

$$\min_{v \in S} \max_{0 \le k \le n-1} \{d_n(v) - d_k(v)\} = 0.$$

For every node $v \in S$ there is a path of length $k \in [0, n-1]$ and cost $d(v)$. This implies that

$$\max_{0 \le k \le n-1} \{d_n(v) - d_k(v)\} = d_n(v) - d(v) \ge 0$$

We need to show that for some $v \in V$ we have $d_n(v) = d(v)$, which implies that $\min_{v \in S}\{d_n(v) - d(v)\} = 0$.

Consider a cycle ω of cost $c(\omega) = 0$ (there is one, since $\mu^* = 0$). Let v be a node on ω. Consider a path P from r to v which then cycles around ω and has length at least n. The path P is a shortest path to v (although not necessarily simple). This implies that any sub-path of P is also a shortest path. Let P' be a sub-path of P of length n and let it end in $u \in V$. Path P' is a shortest path to u, since it is a prefix of a shortest path P. This implies that the cost of P' is $d(u)$. Since P' is of length n, by construction, we have that $d_n(u) = d(u)$. Therefore, $\min_{v \in S}\{d_n(v) - d(v)\} = 0$, which completes the case that $\mu^* = 0$.

For $\mu^* > 0$ we subtract a constant $\Delta = \mu^*$. This implies that in the new costs we have a zero cycle and no negative cycle. We can now apply the previous case. It only remains to show that the formula changes by exactly $\Delta = \mu^*$.

Formally, let $c'(e) = c(e) - \Delta$. For any path p we have $c'(p) = c(p) - |p|\Delta$, and for any cycle ω we have $\mu'(\omega) = \mu(\omega) - \Delta$. This implies that for $\Delta = \mu^*$ we have a

cycle of cost zero. We now consider the formula,

$$\begin{aligned}
0 = (\mu')^* &= \min_{v \in V} \max_{0 \le k \le n-1} \{ \frac{d'_n(v) - d'_k(v)}{n-k} \} \\
&= \min_{v \in V} \max_{0 \le k \le n-1} \{ \frac{d_n(v) - n\Delta - d_k(v) + k\Delta}{n-k} \} \\
&= \min_{v \in V} \max_{0 \le k \le n-1} \{ \frac{d_n(v) - d_k(v)}{n-k} - \Delta \} \\
&= \min_{v \in V} \max_{0 \le k \le n-1} \{ \frac{d_n(v) - d_k(v)}{n-k} \} - \Delta
\end{aligned}$$

Therefore we have

$$\mu^* = \Delta = \min_{v \in V} \max_{0 \le k \le n-1} \{ \frac{d_n(v) - d_k(v)}{n-k} \}$$

□

We would like now to recover the minimum average cost cycle. The basic idea is to recover the cycle from the minimizing vertices in the formula, but some care need to be taken. It is true that some minimizing pair (v, k) the path of length n from r to v has a cycle of length $n - k$ which is the suffix of the path. The solution is that for the path p, from r to v of length n, any simple cycle is a minimum average cost cycle. (See, ["A note of finding minimum mean cycle", Mmamu Chaturvedi and Ross M. McConnell, IPL 2017]).

The running time of computing the minimum average cost cycle is $O(|V| \cdot |E|)$.

2.6 Historical Notes:

- Dynamic Programming was popularized in the 1950's and 1960's by **Richard Bellman** (1920-1984), an American mathematician. Bellman, who coined the term Dynamic Programming, formulated the general theory and proposed numerous applications in control and operations research.
- **Andrew Viterbi** (born 1935) is an American professor of electric engineer, a pioneer in the field of digital communications, and co-founder of Qualcomm Inc. (together with Irwin Jacobs). He has had close ties with the Technion, and has made many contributions to our department.
- **Richard Karp** (born 1935) is an American professor of computer science, known for his contributions to the theory of computing.