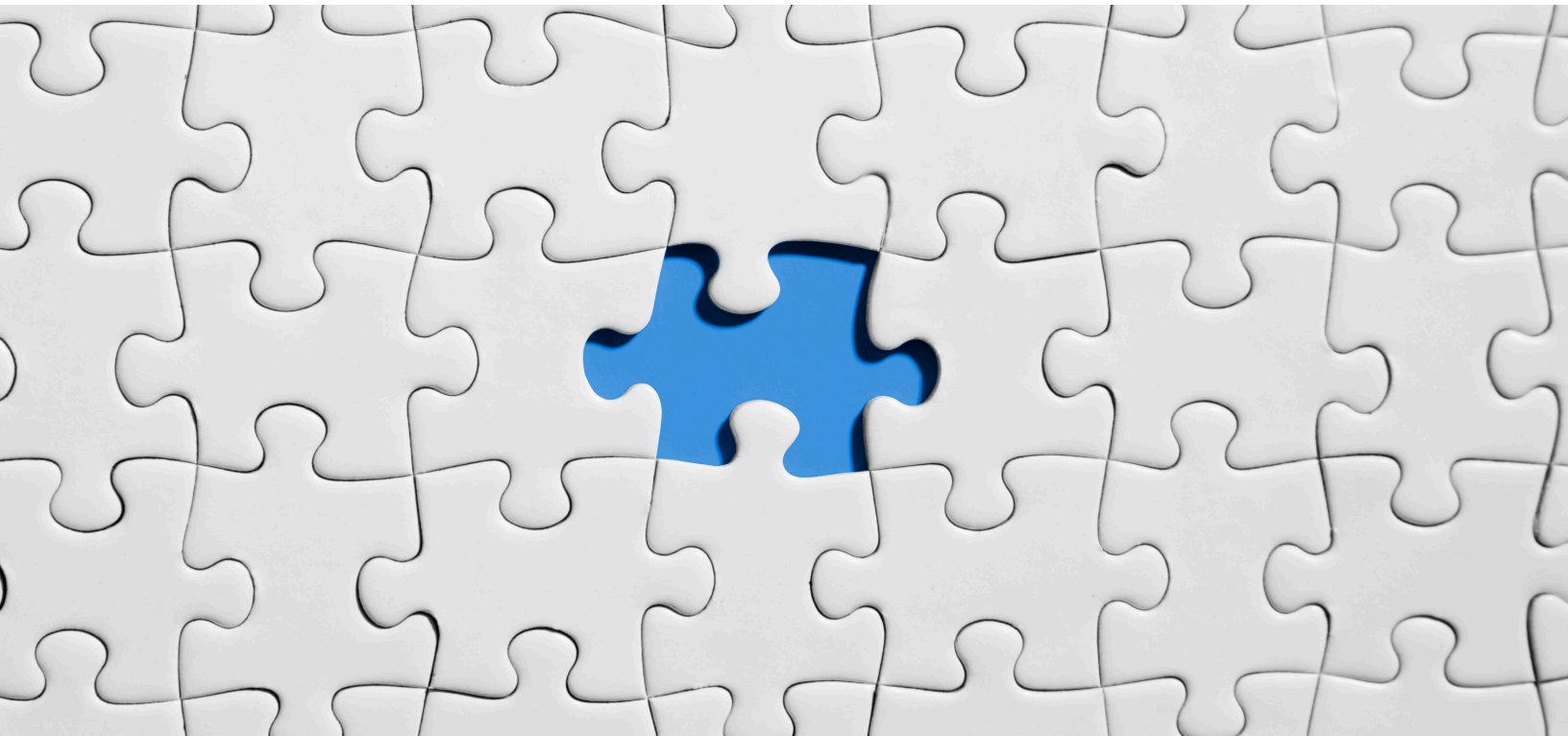


# OPTIMAL PATH ROUTING USING REINFORCEMENT LEARNING



## Rajasekhar Nannapaneni

Sr Principal Engineer, Solutions Architect

Dell EMC

[Rajasekhar.nannapaneni@dell.com](mailto:Rajasekhar.nannapaneni@dell.com)



The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

[Learn more at www.dell.com/certification](http://www.dell.com/certification)

## **Abstract**

Optimal path management is key when applied to disk I/O or network I/O. The efficiency of a storage or a network system depends on optimal routing of I/O. To obtain optimal path for an I/O between source and target nodes, an effective path finding mechanism among a set of given nodes is desired.

In this article, a novel optimal path routing algorithm is developed using reinforcement learning techniques from AI. Reinforcement learning considers the given topology of nodes as the environment and leverages the given latency or distance between the nodes to determine the shortest path.

Thus, this algorithm enables quick optimal shortest paths for a given topology of a storage or a network system. The algorithm developed in this article can be easily extended to software defined networks (SDNs), storage area networks (SANs), backup infrastructure management, distributing computing such as blockchain and other applications where optimal path behavior is required.

## Table of Contents

Abstract .....	3
List of Figures.....	5
Glossary .....	5
A1.1 Introduction.....	6
A1.2 What is Reinforcement Learning?.....	6
A1.3 Reinforcement Learning Outlook.....	8
A1.4 Applications of Reinforcement Learning.....	9
B1.1 Optimal Path Routing .....	10
B1.1.1 Distance (or Cost) .....	10
B1.1.2 Latency (or Delay).....	11
B1.2 Shortest Path Strategies .....	11
C1.1 Evolution of Reinforcement Learning Algorithms .....	12
C1.2 Markov Decision Processes .....	13
C1.3 State Value function and Bellman equation.....	13
C1.4 Dynamic Programming .....	14
C1.5 Monte Carlo technique.....	15
C1.6 Temporal Difference Learning .....	15
C1.6.1 SARSA (on policy).....	16
C1.6.2 Q-Learning (off policy).....	16
D1.1 Modified Q-learning for optimal path routing.....	16
D1.1.1 Modified Q-learning Algorithm .....	17
D1.2 Implementation of optimal path routing.....	17
D1.3 Conclusion .....	21
References .....	22
APPENDIX.....	23

## List of Figures

Figure No.	Title of the figure	Pg.No.
Figure 1	Types of Machine Learning	6
Figure 2	Reinforcement Learning	7
Figure 3	Analogy for Reinforcement Learning	8
Figure 4	Applications of Reinforcement Learning	9
Figure 5	Latency incurred by a cloud provider service	10
Figure 6	Internet traffic for an online gaming application	11
Figure 7	Evaluation of DFS and BFS algorithms	12
Figure 8	Q-learning equation	17
Figure 9	Q-learning table	17
Figure 10	1 <sup>st</sup> Optimal Path Network	18
Figure 11	2 <sup>nd</sup> Optimal Path Network	19
Figure 12	3 <sup>rd</sup> Optimal Path Network	20

## Glossary

ISP	Internet Service Provider
MC	Monte Carlo
MDP	Markov Decision Process
RL	Reinforcement Learning

Disclaimer: The views, processes or methodologies published in this article are those of the author. They do not necessarily reflect Dell Technologies' views, processes or methodologies.

## **A1.1 Introduction**

In the world of distributed computing, it is essential to have an optimum routing mechanism in place to reduce latencies especially when the environment is dynamic. The same thing applies to dynamic network nodes, storage paths, etc. where either more and more nodes are getting modified real-time or the load, latency and cost of the path changes dynamically. Static path definitions and/or rule-based routing methods are not efficient in these scenarios.

There is a need for adaptive dynamic routing mechanism that can adapt to any dynamic scenario be it path management, network routing or distributed computing such as blockchain. Adaptive optimal path management can also enhance routing mechanism for storage I/O or network I/O given a set of nodes.

With the advent of artificial intelligence (AI), one can leverage its algorithms for optimal path routing. This article attempts to develop an optimum path routing algorithm that can dynamically find the shortest path to a given network topology.

## **A1.2 What is Reinforcement Learning?**

In the field of machine learning, the prominent learning tasks are supervised learning and unsupervised learning. Supervised learning tasks use the labelled training data and predict the testing data while unsupervised learning finds the patterns in the given testing data and tries to understand the insights in data without any training.

Figure 1 shows the three variants of machine learning schemes; supervised learning, unsupervised learning and reinforcement learning.

Reinforcement learning is relatively a different paradigm in machine learning which can guide an agent how to act in the world. The interface to a reinforcement learning agent is much broader than just data; it can be the entire environment. That environment can be the real world, or it can be a simulated world such as a video game.

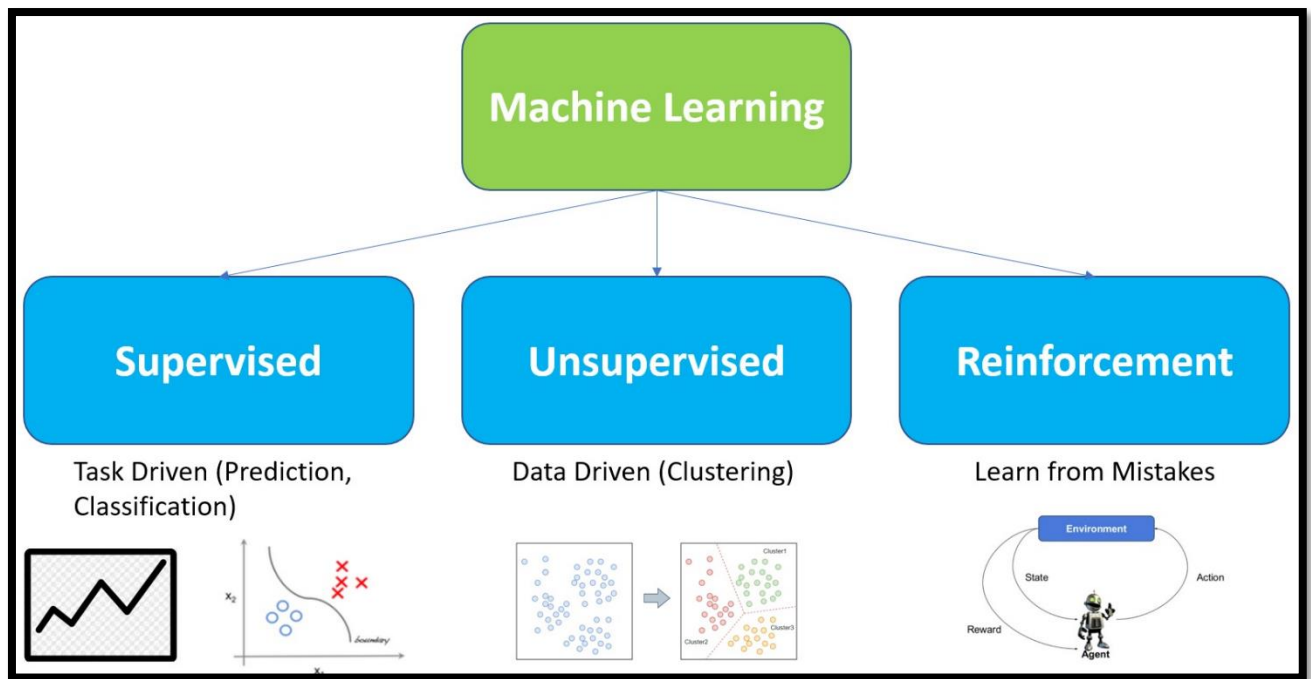


Figure 1: Types of Machine Learning

Here are a couple of examples. A reinforcement learning agent can be created to vacuum a house which would be interacting with the real world. Likewise, a reinforcement learning agent can be created to learn how to walk which also would be interacting with the real world. These are interesting approaches for the military domain who want reinforcement learning agents to replace soldiers not to only walk but fight, diffuse bombs and make important decisions while out on a mission.

Reinforcement learning algorithms have objectives in terms of a goal in the future. This is different from supervised learning where the objective is simply to get good accuracy or to minimize the cost function. Reinforcement learning algorithms get feedback as the agent interacts with its environment so feedback signals or – as we like to call them – rewards are given automatically to the agent by the environment.

There are a couple more things to define that is central to reinforcement learning. First there is the concept of reward where an agent will try to maximize not only its immediate reward but future rewards as well. Often reinforcement learning algorithms will find novel ways to accomplish this. Then there's the concept of state which are different configurations of the environment that the agent consents. The last important concept we need to talk about is actions, what the agent does in its environment.

In essence, in reinforcement learning, the agent is programmed to be an intelligent agent which interacts with its environment by being in a state taking an action based on that state. Based on the

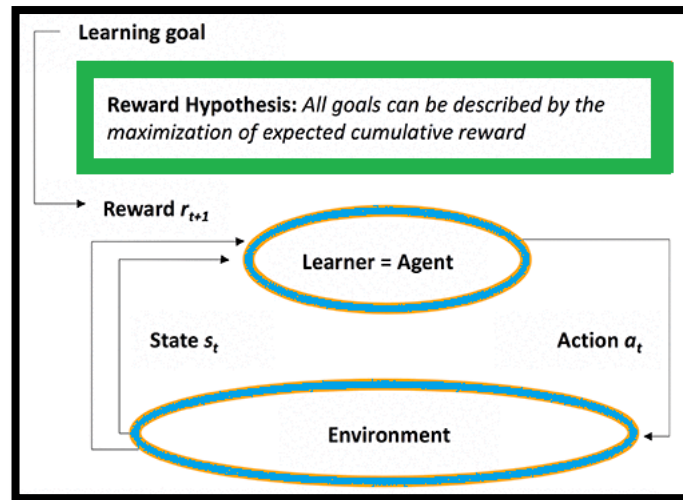


Figure 2: Reinforcement Learning

action the agent moves to another state in the environment and obtains a reward either positive or negative. The goal of the agent is to maximize its total rewards. Figure 2 illustrates that an agent is the learner that interacts with the environment represented as a state space and constantly receives rewards at every time step for every action it took in the previous state.

### A1.3 Reinforcement Learning Outlook

Thinking of AI and machine learning conjures up grand ideas, i.e. machine learning algorithms would enable building a robot that takes out the trash and does things which are shown in sci-fi movies. However, machine learning is not quite like that.

Machine learning is capable of several supervised and unsupervised learning-based tasks that fall under use cases such as classification, regression, clustering, etc. but today we know that AI is capable of so much more.

Just a few years ago Alpha-GO defeated a world champion in the strategy game GO, something that the world's leading experts didn't think would happen for at least another decade.

New AI's don't even have to learn from playing with other humans. They can become superhuman simply playing against themselves. But AI doesn't stop at being superhuman in simple strategy games, it has been performing at or better than human levels at video games.

One of the great achievements of AI will be robot navigation in the real physical world. Walking is a task that most humans learn to do as infants, but our limitation is that when a infant is born, he or she must



learn to walk from scratch. AI wouldn't have such a limitation since it could just copy the walking program from an AI which already knows how to walk. In other words, AI can build on previous discoveries much more easily than humans can.

Figure 3 provides an analogy to reinforcement learning in a simulated car game. The road, buildings, signals, etc., are the environment state space and the car is considered the learner agent which takes actions shown as per the rewards or penalties it receives from the environment.



Figure 3: Analogy for Reinforcement Learning

Reinforcement learning deals with all these use cases where an agent is programmed to interact with an environment and aims at achieving maximum reward focusing on long term returns. This new paradigm of learning is interesting and has many deeper applications in future.

#### A1.4 Applications of Reinforcement Learning

Reinforcement Learning has a number of different applications which are increasingly becoming popular. Most of its applicability can be seen in simulated game environments like Atari games where it has surpassed human learning capabilities. Reinforcement learning can also be applied in the financial domain where it can be used to optimize the investment portfolios to achieve higher rewards.

Robotics is another practical application where reinforcement learning is popular. The agent in this case will be a robot which will interact with the environment and get rewards based on its actions. Figure 4

highlights some of the areas where reinforcement learning can be applied. One such application falls under Computer Systems where reinforcement learning can be used to optimize network routing which is the main theme of this article.

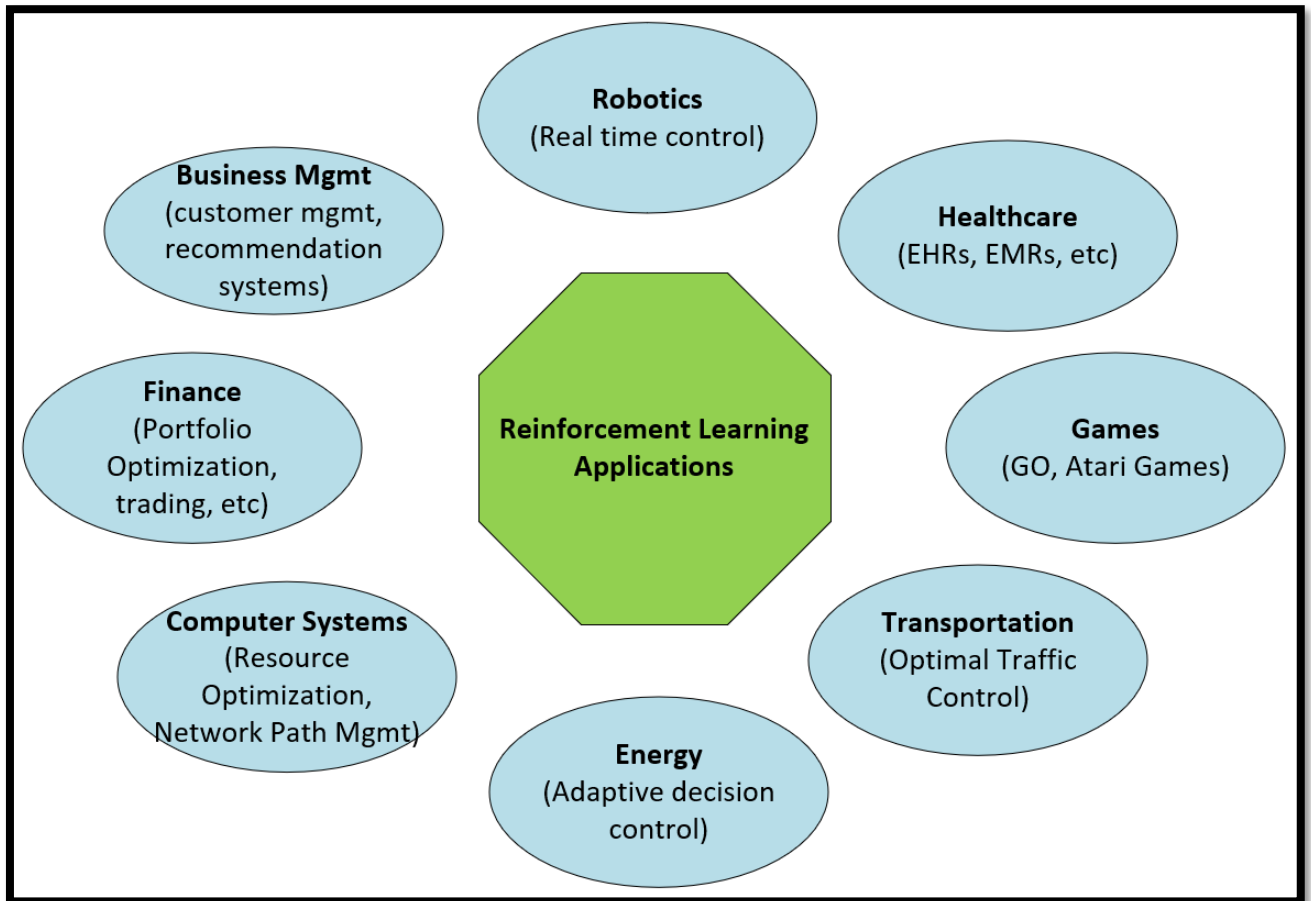


Figure 4: Applications of Reinforcement Learning

## B1.1 Optimal Path Routing

Routing is a mechanism needed in several domains, be it vehicular routing, distributed computing, network routing, and more. Network routing is required by internet service providers, cloud service providers, distributed blockchain networks, etc. Some of the parameters which are critical in finding optimal path routing are distance, latency, and cost. [8]

### B1.1.1 Distance (or Cost)

Distance or cost associated with a given path is one of the primary parameters to determine the shortest path. Any algorithm designed should consider these as important input. Though it appears counter-intuitive, it is possible that a shorter path may have high cost, so appropriate decision must be taken based on other parameters impacting optimality.

### B1.1.2 Latency (or Delay)

Optimality in routing can increase the latency or delay incurred on a packet to traverse from source to destination. When considering network routing, think of cloud provider service latencies across various cities in the world. Figure 5 shows network latencies incurred by a popular cloud service provider.

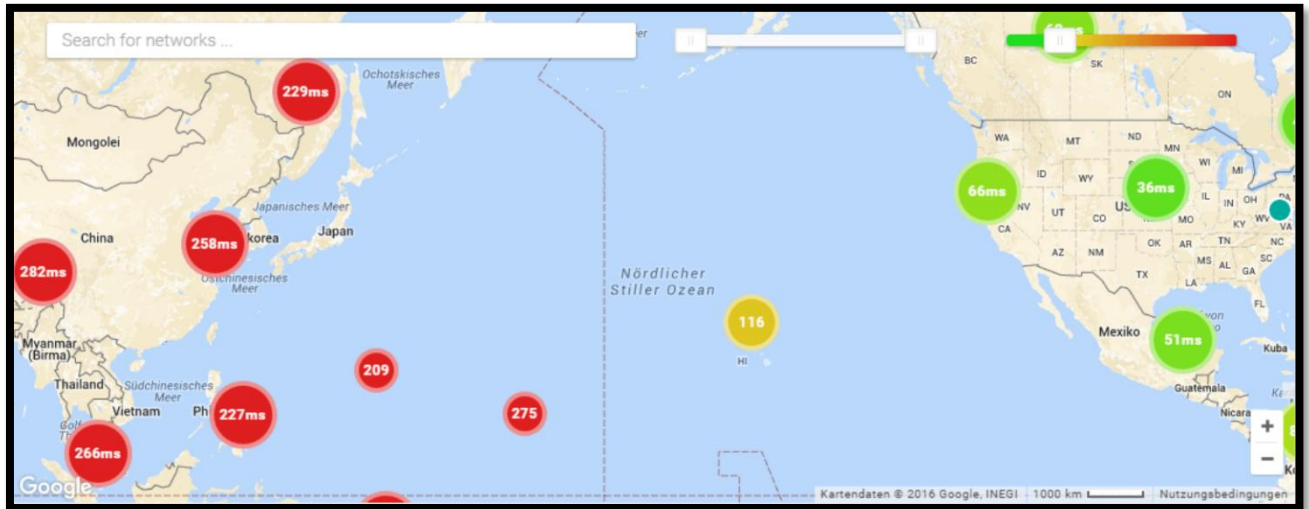


Figure 5: Latency incurred by a cloud provider service

It's important to have an optimal route for network packets to reduce delay incurred due to inefficient routes. Also consider that these network latencies might not remain constant throughout but may change over time.

### B1.2 Shortest Path Strategies

Several shortest path strategies exist, ranging from static, dynamic, adaptive, deterministic, rule-based, model based, model free, etc. These strategies can be evaluated based on complexity, computational needs, time/delays, memory needs, optimality, energy needs, and so on. Most of these strategies take decisions based on a tradeoff between latency and cost. Figure 6 illustrates the shortest path versus lowest latency path between an online gaming application and player. The decision to choose one or the other depends on cost, computational needs, etc. [9]

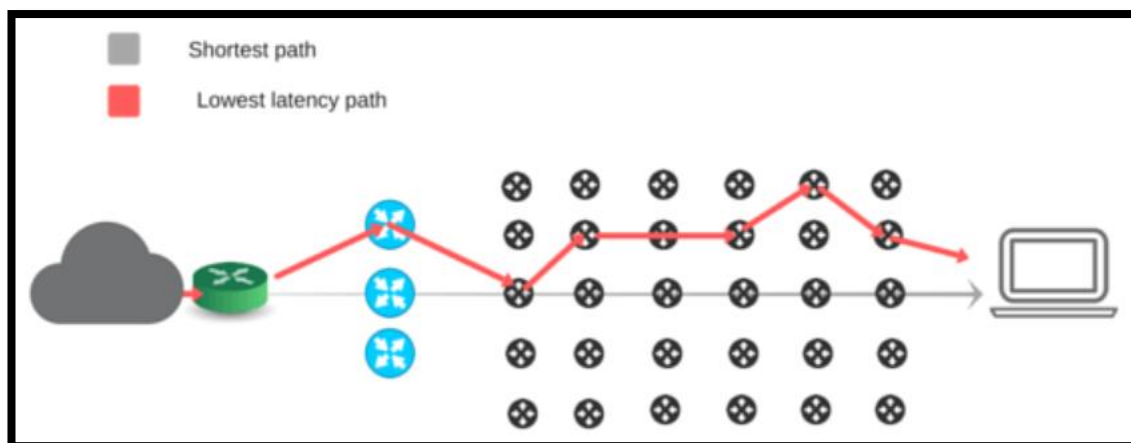


Figure 6: Internet traffic for an online gaming application

The Dijkstra algorithm, a popular algorithm to find shortest paths which is used in IP routing, Google maps and other applications, has its drawbacks in that it can be applied to only topologies that adhere as acyclic graphs. For other network topologies that have negative edges, it fails to provide shortest path. The order and complexities of other search algorithms listed in Figure 7 convey that these algorithms become very inefficient as the network topology increases.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

Evaluation of tree-search strategies.  $b$  is the branching factor;  $d$  is the depth of the shallowest solution;  $m$  is the maximum depth of the search tree;  $l$  is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

Figure 7: Evaluation of DFS and BFS algorithms

This creates the need for an efficient optimal adaptive routing algorithm.

### C1.1 Evolution of Reinforcement Learning Algorithms

Reinforcement learning undergoes an exploration-exploitation dilemma where an agent must tradeoff exploring the various possible state-action pair in the environment for maximum rewards versus exploiting already found best state-action pair. The explore-exploitation dilemma can be addressed through algorithms such as epsilon-greedy, UCB1 and Thomson sampling, etc.

## C1.2 Markov Decision Processes

The concepts of reinforcement learning can be put into a formal framework called Markov Decision Processes, a five-tuple made up of the set of states, set of rewards, set of actions, the state-transition probabilities, and the reward probabilities, which we previously discussed as a joint distribution. [5]

The Markov property, a core component of the Markov Decision Process, specifies how many previous states ( $x$ 's) the current  $x$  depends on. So, first-order Markov means  $x(t)$  depends only on  $x(t-1)$ .

Markov property is defined as:

$$p\{x_t | x_{t-1}, x_{t-2}, \dots, x_1\} = p\{x_t | x_{t-1}\}$$

Markov property in reinforcement learning is given by:

$$p\{S_{t+1}, R_{t+1} | S_t, A_t, S_{t-1}, A_{t-1}, \dots, S_0, A_0\} = p\{S_{t+1}, R_{t+1} | S_t, A_t\}$$

That is, doing an action  $A(t)$  while in state  $S(t)$  produces the next state  $S(t+1)$  and a reward  $R(t+1)$ . In this case the Markov property says,  $S(t+1)$  and  $R(t+1)$  depend only on  $A(t)$  and  $S(t)$  but not any  $A$  or  $S$  before that. Please note that the state transition probabilities are deterministic, and the environment is stationary.

The way it makes decisions for what actions to do in what states is called a policy. A policy that can give maximum returns of all the actions that are possible for each of the states is called optimal policy.

A fundamental concept in reinforcement learning is value function which is also central to just about every reinforcement learning algorithm we will encounter. Value function at state ' $s$ ' is the expected average sum of all the rewards in future given. Currently, we are state ' $s$ '.

## C1.3 State Value function and Bellman equation

The state value function can be written as:

$$V_\pi(S) = E_\pi[G(t) | S_t = s]$$

where  $G(t)$  is sum of all the future rewards at state ' $s$ ' and ' $\pi$ ' is the policy.

The state value function has only one input parameter-state which may not be feasible for all environments and hence another function called action-value function can be defined over two input parameters, actions and states as:

$$Q_\pi(s, a) = E_\pi[G(t) | S_t = s, A_t = a],$$

where  $G(t)$  is sum of all the future rewards at state ' $s$ ' when action ' $a$ ' is taken following ' $\pi$ ' is the policy.

Everything that needs to be known about future rewards is already encapsulated in the value function. This recursive equation has a special place in reinforcement learning and it's the equation from which all the algorithms are based. Named the Bellman equation after the famous mathematician Richard Bellman, it's related to recursion but it's efficient since it builds up the solution using a bottom up approach. In fact, one of the techniques that can solve Markov Decision Processes is called Dynamic Programming.

The Bellman equation is given by:

$$V_{\pi}(s) = \sum \pi(a|s) \sum p(s', r|s, a) \{r + \gamma V_{\pi}(s')\}$$

Where ' $\pi$ ' is the policy,  $s'$  is the future state,  $r$  is the reward at  $s'$  state,  $a$  is action at  $s$  state, and  $\gamma$  is the discount factor applied to rewards in the long run.

### **C1.4 Dynamic Programming**

Dynamic programming is one of the solutions to solve the Markov Decision Processes. One of the main objectives in reinforcement learning is to evaluate the value function and the Bellman Equation can be used directly to solve for the value function. It can be seen as a set of  $S$  equations with  $S$  unknowns which are linear equations and usually solved through matrix algebra. However, a lot of matrix entries would be 0 and hence the other ways to solve them is through policy evaluation. The input into iterative policy evaluation is a policy  $\pi$  and the output is the value function for this policy. The iterative policy evaluation will recursively converge and this act of finding value function is called prediction problem in reinforcement learning.

Using the action-value function, one can recursively find the best policy or, in other words, optimal policy using the value function determined in iterative policy evaluation, called policy iteration. This act of finding the optimal policy is called control problem in reinforcement learning.

One disadvantage of policy iteration is that it may not be feasible in large state spaces due to its iterative nature which has cascaded iterations. An alternative approach to it is called value iteration which combines policy evaluation and policy improvement into one step. This slightly modifies the Bellman equations as:

$$V_{k+1}(s) = \max \sum p(s', r|s, a) \{r + \gamma V_k(s')\}$$

This is also an iterative procedure. However, we don't need to wait for  $k$ th iteration of  $V$  to finish before calculating  $(k+1)$  term.

To summarize, dynamic programming is one of the methods to find solutions to MDP and does this by solving prediction and control problems iteratively.

## C1.5 Monte Carlo technique

Dynamic programming requires full model of the environment as it requires the state transition probabilities  $p(s', r | s, a)$  where as in the real world, it is difficult to obtain or evaluate the full model of the environment. Another constraint with dynamic programming is that it requires initialization of estimates – called bootstrapping – which may not be feasible all the time.

Monte Carlo, another technique to solve Markov decision processes, is model free-based and learns purely by its experience. Monte Carlo refers to any algorithm that involves a significant random component in reinforcement learning. With Monte Carlo methods, instead of calculating the true expected value which requires probability distributions, a sample mean can be taken instead.

Here tasks are episodic in nature as episodic tasks allow calculation of returns once episode is terminated. This also means Monte Carlo methods are not fully online algorithms and updates are not done after every action but rather after every episode.

Prediction problem: Instead of using  $V_{\pi}(S) = E_{\pi}[G(t) | S_t = s]$ , we will use  $V_{\pi}(S) = \frac{1}{N} \sum G$  which will be sample mean.

Control problem: Since  $V(s)$  is there for a given policy, it is not known what actions will lead to  $V(s)$  and hence one can use state-action value function  $Q(s,a)$ . Policy iteration has limitations as it does iteration inside iteration and hence the control problem is solved through value iteration where  $Q$  is updated, and policy improvement is done after every episode. Thus, it does one episode per iteration.

Recall that one of the disadvantages of dynamic programming is that one must loop through the complete set of states on every iteration and that this is bad for most practical scenarios in which there are many states. Monte Carlo only updates the value for states that are visited. This means even if the state base is large, it doesn't matter as it will only ever visit a small subset of states.

Also notice that with Monte Carlo, it doesn't need to know what the states are. It can simply discover them by playing the game. So, there are some advantages to Monte Carlo in situations where doing full, exact calculations is not feasible.

## C1.6 Temporal Difference Learning

Monte Carlo methods can only update after completing an episode and is not fully online. Temporal difference (TD) learning on the other hand used bootstrapping like dynamic programming and can update value during an episode while it doesn't need full model of the environment like dynamic programming does. Thus, it combines the benefits of both Monte Carlo and dynamic programming.

Prediction problem: Temporal difference learning uses TD(0) method to solve prediction problem. In TD(0), instead of sampling the return, it can use the recursive definition:

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$$

In TD(0), there is another source of randomness; G is an exact sample in MC and  $r + \gamma V(s')$  is itself an estimate of G.

Control problem: Similar to MC techniques, we use  $Q(s,a)$  in value iteration to solve control problem.

**C1.6.1 SARSA (on policy)** The method of using  $Q(s,a)$  using the 5 tuple  $(s,a,r,s',a')$  is called SARSA and the value iteration is solved using the equation:

$$Q(s, a) < - Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

**C1.6.2 Q-Learning (off policy)** This is another method to solve control problem under Temporal differencing. The main theme is to perform generalized policy iteration (policy evaluation + policy improvement) which is based on the equation:

$$Q(s, a) < - Q(s, a) + \alpha[r + \gamma \max Q(s', a') - Q(s, a)]$$

If the policy used for Q-learning is greedy, SARSA and Q-Learning will be done at the same time.

Unlike dynamic programming, a full model of the environment is not required in Temporal differencing. Agent learns from experience and it updates only the states that it visits. Unlike Monte Carlo, it doesn't need to wait for an episode to finish before it can start learning. This can be advantageous in situations where there are very long episodes. Agent can improve its performance during the episode itself rather than having to wait until the next episode. It can even be used for continuing tasks in which there are no episodes at all.

### D1.1 Modified Q-learning for optimal path routing

The Q-learning off policy algorithm was given by the equation:

$$Q(s, a) < - Q(s, a) + \alpha[r + \gamma \max Q(s', a') - Q(s, a)]$$

The breakdown and details of each of the elements of the Q-learning equation is given in Figure 8.

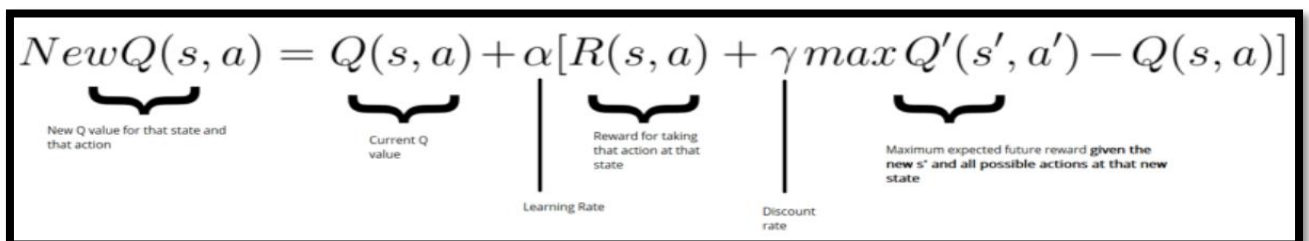


Figure 8: Q-learning equation



The Q-learning mechanism maintains a Q-table which contains the value of a state-action pairs. During the exploration of the state space, the Q-learning accumulates the value of particular state-action pairs in a table and these will further help find optimal path.

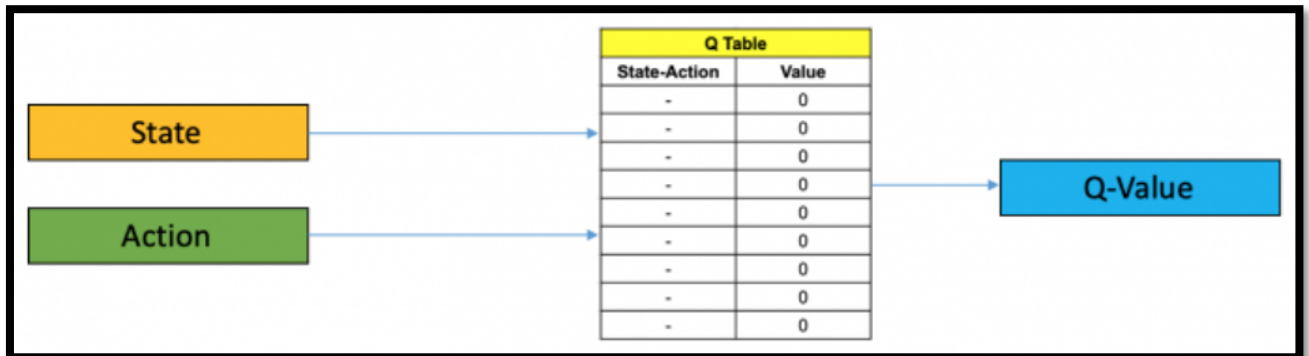


Figure 9: Q-learning table

The optimal path routing problem can be considered a minimization problem as the Q-learning algorithm must be modified to adhere to minimization requirements. Hence, the Q-learning equation mentioned earlier has been modified:

$$Q(s, a) < - Q(s, a) + \alpha[r + \gamma \min Q(s', a') - Q(s, a)]$$

Since the discount factor  $\gamma$  is also not a mandatory element and is needed on a case-by-case basis, the modified Q-learning algorithm can also be modified as:

$$Q(s, a) < - Q(s, a) + \alpha[r + \min Q(s', a') - Q(s, a)]$$

### D1.1.1 Modified Q-learning Algorithm

*“Initialize  $Q(s,a)$  arbitrarily”*

*“Repeat (for each episode):”*

*“Initialize  $s$ ”*

*“Repeat (for each step of episode):”*

*“Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)”*

*“Take action  $a$ , observe  $r, s'$ ”*

$$Q(s, a) < - Q(s, a) + \alpha[r + \gamma \min Q(s', a') - Q(s, a)]$$

*“ $s \leftarrow s'$ ”*

*“until  $s$  is terminal”*

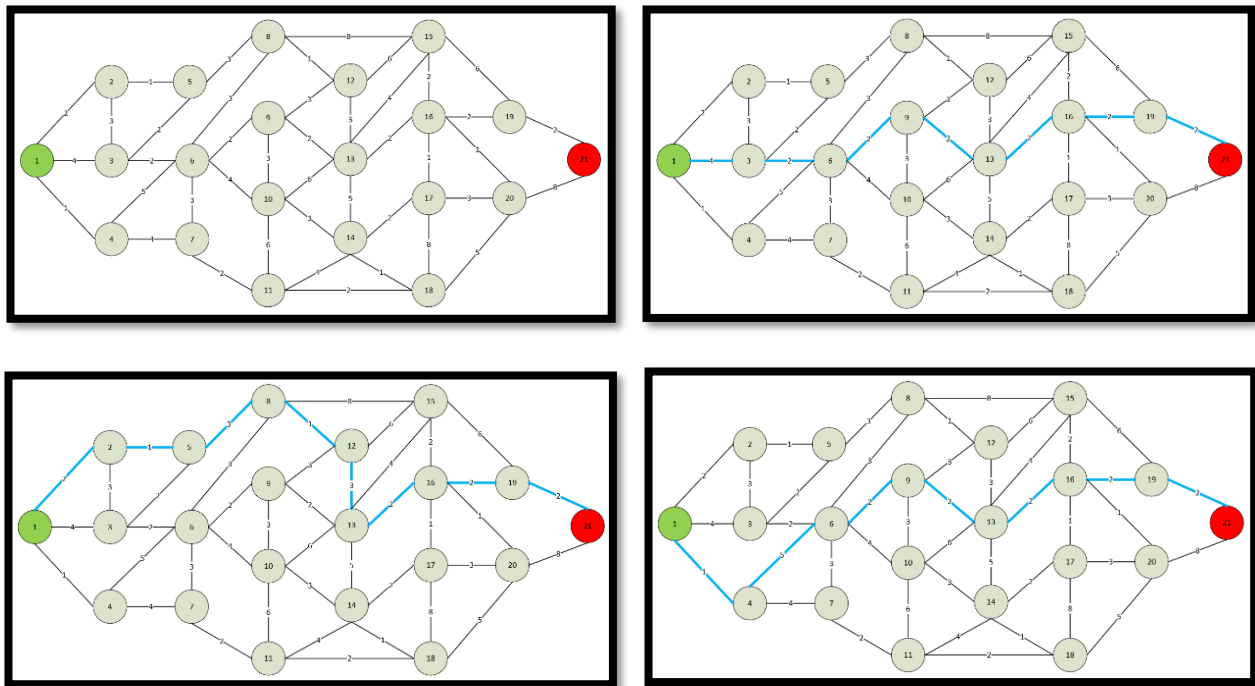
### D1.2 Implementation of optimal path routing

Implementation of optimal path routing using reinforcement learning is performed by developing the modified Q-learning algorithm as mentioned in section D1.1.

The code for this algorithm has been implemented in Python which has been shared partially in APPENDIX.

The code expects a network topology that is either unidirectional or bi-directional network links with

associated costs between the network nodes. These network topologies can be the topology of multi-node networks of an internet service provider, public cloud provider, public blockchain distributed network, etc.



**Figure 10: 21-node network (Top Left), 1<sup>st</sup> optimal path (Top Right), 2<sup>nd</sup> optimal path (Bottom Left) & 3<sup>rd</sup> optimal path (Bottom Right)**

The evaluation of this code is done over 3 different network topologies. The 1<sup>st</sup> network topology shown in Figure 10 is a bi-directional 21 node network where the 1<sup>st</sup> node is considered as source and the 21<sup>st</sup> node as destination.

**1<sup>st</sup> Network Output:** Time taken is 0.031 seconds with cost of 16. 3. Optimal paths to destination node 21 as shown in Figure 10.

**time is: 0.03121185302734375**

**[21]**

**{16: 3}**

**{21: 3}**

**{21: [[1, 3, 6, 9, 13, 16, 19, 21], [1, 4, 6, 9, 13, 16, 19, 21], [1, 2, 5, 8, 12, 13, 16, 19, 21]]} -> 3 optimal paths with cost of 16.**

The 2<sup>nd</sup> network topology shown in Figure 11 is a unidirectional 20 node network where the 1<sup>st</sup> node is considered as source and the 20<sup>th</sup> node as destination.

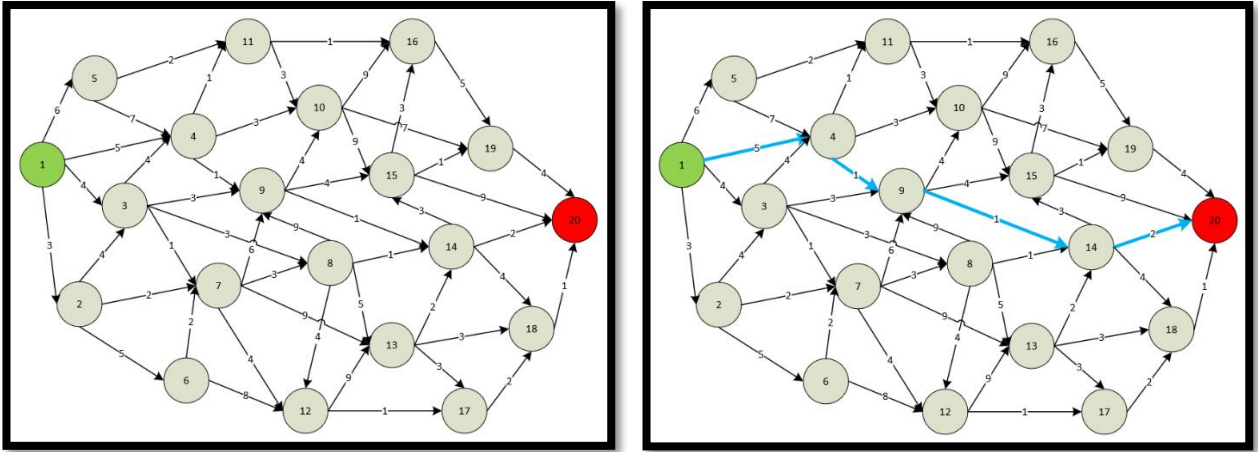


Figure 11: 20-node network (Left) & optimal path (Right)

**2<sup>nd</sup> Network Output:** Time taken is 0.015 seconds with cost of 9. 1. Optimal path to destination node 20 as shown in Figure 11.

**time is: 0.015632152557373047**

**[20]**

**{9: 1}**

**{20: 1}**

**{20: [[1, 4, 9, 14, 20]]} -> 1 optimal path with cost of 9.**

The 3<sup>rd</sup> network topology shown in Figure 12 is a unidirectional 10 node network where the 1<sup>st</sup> node is considered as source and the 10<sup>th</sup> node as destination.

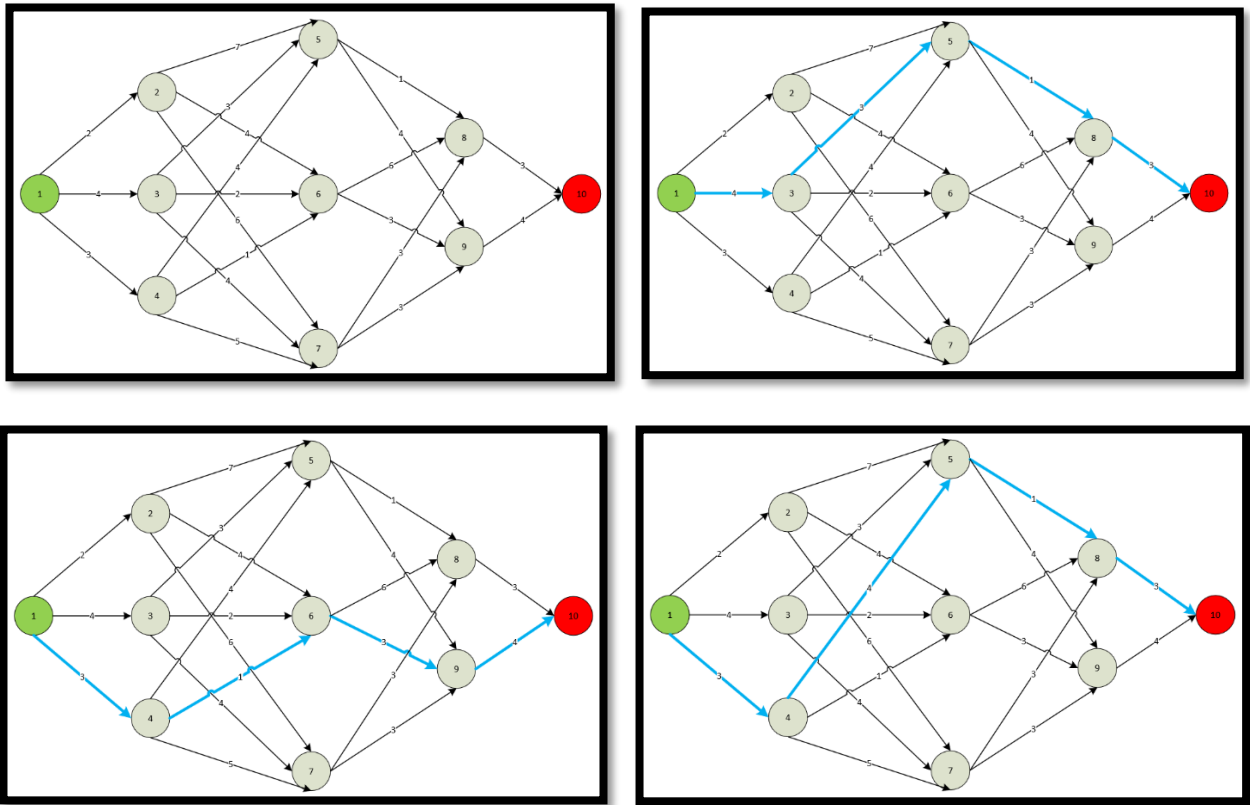


Figure 12: 10-node network (Top Left), 1<sup>st</sup> optimal path (Top Right), 2<sup>nd</sup> optimal path (Bottom Left) & 3<sup>rd</sup> optimal path (Bottom Right)

**3<sup>rd</sup> Network Output:** Time taken is 0.015 seconds with cost of 11. 3. Optimal paths to destination node 10 as shown in Figure 12.

**time is: 0.015653133392333984**

**[10]**

**{11: 3}**

**{10: 3}**

**{10: [[1, 3, 5, 8, 10], [1, 4, 5, 8, 10], [1, 4, 6, 9, 10]]} -> 3 Optimal paths with same cost 11.**

### **D1.3 Conclusion**

The implemented algorithm which is a modified Q-learning from reinforcement learning adaptively finds the optimal path for a given network topology. This algorithm not only finds a single optimal path but would find all the optimal paths that are possible for a given network topology.

One of the best parts of using the reinforcement learning approach for optimal path routing rather than using traditional shortest path algorithms is that this approach is model free and can adaptively find optimal paths in a dynamic and larger network.

This algorithm can be easily extended to any network topology regardless of cyclic or acyclic networks. The applications to this algorithm will be far reaching to distributed computing, public blockchain, cloud service providers, and more. This work can be further extended to very large state space when integrated using deep learning networks.

## References

- [1] "Chettibi S., Chikhi S. (2011) A Survey of Reinforcement Learning Based Routing Protocols for Mobile Ad-Hoc Networks. In: Özcan A., Zizka J., Nagamalai D. (eds) Recent Trends in Wireless and Mobile Networks. CoNeCo 2011, WiMo 2011. Communications in Computer and Information Science, vol 162. Springer, Berlin, Heidelberg"
- [2] "Li, Xue Yan & Li, Xuemei & Yang, Lingrun & Li, Jing. (2018). Dynamic route and departure time choice model based on self-adaptive reference point and reinforcement learning. Physica A: Statistical Mechanics and its Applications. 502. 10.1016/j.physa.2018.02.104."
- [3] "Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA."
- [4] "Russell, Stuart J, and Peter Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, N.J: Prentice Hall, 1995. Print."
- [5] "Udemy. (2019). Online Courses - Available at: <https://www.udemy.com/artificial-intelligence-reinforcement-learning-in-python/learn/v4/overview> [Accessed 7 Feb. 2019]."
- [6] "Watkins, C.J .C.H, & Dayan, P. (1992). Q-learning. Machine Learning, 8, 279-292."
- [7] "Yogendra Gupta and Lava Bhargava. (2016). Reinforcement Learning based Routing for Cognitive Network on Chip. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (ICTCS '16). Association for Computing Machinery, New York, NY, USA, Article 35, 1–3."
- [8] "You, Xinyu & Li, Xuanjie & Xu, Yuedong & Feng, Hui & Zhao, Jin. (2019). Toward Packet Routing with Fully-distributed Multi-agent Deep Reinforcement Learning."
- [9] "Z. Mammeri. (2019). Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches, in *IEEE Access*, vol. 7, pp. 55916-55950."

## APPENDIX

### APPENDIX – Python code for Q-learning – optimal path routing

---

```
def Q_routing(T,Q,alp,eps,epi_n,start,end):
    num_nodes = [0,0]
    for e in range(epi_n):
        if e in range(0, epi_n,1000):
            print("loop:",e)
        state_cur = start
        #route_cur = [start]
        goal = False
        while not goal:
            mov_valid = list(Q[state_cur].keys())

            if len(mov_valid) <= 1:
                state_nxt = mov_valid [0]
            else:
                act_best = random.choice(get_key_of_min_value(Q[state_cur]))
                if random.random() < eps:
                    mov_valid.pop(mov_valid.index(act_best))
                    state_nxt = random.choice(mov_valid)

                else:
                    state_nxt = act_best

            Q = Q_update(T,Q,state_cur, state_nxt, alp)

        if state_nxt in end:
            goal = True
            state_cur = state_nxt

    if e in range(0,1000,200):
        for i in Q.keys():
            for j in Q[i].keys():
```

```

        Q[i][j] = round(Q[i][j],6)
    nodes = get_bestnodes(Q,start,end)
    num_nodes.append(len(nodes))
    print("nodes:", num_nodes)
    if len(set(num_nodes[-3:])) == 1:
        break
return Q

```

```

def Q_update(T,Q,state_cur, state_nxt, alp):
    t_cur = T[state_cur][state_nxt]
    q_cur = Q[state_cur][state_nxt]
    q_new = q_cur + alp * (t_cur + min(Q[state_nxt].values()) - q_cur)
        #q_new = q_cur + alp * (t_cur + gamma * min(Q[state_nxt].values()) )
        #q_cur[action] = reward + self._gamma * np.amax(q_s_a_d[i])
    Q[state_cur][state_nxt] = q_new
return Q

```



Dell Technologies believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." DELL TECHNOLOGIES MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying and distribution of any Dell Technologies software described in this publication requires an applicable software license.

Copyright © 2020 Dell Inc. or its subsidiaries. All Rights Reserved. Dell Technologies, Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.