ε.δε.μ²
ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## 1. Model-Free Methods in Reinforcement Learning

### Direct Approximations, Random Trajectories & State-Action Counting

### Temporal-Difference & Stochastic Q-Learning

## 2. Distributed Multi-Agent Reinforcement Learning

## 3. Bellman-Ford Algorithm, BGP Routing in the Internet


Prof. Vasilis Maglaris

maglaris@netmode.ntua.gr

www.netmode.ntua.gr

Room 002, New ECE Building


Tuesday April 29, 2025

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Summary of Dynamic Programming Definitions (1/2)

*D. P. Bertsekas* & *J. Tsitsiklis*, "**Neuro-Dynamic Programming**," Athena MA 1996

*R. S. Sutton* & *A. G. Barto*, "**Reinforcement Learning**," MIT Press 2018

### Dynamic Programming - Reinforcement Learning (RL) Definitions: Cost Minimization

***Observed Cost*** of transition step $i \to j$ with agent action $a$: $g(i,a,j)$

***Immediate Expected Cost*** of environment state $i$ and agent action $a$:

$$c(i,a) \triangleq \sum_{j=1}^{N} p_{ij} g(i,a,j)$$

***Cost-to-Go*** : $J^{\mu}(i) = c(i,\mu(i)) + \gamma \sum_{j=1}^{N} p_{ij}(\mu(i)) J^{\mu}(j)$ , $i = 1,2,\dots,N$ under policy $\mu(i)$

Optimal ***Cost-to-Go*** (***Bellman***): $J^{*}(i) = \min_{a \in \mathcal{A}_i} \left( c(i,a) + \gamma \sum_{j=1}^{N} p_{ij} J^{*}(j) \right)$, $i = 1,2,\dots,N$

Definition of ***Q-Factors***: $Q^{\mu}(i,a) \triangleq c(i,a) + \gamma \sum_{j=1}^{N} p_{ij}(a) J^{\mu}(j)$ for $\forall i$ and $\forall a \in \mathcal{A}_i$

### Dynamic Programming - Reinforcement Learning (RL) Definitions: Reward Maximization

***Observed Reward*** of transition step $i \to j$ with agent action $a$: $R(i,a,j)$

***Immediate Expected Reward*** of environment state $i$ and agent action $a$:

$$r(i,a) \triangleq \sum_{j=1}^{N} p_{ij} R(i,a,j)$$

***Value Function*** : $V^{\mu}(i) = r(i,\mu(i)) + \gamma \sum_{j=1}^{N} p_{ij}(\mu(i)) V^{\mu}(j)$ , $i = 1,2,\dots,N$ under policy $\mu(i)$

Optimal ***Values*** (***Bellman***): $V^{*}(i) = \max_{a \in \mathcal{A}_i} \left( r(i,a) + \gamma \sum_{j=1}^{N} p_{ij} V^{*}(j) \right)$, $i = 1,2,\dots,N$

Definition of ***Q-Factors***: $Q^{\mu}(i,a) \triangleq r(i,a) + \gamma \sum_{j=1}^{N} p_{ij}(a) V^{\mu}(j)$ for $\forall i$ and $\forall a \in \mathcal{A}_i$

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

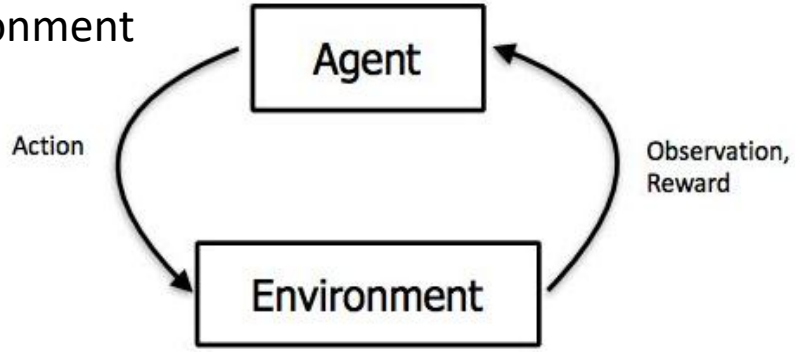
## Summary of Dynamic Programming Definitions (2/2)

### Model-based & Model-free Reinforcement Learning (RL)

***Nicolas Pröllochs*** & ***Stefan Feuerriegel***, *Business Analytics Practice* 2015/16
https://www.is.uni-freiburg.de/ressourcen/business-analytics/13_reinforcementlearning.pdf

- Algorithms based on ***Markov Decision Processes*** models of environment evolution with a-priori known transition probabilities $p_{ij}(a)$, are referred to as ***Model-based*** and involve ***Value Iteration*** or ***Policy Iteration*** methods (policy iteration usually converges faster than value iteration)

- ***Model-free*** methods search for ***Agent*** actions based on its growing understanding while at the training phase, **without** prior knowledge of the transition probabilities of the underlying ***Markov Decision Process***. Sate-Action assignment algorithms are based on observed (or ***Monte Carlo*** simulated) state ***trajectories*** of independent ***episodes*** (state evolutions) of the environment




***Steve Brunton***, University of Washington – YouTube 2022
Model Based RL: https://www.youtube.com/watch?v=sJIFUTITfBc
Model Free RL: https://www.youtube.com/watch?v=0iqz4tcKN58

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Direct Approximate Methods of Dynamic Programming (1/3)

- The ***Model-based*** Dynamic Programming Algorithms (***Value Iteration*** & ***Policy Iteration***) assume knowledge of **transition probabilities** $p_{ij}(a)$ and of the ***Immediate Expected Cost*** at state $i$ and action $a = \mu(i)$

$$c(i,a) = \sum_{j=1}^{N} p_{ij}(a) g(i,a,j)$$

**estimated** on the basis of ***observed costs*** of transitions $i \to j$ under an agent policy $\mu(i)$

$$g(i,a,j) = g(i,\mu(i),j) \triangleq g(i,j)$$

- The ***Model-free Direct Approximate Dynamic Programming Methods*** determine in every step the next transition $i \to j$ with agent action $a = \mu(i)$ with cost $g(i,a,j)$ and estimate the ***Immediate Expected Cost*** $c(i,a)$ as the mean value of $g(i,a,j)$ for all next states $j$, observed in independent ***trajectories*** that reached $i$ during the ***training phase***

***Model-free Direct Approximate Methods*** mainly apply to moderate size state-action space problems with no a-priori knowledge of transition probabilities. They correspond to model-based Dynamic Programming algorithms as follows:

- **Value Iteration** → **Temporal-Difference Learning**
- **Policy Iteration** → ***Q*-Learning**

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

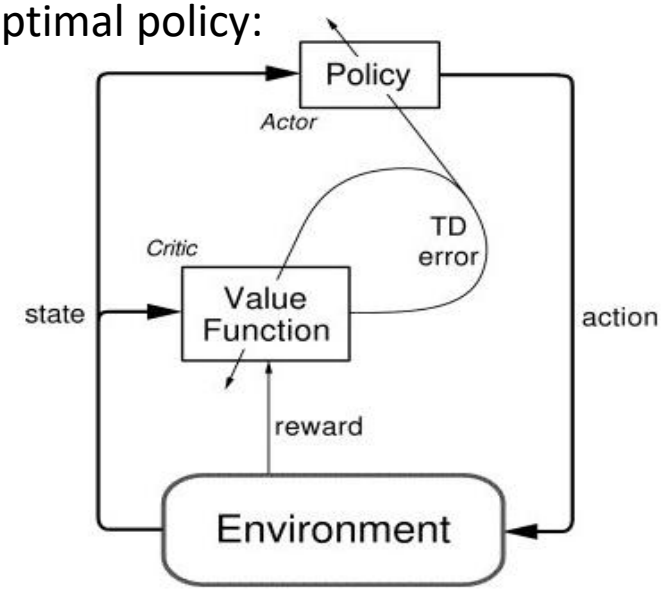
## Direct Approximate Methods of Dynamic Programming (2/3)

**Definitions**: ***on-policy***, ***off-policy***

- An ***on-policy*** estimates in every step (via measurements or ***Monte Carlo*** simulations) ***Costs-to-Go*** $J^{\mu}(i)$ of environment states $i$ reached with actions of the current policy μ. The frequency of state visits along ***trajectories*** resulting from μ provides estimates for $J^{\mu}(i)$, consistent with ***Costs-to-Go*** recursive equations. Subsequent minimization iterations aim to enforce the ***Bellman Equations***, thus yielding $J^{*}(i)$ of the optimal policy:

  **Value Iteration → *TD*(0)-Learning** (***on-policy***)

**Actor-Critic *TD*-Learning Model**: ***A.G. Barto***, ***R.S. Sutton*** & ***C.W. Anderson*** "**Neuronlike adaptive elements that can solve difficult learning control problems**," IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-13, Sept. – Oct. 1983




- The ***off-policy*** compares choices of actions $a$ for a state $i$ of a ***trajectory***, so that the agent can ***greedily*** select in the ***next*** step an action $a$ that minimizes ***Q-Factors*** $Q(i,a)$, without considering next states $i \rightarrow j$ if a policy is applied. The $Q(i,a)$ and the ***Cost-to-Go*** $J^{\mu}(i)$ of an intermediate policy μ are estimated by measurements (or ***Monte Carlo*** simulations) of up-to-now defined ***trajectories***. Future greedy decisions will result from updated ***Q-Factors***:

  **Policy Iteration → *Q*-Learning** (***off-policy***)

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Direct Approximate Methods of Dynamic Programming (3/3)

### Monte Carlo Simulation of System Trajectories

- Multiple scenarios of plausible ***system trajectories*** from an initial state $i_0$ to a terminal state $i_n \to i_T$ ($T$ – *Terminal*) can be generated via ***Monte Carlo*** simulations of the ***Markov Decision Process*** that fits our understanding of the environment

- For finite horizon cases $\{0,1, \dots, K\}$ we define as an **episode** the instance of a full trajectory $i_0 \to i_K$. Infinite horizon models are treated as limiting cases with large $K$

- The training process consists of ***Monte Carlo Sampling*** i.e. generation of multiple independent ***trajectories*** and counting the number of visits to state $i_n$, maintained in a **table** format. Trajectories may never visit states that are tough-to-reach or rarely visited; thus, sampling may enforce seemingly undesirable transitions to a next state $i_{n+1}$ in order to ***explore*** a wide range of the state-space (as in ***Simulated Annealing***)

- The ***cost-to-go*** functions $J^{\mu}(i)$ are updated by adding the ***observed costs*** $g(i,j)$ of transitions $i \to j$ under agent policy $a = \mu(i)$. These may be assumed (or approximated) from the problem specifications

***Monte Carlo*** methods require a good **understanding** of the environment by user/developer experience (due to lack of direct knowledge of transition probabilities), a **moderate** number of ***observable*** environment states and a **large number** of independent ***trajectories*** for reliable estimations. They may be simple but may require **extensive storage** for state-visit tables and **lengthy processing times** to converge, thus they do not scale to handle extremely large models (e.g. **LLM**s) that may suffer from the ***curse of dimensionality***

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Approximate Algorithm *TD*(0) Learning (1/2)

### Value Iteration → Temporal-Difference *TD*(0) Learning

- With repeated ***Monte Carlo*** simulations we sample $M$ ***trajectories*** of the environment from $i_n, n < T$ to a terminal state $i_T$ (we assume that $i_T = 0$) using some policy μ (**on-policy**). In each step we estimate ***Costs-to-Go*** $J^{\mu}(i_n)$ based on recursive equations:

$$J^{\mu}(i_n) = \mathrm{E}[g(i_n, i_{n+1}) + \gamma J^{\mu}(i_{n+1})] = \mathrm{E}\left[\sum_{k=0}^{T-n-1} \gamma^k g(i_{n+k}, i_{n+k+1})\right] = \mathrm{E}[c(i_n)]$$

- The $J^{\mu}(i_n)$ are approximated as ***ensemble averages*** of expected cost $J(i_n)$ along $M$ sub-trajectories $\{i_n, i_{n+1}, \dots, i_T\}$. A sub-trajectory cost is $c(i_n) \triangleq \sum_{k=0}^{T-n-1} \gamma^k g(i_{n+k}, i_{n+k+1})$ and for $M$ independent trajectories the average $J(i_n) = \mathrm{E}[c(i_n)]$ is estimated as

$$J(i_n) = \mathrm{E}[c(i_n)] \cong \frac{1}{M} \sum_{M} c(i_n)$$

- Costs $J(i_n)$ are evaluated using ***Robbins-Monro Successive Approximations*** in iterations that correct estimations by ***updates*** in every visit at $i_n$ of a trajectory that drives $i_n \to i_{n+1}$ with ***learning rate*** $\eta_n$:

$$J(i_n) \coloneqq J(i_n) + \eta_n[g(i_n, i_{n+1}) + \gamma J(i_{n+1}) - J(i_n)] = J(i_n) + \eta_n d_n$$

- The error $d_n \triangleq g(i_n, i_{n+1}) + \gamma J(i_{n+1}) - J(i_n), n = 0,1, \dots, T-1$ is referred to as the ***Temporal Difference -TD***, change in the expected cost in step (**time**) $n+1$ of a ***trajectory***. It drives $J(i_n)$ to convergence by minimizing $d_n$ in repetition of independent trajectories under a policy μ

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Approximate Algorithm *TD*(0) Learning (2/2)

### Value Iteration → Temporal-Difference *TD*(0) Learning

The ***cost-to-go update*** algorithm results from the ***recursive*** formula:

$$J(i_n) \coloneqq J(i_n) + \eta_n \left( \sum_{k=0}^{T-n-1} \gamma^k g(i_{n+k}, i_{n+k+1}) - J(i_n) \right) = J(i_n) + \eta_n \sum_{k=0}^{T-n-1} \gamma^k d_{n+k}$$

The ***costs-to-go*** are approximated as ***ensemble averages*** of $M$ independent trajectories that visit state $i_n$ at step $n$:

$$J^{\mu}(i_n) = \mathrm{E}\left[ \sum_{k=0}^{T-n-1} \gamma^k g(i_{n+k}, i_{n+k+1}) \right] = \mathrm{E}[c(i_n)] \cong J(i_n) = \frac{1}{M} \sum_{M} c(i_n)$$

where

$$c(i_n) \triangleq \sum_{k=0}^{T-n-1} \gamma^k g(i_{n+k}, i_{n+k+1})$$

The functions $J(i_n)$ are approximated by repeated visits to $i_n$ in ***observable Monte Carlo*** trajectories of $T$ transitions

$$J(i_n) \coloneqq J(i_n) + \eta_n \big( c(i_n) - J(i_n) \big)$$

with initial conditions $J(i_n) = 0$ and ***learning rate*** $\eta_n = {}^{1}/_{n}\,,\ n = 1,2,\dots,T$

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Approximate Algorithm *Q*-Learning (1/2)

**Review of Model Based Evaluation of $Q^*(i,a)$ with Successive Approximations (*Robins-Monro*)**

$$Q(i,a) \coloneqq (1-\eta)Q(i,a) + \eta \sum_{j=1}^{N} p_{ij}(a)\left[g(i,a,j) + \gamma \min_{b\in\mathcal{A}_j} Q(j,b)\right] \quad \forall(i,a)$$

The limit $Q^*(i,a)$ of the above successive approximations determines the optimal policy π as:

$$\mu^*(i) = \arg\min_{a\in\mathcal{A}_i} Q^*(i,a) \text{ for } i = 1,2,\dots,N$$

- **Model-free Implementation**: Determine ***off-policy behavior*** by processing of several ***trajectories*** generated for plausible decision scenarios: ***Q*-Learning**
- Define $s_n \triangleq (i_n, a_n, j_n, g_n)$ at the $n^{th}$ transition of a ***trajectory*** when the environment state is guided to $i_n \to i_{n+1} = j_n$ with the agent decision $a_n$ involving ***observed*** transition cost $g_n = g(i_n, a_n, j_n)$
- Based on registering $s_n$ in alternate ***trajectories*** the ***Q*-Learning** algorithm guides the system to the optimal policy as a variation of the ***policy iteration*** algorithm
- **Precondition**: States $i_n$ that form a ***trajectory*** must be ***fully observable***

## Approximate Algorithm *Q*-Learning (2/2)

### Stochastic Model-free Implementation via Monte Carlo Trajectory Sampling

- To avoid knowledge of $p_{ij}(a)$ we sample a ***trajectory*** from initial state $i_0$ to $i_n$ via ***Monte Carlo*** simulation based on of the environment behavior and using a ***behavior policy***
- In step $n$ the ***agent*** **estimates** ***Q-factors*** and the expected ***costs-to-go*** $J_n(j)$ for states $j$ resulting from a ***greedy estimation policy*** (different from the ***behavior policy*** that sampled the environment state trajectory, **off-policy**)
- The ***average*** in ***Q-factor*** evaluations, instead of considering next state $j$ that requires transition probabilities $p_{ij}(a)$ is approximated via observing targets and computing the average of multiple independent trajectories
- For each state-action pair $(i,a) = (i_n, a_n)$ visited in a ***trajectory***, its ***Q-factor*** is updated at the next step (**time**) $Q_n(i,a) \to Q_{n+1}(i,a)$:
  - $Q_{n+1}(i,a) = (1-\eta_n)Q_n(i,a) + \eta_n[g(i,a,j) + \gamma J_n(j)]$
  - $J_n(j) = \min_{b \in \mathcal{A}_j} Q_n(j,b)$ where $j = i_{n+1}$ is a follow-up state in the trajectory
  - $Q_{n+1}(i,a) = Q_n(i,a)$ for all other pairs $(i,a) \neq (i_n, a_n)$
  - After several rounds $Q_n(i,a) \to Q^*(i,a)$
  - The ***learning parameter*** $\eta_n$ is decreasing on $n$, e.g. $\eta_n = \alpha/(\beta + n)$ $\alpha, \beta > 0$

To avoid trapping of states within trajectories formed by ***greedy*** decisions (***exploitation***), we need to consider multiple trajectories of varying initial states that visit a wide range of states (***exploration***). The range of search can widen by enforcing ***greedy*** agent actions with probability $(\mathbf{1} - \boldsymbol{\epsilon})$ and other choices with probability $\boldsymbol{\epsilon}$

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Distributed Implementation of Reinforcement Learning

### Cooperative Optimization Model via Multi-Agent Reinforcement Learning - MARL

***Michael Littman***, 1994 https://www2.cs.duke.edu/courses/spring07/cps296.3/littman94markov.pdf

- Dynamic Programming as a ***cooperative zero-sum game*** among autonomous ***agents***
- Agents act towards optimal policies that may affect autonomous cooperators of a ***Markov*** (***Stochastic***) ***Game***
- Distributed ***Q-Learning Algorithm*** with ***asynchronous updates*** among ***agents***
- Definition of ***Q-factors*** as ***minimax Q-factors*** to model interacting decisions of cooperating ***agents***.
- Estimation of ***minimax Q-factors***, may be accomplished by repeated ***Linear Programs*** but with significant processing overhead. In practice and for specific applications it can be simplified by using fast, ***scalable*** heuristics

**A Succes Story**: Large scale application with $\sim \mathbf{83,200}$ ***agents/routers*** collaborating in the ***Border Gateway Protocol*** (***BGP***) to enable **global routing** among the $\sim \mathbf{1,200,000}$ ***advertised networks*** of the current (**2025**) ***Internet***

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (1/7)

**V. Maglaris**, **"*Network Management – Intelligent Networks*"** *Lectures*, *NTUA*-2024

https://www.netmode.ntua.gr/wp-content/uploads/2024/10/NetMan_Internet_Routing_BGP_Academic_Commercial_ISP_2024_10_07.pdf

The global ***Internet*** consists (*9/2024*) of $\sim 1,200,000$ reachable ***networks*** (e.g. the **NTUA** Local Network, **IP**: **147.102.0.0/16**), organized in $\sim 82,590$ ***Domains*** (***Autonomous Systems***, **AS**) e.g. **GRNET/ΕΔΥΤΕ**, Autonomous System Number – **ASN: 5408**

***Intra-domain*** routing is centrally controlled via ***Interior Gateway Protocols*** – ***IGP*** (e.g. **OSPF**). Among the $82,590$ AS's, ***Inter-domain*** routing is controlled in a distributed mode via ***Routing Tables*** in ***Border Gateways*** (***Border Routers***). These maintain path suggestions to **all** $\sim 1,200,000$ reachable networks of the ***Internet***

The ***inter-domain routing tables*** are stored in electronic fast memories of ***Border Gateways*** and comply by the ***Border Gateway Protocol*** - ***BGP*** standards **(RFC 4271)**. Their (hopefully) collaborative management relies on **82,590** Autonomous **Domain Administrators**

- The ***Border Routers*** (***Gateways***) of an ***AS*** announce (via ***BGP signaling***) to the $82,590$ ***Peer AS***'s of the ***Internet*** all $1,200,000$ networks – potential destinations under its control or are ***reachable*** through them, with cost estimates of end-to-end ***inter-AS*** paths
- The ***Border Gateways*** estimate autonomously optimal routes to all **1,200,000** destination networks based on administrator preferences (politics) and update routing choices in case of topology changes
- The distributed algorithm that estimates "optimal" paths to the $1,200,000$ potential destinations, conveys to BGP peers ***reachability*** information and interconnection costs to its ***immediate*** neighboring peers. Path cost are estimated via the ***Bellman – Ford Algorithm***

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (2/7)

### BGP Distance Vector Algorithm (Bellman – Ford)

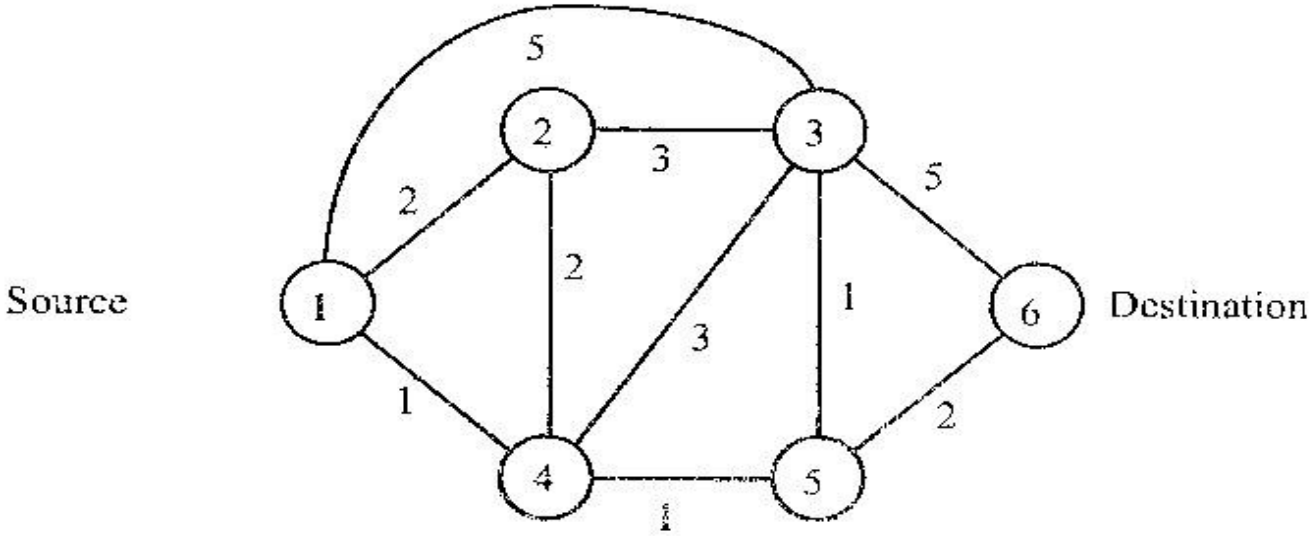
- The ***Border Gateways*** of an ***AS*** identify ***shortest paths*** of transit and final ***AS***'s to all advertised destination networks via a ***dynamic programming*** version
- The algorithm requires measurements of costs (weights) of direct ***Inter AS Interfaces*** (links) and estimates of ***distance vectors*** to all networks, potential destinations of the ***Internet*** ($\sim \mathbf{1,200,000}$ - *9/2024*)
- It is based on distributed ***Bellman - Ford*** dynamic programming and makes use of ***BGP Announcements*** among all $\sim \mathbf{82,590}$ ***AS***'s of the global ***Internet*** that convey routing information and cost vectors
- Within the context of ***Reinforcement Learning***, ***BGP*** is a distributed version of Dynamic Programming, a ***cooperative game*** of $\mathbf{82,590}$ ***Autonomous Agents***

**BGP is a major enabler for the success story of the *Internet***

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (3/7)

**Example of a Network (Graph) with $N = 6$ Nodes (Autonomous Systems)**

- The graph nodes represent ***AS***'s of a very limited scope of the ***Internet***
- Source and destination networks are imbedded within the graph nodes
- Links (graph edges) depicted in the figure refer to both directions based on measurements of adjacent domain administrators
- In this example ***shortest path trees*** are evaluated from all nodes $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$ to the ***root*** node $\{6\}$
- The root role as ***destination*** was arbitrarily chosen. The algorithm holds for any choice of node role, as destination or source



# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (4/7)

## Evaluation of Shortest Path Tree Rooted in $\{6\}$

### *Q*-Learning (*Off-policy*) with *Asynchronous Updates*

$\{i\}$ **State**: Node (***AS***) $i = 1,2, \dots, N$ ($N = 6$, in the example, over $82{,}000$ in the ***Internet***)

$P^{(n)}(i)$ **Action**: Next node in a path (trajectory) from $\{i\}$ to $\{6\}$, ετransit or final in ***Iteration*** $n$

$d_{ij}$ **Cost** (weight) of edgeς $(i, j)$ set by the routing policy of autonomous agent $\{i\}$ and/or via measurements by directly connected neighbors $\{i, j\}$. If $d_{ij} = c, \forall (i, j) \Rightarrow$ ***min hop routing***

$L^{(n)}(i)$ ***Labels***, ***Q-Factors*** $L^{(n)}(i) \triangleq Q(i, P^{(n)}(i))$ : Minimum cost from $\{i\}$ to $\{6\}$ in the $n^{th}$ iteration (asynchronously updated according the most recent ***estimates*** and the ***update*** order of ***Q-Factors***). The ***trajectories*** αare path choices from $\{i\}$ to $\{6\}$ in every iteration

## The Bellman - Ford Algorithm

- Initialize $L_i^{(0)} = \infty \ \forall i \neq 6, \ L_6^{(n)} = 0 \ \forall n,$
- In every ***iteration*** $n = 1,2, \dots$ and $\forall i$ we ***asynchronously*** update minimum cost estimates for paths from the current state to the destination $\{6\}$ based on ***Bellman***'s recursive equations to evaluate ***Q-Factors*** $L_j^{(n)}$ for all directly connected neighbors $j$ of $i$:

$$L_i^{(n+1)} = \min_j \left\{ L_j^{(n)} + d_{ij} \right\} \forall i \neq 6$$

- If $L_i^{(n+1)} = L_i^{(n)} \ \forall i$ stop and determine optimal paths as a ***Shortest Path Tree*** rooted to $\{6\}$ according to actions $P^{(n)}(i)$
- Algorithm complexity: $O(N^3)$

# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (5/7)

## Execution of Algorithm for Destination{6}

INITIAL LABELS: L(1)=L(2)=...=L(5)= ∞, L(6)=0

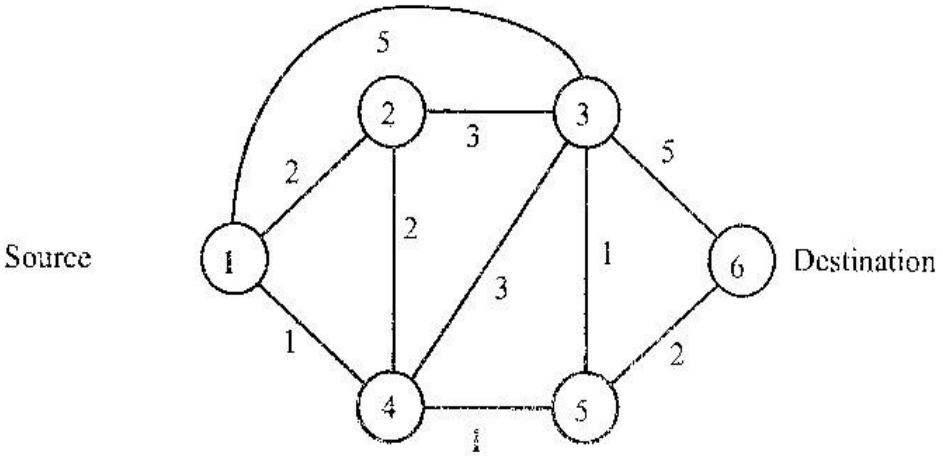
**UPDATE ORDER 5,4,3,2,1**

| Iteration Number | Labels L(n), Current Predecessor Node P(n) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | L(5), P(5) | | L(4), P(4) | | L(3), P(3) | | L(2), P(2) | | L(1), P(1) | |
| 1 | 2 | 6 | 3 | 5 | 3 | 5 | 5 | 4 | 4 | 4 |
| 2 | 2 | 6 | 3 | 5 | 3 | 5 | 5 | 4 | 4 | 4 |

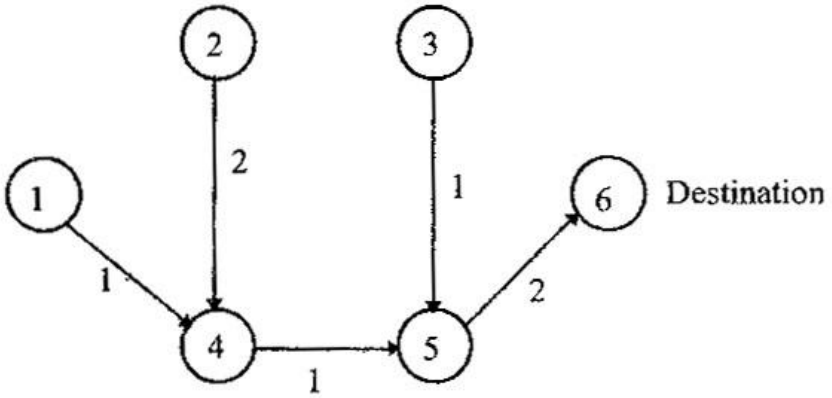
**UPDATE ORDER 1,2,3,4,5**

| Iteration Number | Labels L(n), Current Predecessor Node P(n) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | L(1), P(1) | | L(2), P(2) | | L(3), P(3) | | L(4), P(4) | | L(5), P(5) | |
| 1 | ∞ | - | ∞ | - | 5 | 6 | **8** | **3** | 2 | 6 |
| 2 | **9** | **4** | 8 | 3 | 3 | 5 | 3 | 5 | 2 | 6 |
| 3 | 4 | 4 | 5 | 4 | 3 | 5 | 3 | 5 | 2 | 6 |
| 4 | 4 | 4 | 5 | 4 | 3 | 5 | 3 | 5 | 2 | 6 |

**The speed of convergence depends on the update order of Node Labels**



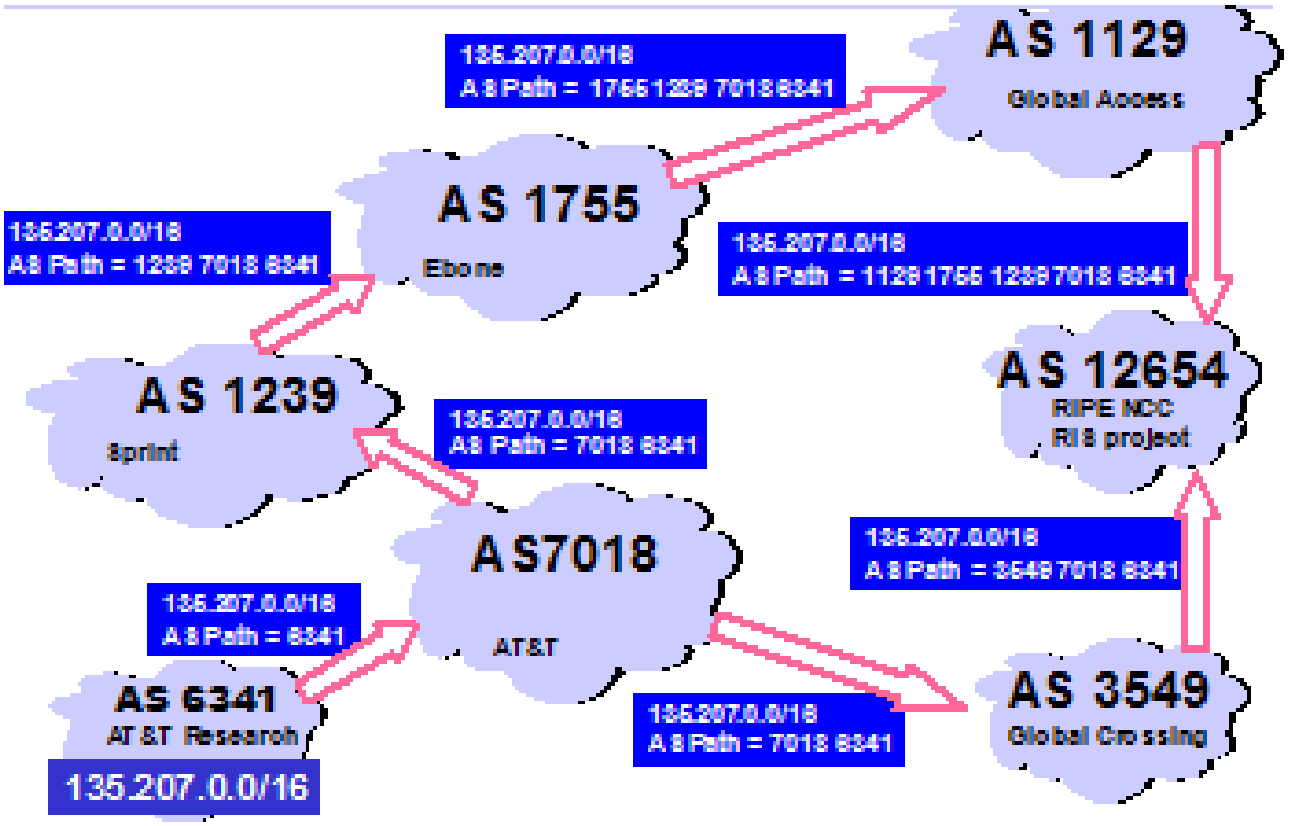

**SHORTEST PATH TREE**



## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (6/7)

### Example of Path Reachability Learning - Advertisement of Network 135.207.0.0/16

(*Timothy G. Griffin, AT&T Research, Paris 2002*)

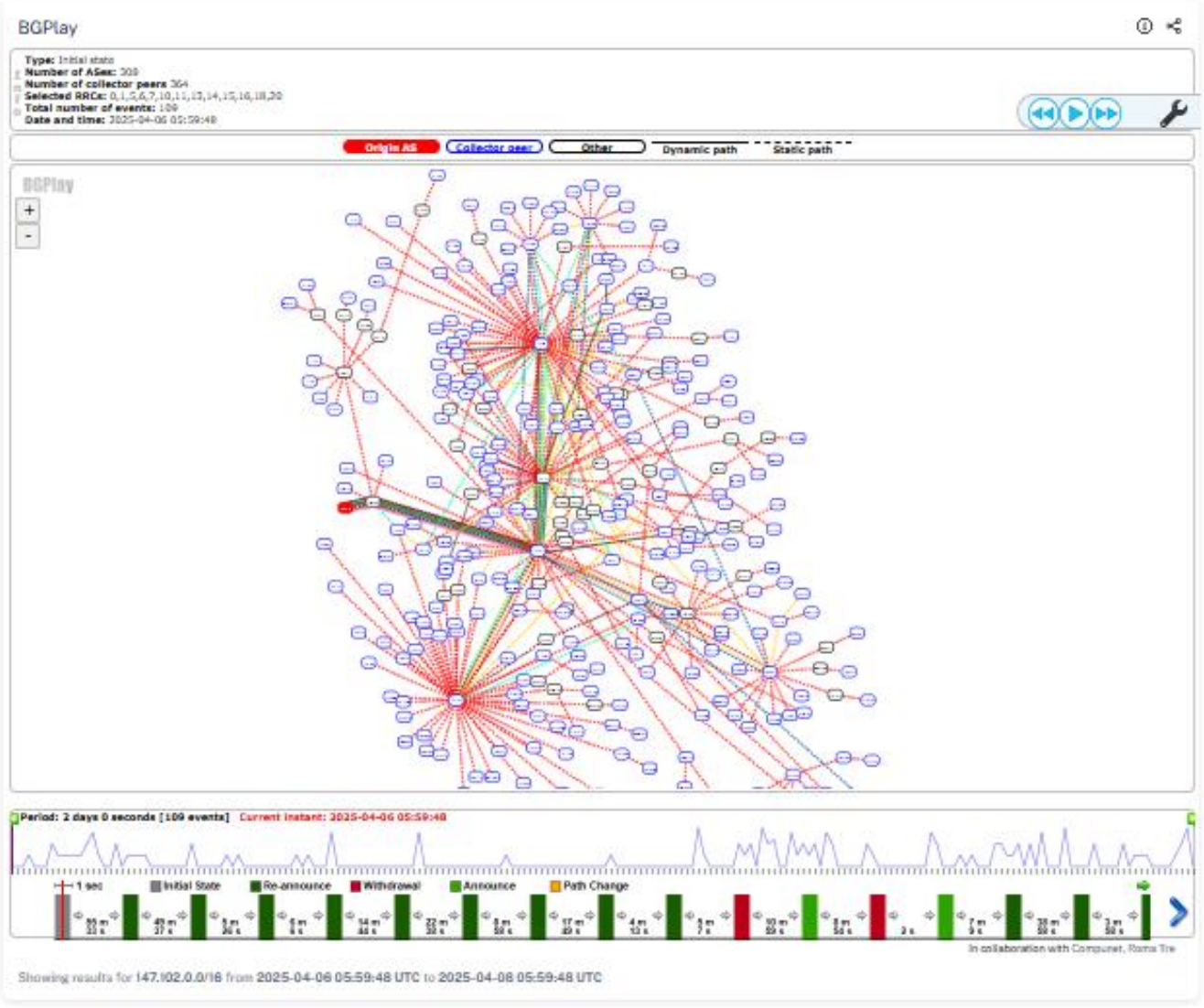


# STOCHASTIC PROCESSES & OPTIMIZATION IN MACHINE LEARNING

## Dynamic Programming Example: Internet Border Gateway Protocol (BGP) - RFC 4271 (7/7)

## BGP Routes to ntua.gr - 147.102.0.0/16 (8/4/2025)

https://stat.ripe.net/widget/bgplay




**NTUA (AS: 3323)**
**GRNET (AS: 5408)**
**GÉANT (AS: 21320)**

**GÉANT Internet Feeds**

- **LEVEL3 (AS: 3356)**
- **COGENT 174 (AS: 174)**
- **NORDUnet (AS: 2603)**