A Floyd-Warshall-based Reoptimization of Q Matrix on the Single DVRPPD with On-demand Cancellations

Jasper Kyle Catapang[1], Geoffrey A. Solano[2]
[1]Department of English Language and Linguistics, University of Birmingham, United Kingdom
[2]Department of Physical Sciences and Mathematics, College of Arts and Sciences
University of the Philippines Manila - Manila City, Philippines
Email: [1]jxc1354@student.bham.ac.uk, [2]gasolano@upm.edu.ph

***Abstract*—Delivery services have reached an all-time high in terms of demands due to the COVID-19 pandemic. An optimal routing plan for these different courier services is a must. The vehicle routing problem (VRP) is an NP-hard problem in logistics research and it can be solved by exact algorithms, heuristics, and machine learning through reinforcement learning. This study introduces a reoptimization alternative as opposed to trivial Q-learning retraining on a newly introduced subproblem of dynamic VRP and VRPPD, the dynamic vehicle routing problem with pickup, delivery, and cancellation (DVRPPDC). The reoptimization technique is called Floyd-Warshall LookUp Reoptimization of Rewards Yearned (FLURRY). Combined with a one-time Q-learning computation beforehand, the Q matrix produced is updated at the cell level by a lookup table containing all the shortest paths among the pairs of parcel lockers. The lookup table is generated via Floyd-Warshall algorithm, a well-known shortest path algorithm. Upon testing on a new dataset for DVRPPDC, one-time Q-learning combined with FLURRY is 6.1x to 10.6x faster to compute than Q-learning retraining. Furthermore, an additional study done after the main series of experiments reveal that the methodology does not necessitate any Q-learning training at the first time step at all. FLURRY can be done on a Q matrix of zeroes and achieve the same path output and traversal time as Q-learning with FLURRY. The standalone FLURRY algorithm further speeds up the computation, compared to the naive Q-learning approach, from 6.1x - 10.6x to 26.5x - 117.5x.**

***Index Terms*—q-learning, reinforcement learning, vehicle routing problem, VRPPD, Floyd-Warshall algorithm**

I. Introduction

The Vehicle Routing Problem (VRP) is a popular NP-hard combinatorial problem in the field of logistics research, that has been introduced in 1959 by Dantzig and Ramser [1]. In the vanilla VRP, depots and different pickup or drop-off locations are considered as vertices V of a graph G and the distances between them are treated as edges E [2]. The common objective of all variants of the vehicle routing problem is to minimize total cost, as represented by Eq. (1), where c_{ij} is the traveling cost from i to j, and x_{ij} is 1 if the vehicle chose to take the path and 0 if the vehicle did not.

$$minimize \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

VRP has multiple variants. Some of these subproblems are the capacitated VRP (CVRP), VRP with time windows (VRPTW), open VRP (OVRP), VRP with pickup and delivery (VRPPD), and dynamic VRP (DVRP). CVRP is a variation of VRP wherein every vehicle has a single capacity (like parcel weight or parcel count) it can carry for every vehicle [2]. CVRP's objectives are to deploy the least amount of vehicles and to have the smallest sum of travel time, while keeping in mind that each vehicle cannot exceed its capacity [2]. VRP with time windows (VRPTW) is straightforward. It is a subproblem of VRP that incorporates time windows for when demands are requested [3]. Open VRP is a type of VRP where the vehicles deployed are not required to return to the depot [4]. In VRP with pickup and delivery or VRPPD, vehicles deployed are to deliver and pickup ordered parcels while applying the same constraints as the rest of the VRPs [5]. Another variant of VRPPD utilizes parcel lockers instead of houses/establishments. Lastly, dynamic VRP or DVRP, is a VRP subproblem that exploits technological advances such as instant messaging and communication. DVRP enables vehicles to receive new orders on the way, in a series of time steps [6].

Different classes of solutions exist to solve VRP. Several of these solutions are exact algorithms, heuristics, and machine learning algorithms. Branch and bound [7], branch and cut [8], and branch and cut and price [9], are a few of the exact algorithms used in solving different subclasses of the vehicle routing problem. Due to VRP's NP-hard nature, researchers have started to use heuristics instead of exact algorithms. Some of these heuristics are Tabu search, simulated annealing, genetic algorithms, ant colony optimization (ACO), and adaptive large neighborhood search (ALNS) [10]. Since the objective of solving VRP is to minimize Eq. (1), machine learning algorithms have been leveraged to learn this equation and its other variations. Particularly, researchers have employed reinforcement learning (RL) to solve VRPs. According to [2], the majority of RL solutions are computationally expensive but they learn the most efficient solution the best due to their concept of rewards, goals, and penalties. Researchers [11] have used

RL on a variant of DVRP with congestion. They utilize Q-routing, a method intended for packet routing. Another article proposes the use of Q-learning for CVRP and claims that their methodology achieves state-of-the-art compared to other machine learning approaches [12]. Nazari et al. [13] propose an RL solution that uses recurrent neural networks (RNN) and attention. Due to the recent novel approaches brought about by reinforcement learning, the authors of this article choose to utilize reinforcement learning even though it is computationally expensive. The rationale is discussed further in the succeeding subsections.

With the brief background on VRP—its subproblems, and its solutions—already discussed, the authors propose a variation of VRP by combining features of VRPPD and DVRP, with some adjustments. Dynamic vehicle routing problem with pickup and delivery or DVRPPD, follows the same concept as VRPPD: pickup and drop-off locations. This variant is dynamic since some of the pickup locations and drop-off locations are unknown to the rider of the vehicle upon the start of travel. These orders are received and processed along the way. For this paper, the authors are considering a single-vehicle version of DVRPPD. The constraints and features that make this subproblem a different version from the existing DVRPPD are the following:

1) A single vehicle is used to solve the DVRPPD.
2) The vehicle must start from and return to the depot.
3) The rider of the vehicle will await for new orders as time goes by.
4) The DVRPPD variant introduces the concept of midpoints in between parcel lockers to symbolize points in the middle of transit.
5) The rider of the vehicle must accept pickup and drop-off orders at the designated parcel lockers only. Midpoints cannot be set as pickup and drop-off locations.
6) The rider of the vehicle must also accept pickup and drop-off cancellations along the way.
7) The rider of the vehicle must consider the distances and average speeds across different parcel lockers and their midpoints, in order to minimize the travel time.

The average speed between two vertices (V_i, V_j) is made possible by the current advances in GPS technology, like Google Maps or Waze. As such, the inputs of the solution that the authors propose are the travel distance and the average travel speed.

Since the implementation of Q-learning—according to the results of [12]—achieved state-of-the-art, the authors employ Q-learning as the base algorithm for the DVRPPD with cancellations. Although Q-learning has achieved state-of-the-art results, the problems of DVRPPD's dynamic properties with respect to time and Q-learning's computational cost still stand. The simple way to solve DVRPPD's evolution through time is by implementing Q-learning on every update. That is, when the queue of current orders is updated, Q-learning is performed again so that the goal and paths learned by Q-learning would be calibrated to the latest state of the order queue. This is the technique that the authors propose as the naive Q-learning approach for DVRPPD with cancellations. This technique is trivial and thus invites the introduction of a much better approach.

The naive Q-learning approach might have solved DVRPPD's time problem but Q-learning is still computationally expensive to run on every update of the order queue. The authors propose the concept of reoptimization on the DVRPPD with cancellations. Instead of recalculating the new reward and Q matrices for Q-learning for every update, Q-learning will only be used on the first time step once. The reoptimization the authors propose make use of a lookup dictionary provided by the calculation of the shortest paths of all the pairs of parcel lockers via Floyd-Warshall algorithm [14]. The lookup dictionary updates a part of the Q matrix from the first time step, every time there is an update on the order queue. This reoptimization technique is called the Floyd-Warshall LookUp Reoptimization of Reward Yearned or FLURRY.

The following are the researchers' contributions:

1) The introduction of the dynamic vehicle routing problem with pickup, delivery, and cancellation on parcel lockers given the lockers' respective distances and average traversal speed among each of them.
2) The introduction of the Floyd-Warshall LookUp Reoptimization of Reward Yearned (FLURRY) as a reoptimization step on the Q matrix output of the Q-learning in the first time step of the DVRPPD with cancellations.
3) The proposal of FLURRY as a standalone solution to DVRPPD with cancellations, without any Q-learning training.

This paper is divided into six main sections, with the introduction being the first section. The second section is the theoretical framework. The authors discuss Q-learning, the Floyd-Warshall algorithm, and FLURRY in Section 2. Next, the experimental setup. This section includes the data the paper uses and the experimental settings of the Q-learning algorithm with FLURRY for reoptimization. Section 3 also includes an additional study: component contribution analysis on Q-learning with FLURRY. The fourth section is the results and discussion. This is the section that shows the comparison between the performance of naive Q-learning and Q-learning with FLURRY. The fifth section summarizes the contributions and significant findings of the paper. Finally, the sixth section gives the possible direction for future iterations of this study.

II. Theoretical Framework

A. *Q-learning*

Q learning, introduced by Watkins [15] in 1989 is a simple technique to enable agents to act on a policy based on the rewards, goals, and penalties that the agents have encountered in Markovian decision processes. It is a type of RL algorithm that is model-free. It can be considered as a dynamic programming approach that is asynchronous. The agents in Q-learning learn through the multiple experiences they have with every training epochs in a simulation. Furthermore, Q-learning is considered to be primitive according to Watkins [15] but it can still be employed on advanced machines. Q-learning is heavily anchored on a Bellman equation as a value iteration update.

$$\underbrace{Q(s,a)_{new}}_{\substack{\text{New}\\ \text{Q-Value}}} = \underbrace{Q(s,a)}_{\substack{\text{Current}\\ \text{Q-Value}}} + \alpha\left[\underbrace{R(s,a)}_{\text{Reward}} + \gamma \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given}\\ \text{new state and all possible actions}}} - Q(s,a)\right] \quad (2)$$

Eq (2) shows the core of the Q-learning algorithm, where $R(s,a)$ is the reward calculated when taking the action from s to s', and α is the learning rate. The learning rate must be between 0 and 1, including 1.

Furthermore, in Q-learning, for every n^{th} episode, an agent does the following:

- observe its current state s
- select and take action a
- observe next state s'
- receive reward $R(s,a)$

B. Floyd-Warshall algorithm

The Floyd-Warshall algorithm is used to calculate the shortest paths of all pairs in a given graph [14]. The Floyd-Warshall lookup dictionary is computed before the one-time Q-learning is performed. The Floyd-Warshall algorithm uses a dynamic programming technique according to [14]. The dynamic programming solution is performed by combining the solutions of the subproblems formed. Additionally, the same algorithm can be applied to graphs that have negative edge weights [14]. The algorithm can be found in Algorithm 1.

Algorithm 1: Floyd-Warshall algorithm

```
Data: Distance matrix $D$
Result: Shortest path matrix $D$
for $k$ in $[n]$ do
    for $i$ in $[n]$ do
        for $j$ in $[n]$ do
            $D_{ij} = \min(D_{ij}, D_{ik} + D_{kj})$
        end
    end
end
return D
```

C. FLURRY

The Floyd-Warshall LookUp Reoptimization of Reward Yearned or FLURRY is a reoptimization of the Q matrix learned by the one-time Q-learning in the first step of the DVRPPD with cancellations. The reoptimization technique uses the calculated Floyd-Warshall costs for every pair of vertices V in the graph G.

Algorithm 2: FLURRY reoptimization algorithm

```
Data: Q matrix $Q$, updated order queue $O$,
      Floyd-Warshall dictionary $F$, current state $s$,
      Q-value $q$
Result: Updated Q matrix $Q$
$C = array()$ // array to store path candidates
for $index, o$ in $O$ do
    $path, length = F[s][o]$
    $C \leftarrow path, length$
end
$min \leftarrow$ Get minimum $length$ in $C$
$path \leftarrow C[min]$
for $index, v$ in $path$ do
    $Q[v, path[index + 1]] = q \times (index + 1)$
end
return Q
```

The FLURRY reoptimization algorithm, shown in Algorithm 2, takes the one-step Q matrix Q from the first time step, the updated order queue O, the Floyd-Warshall dictionary F, the current state s when the update to the order queue took place, and the Q-value q, as the inputs. The Q-value q must ideally be large, so that the Q matrix being reoptimized by FLURRY focuses on the vertices from the optimal path provided by the Floyd-Warshall dictionary F. The authors recommend setting q to a value bigger than the current maximum of the one-step Q matrix Q.

III. Experimental Setup

The authors use Python 3.8 on a local machine with Intel i9, RTX 3070, and 32 GB of RAM. The Q-learning is coded with NumPy and the graphs are represented by NetworkX using Kamada-Kawai and spring layouts.

A. Data

Since the DVRPPD with cancellations has a different set of constraints and objectives than that of the previous VRP variants, the authors make use of their own dataset instead of the standard Solomon benchmark [16].

The data used for this paper is a set of graphs G that contain information about the vertices V and edges formed in between the vertices E. The data also generates at random an order queue represented by a time step sequence. The data is controlled and generated by setting a random seed in the NumPy library.

The data can be described by Table 1:

parcel lockers	total vertices	total edges
10	26	60
15	37	84
50	122	284

TABLE I: Graphs considered for testing Q-learning with FLURRY

In theory, the data is infinite since it makes use of random seeds. The random seeds used on this data for this paper are 42, 0, 1, 2, and 3. An example of a directed weighted graph designed for this subproblem is provided in Figure 1. The light blue vertices are the parcel lockers; the orange vertices are the midpoints; and the green vertex is the depot.

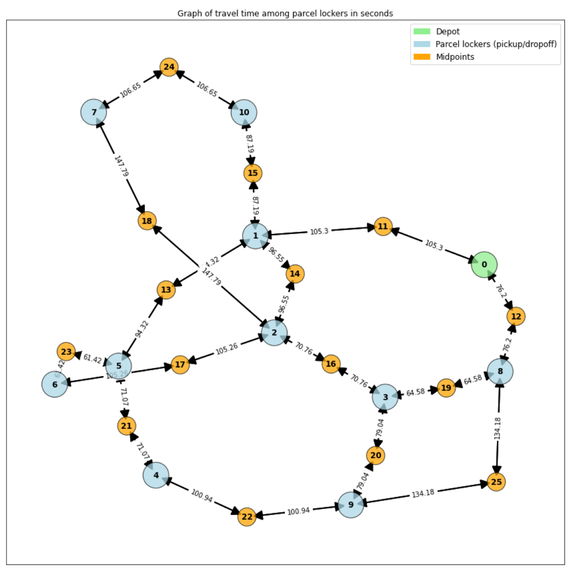

Fig. 1: DVRPPD with cancellations on 10 parcel lockers

B. *Q-learning and Q-learning with FLURRY*

The Floyd-Warshall dictionaries are filled up by the Floyd-Warshall algorithm by providing their respective graphs. The Floyd-Warshall algorithm calculates the shortest paths among the different pairs of vertices. Next, a simple version of Q-learning is done on the first time step. The hyperparameters used in the experiment are obtained from a series of sub-experiments. The optimal values are: α = 0.8, reward matrix R (described in Eq(3)), graph edge $E = \dfrac{d}{s \times 3.6}$, where d is distance in meters and s is speed in kilometers per hour. The epochs/episodes n used for Q-learning is calculated by $n = \dfrac{200V}{3}$. For the FLURRY reoptimization, the Q-value q is set to 100,000.

$$\text{for } r_{i,j} \text{ in R} = \begin{cases} \dfrac{5000 \times s}{d} & \text{if an edge and i or j is a goal} \\ \dfrac{500 \times s}{d} & \text{if an edge} \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

, where d and s are the same values from the previous paragraph.

For the naive Q-learning approach for the DVRPPD with cancellations, instead of using the FLURRY reoptimization for the next time steps when an update on the order queue takes place, the authors continue updating the Q matrix by retraining the Q-learning again with the same hyperparameters.

C. *Component contribution analysis*

Newell [17] describe a type of analysis wherein components of modular artificial intelligence systems are checked by assessing the individual contribution or performance of the components. The authors adopt the same principle and apply component contribution analysis by observing the effect of skipping the one-time Q-learning procedure. Instead of training a Q matrix input for FLURRY by performing Q-learning once, the authors use a zero matrix as input to FLURRY straight away.

D. *Evaluation Metrics*

The evaluation metrics for this experiment are simple. They are the travel time ratio (TTR) and the algorithm run time ratio (ARR). The TTR is calculated by the travel time of the Q-learning with FLURRY TT_{ours} and the travel time of the naive Q-learning approach TT_{NQL} via Eq(4). The ARR is calculated via Eq(5) wherein the same concept applies. Take note that a value of $TTR < 1$ indicates that Q-learning with FLURRY has a faster path calculated than the naive Q-learning. A value of TTR of exactly 1 means that the travel times of the paths taken of both algorithms are the same. A value of $TTR > 1$ means that the path calculated by the naive Q-learning approach is faster. As for the ARR in Eq(5), if the Q-learning with FLURRY reoptimization is better, the value should be more than 1.0.

$$TTR = \frac{TT_{ours}}{TT_{NQL}} \tag{4}$$

$$ARR = \frac{AR_{NQL}}{AR_{ours}} \tag{5}$$

IV. Results and Discussion

The results of the experiment, evaluated by TTR and ARR, are listed in Table 2 and Table 3.

Q-learning for both naive Q-learning and Q-learning with FLURRY make use of the following number of episodes: 10 parcel lockers $\rightarrow$ 1733 epochs, 15 parcel lockers $\rightarrow$ 2466 epochs, and 50 parcel lockers $\rightarrow$ 8133 epochs.

parcel lockers	seed	TT_{ours}	TT_{NQL}	TTR
10	42	2783.46	2783.46	1
10	0	2207.09	2207.09	1
10	1	2072.04	2072.04	1
10	2	1776.47	1776.47	1
10	3	**2721.06**	2802.32	**0.97**
15	42	3041.42	3041.42	1
15	0	**2602.51**	2624.67	**0.99**
15	1	2887.87	**2866.86**	**1.01**
15	2	3043.09	**2717.52**	**1.11**
15	3	2121.5	2121.5	1
50	42	**6216.5**	6376.02	**0.97**
50	0	5359.28	5359.28	1
50	1	**5170.68**	5627.5	**0.92**
50	2	5784.89	**5212.61**	**1.1**
50	3	**4919.43**	5078.95	**0.97**

TABLE II: Travel time ratio on training data

parcel lockers	seed	AR_{NQL}	AR_{ours}	ARR
10	42	1.14	**0.17**	**6.71**
10	0	1.01	**0.17**	**5.94**
10	1	1.13	**0.17**	**6.65**
10	2	1.02	**0.18**	**5.67**
10	3	1	**0.18**	**5.56**
15	42	2.42	**0.23**	**10.52**
15	0	2.2	**0.22**	**10**
15	1	2.64	**0.22**	**12**
15	2	2.26	**0.22**	**10.27**
15	3	2.23	**0.22**	**10.14**
50	42	11.8	**1.13**	**10.44**
50	0	11.78	**1.13**	**10.42**
50	1	12.75	**1.13**	**11.28**
50	2	12.77	**1.13**	**11.3**
50	3	10.91	**1.13**	**9.65**

TABLE III: Algorithm run time ratio on training data

Table 2 shows the performance of the naive Q-learning (NQL) approach vs. Q-learning with FLURRY (QLF) reoptimization on the DVRPPD with cancellations data, in terms of path traversal time in seconds. There are five instances in the results where QLF reoptimization outperformed naive Q-learning, while there are three instances wherein NQL outperformed QLF reoptimization. The rest of the table shows that the path traversal times of the two algorithms are identical. The average TTR of Table 2 is 1.004.

Table 3, on the other hand, lists down the performance of NQL and QLF reoptimization, with respect to their algorithmic run time in seconds. For the 10 parcel lockers scenario, there is an average of 6.10 times speedup when using Q-learning with FLURRY reoptimization instead of utilizing the naive Q-learning. For the 15 parcel lockers and 50 parcel lockers scenarios, the average speedups are 10.6 times compared to the naive Q-learning.

seed	ARR_{main}	ARR_{CCA}
42	6.71	28.5
0	5.94	25.25
1	6.65	28.25
2	5.67	25.5
3	5.56	25

TABLE IV: ARR of NQL vs Q-learning with FLURRY and ARR of NQL vs standalone FLURRY on the 10 parcel lockers test data

parcel lockers	average ARR_{main}	average ARR_{CCA}
10	6.10	26.5
15	10.59	117.5
50	10.62	54.55

TABLE V: Average ARR of NQL vs. Q-learning with FLURRY and average ARR of NQL vs. standalone FLURRY on different parcel lockers

The component contribution analysis shows that Q-learning training is not required to achieve the same path output as Q-learning with FLURRY. FLURRY on a zero Q matrix generate the same output as Q-learning with FLURRY but the algorithmic run time of standalone FLURRY is much faster. Table 4 shows the speedup provided by standalone FLURRY compared to naive Q-learning on the 10 parcel lockers test data. Table 5 compares the difference between the average algorithmic run time ratios among different parcel lockers. Standalone FLURRY further boosts run time speed from 6.10x to 26.5x, for 10 parcel lockers; from 10.59x to 117.5x, for 15 parcel lockers; and from 10.62x to 54.55x, for 50 parcel lockers. Performing standalone FLURRY instead of Q-learning with FLURRY maintains the quality of output while being cheaper and faster to compute. Without performing component contribution analysis, these performance boosts—in terms of run time—would not have surfaced.

V. Conclusion

VRP is an NP-hard problem in logistics research and it can be solved by exact algorithms, heuristics, and machine learning through reinforcement learning. This article proposes a reoptimization alternative to employing Q-learning for every update in the queue of parcel orders on the newly introduced subproblem of dynamic VRP and VRPPD, since retraining the Q matrix for every update is trivial and impractical since it is computationally expensive. The subproblem of dynamic VRP and VRPPD is called the dynamic vehicle routing problem with pickup, delivery, and cancellations (DVRPPDC). The reoptimization approach's objectives are to lessen the computational resources necessary to solve the DVRPPD with cancellations and to drastically reduce the solution's run time as opposed to the application of the slow naive Q-learning for every update in the order queue. The reoptimization method makes use of Floyd-Warshall algorithm to generate a lookup table for updating a portion of the Q matrix produced by running Q-learning once, at the beginning. This reoptimization is called Floyd-Warshall LookUp Reoptimization of Reward Yearned or FLURRY. FLURRY updates the specific cells in the Q matrix where the Floyd-Warshall algorithm has calculated the shortest path from the current vertex up to the nearest goal (order).

On a testbed of data containing 10, 15, and 50 parcel lockers, Q-learning with FLURRY matches, if not outperforms, naive Q-learning on DVRPPDC. Additionally, Q-learning with FLURRY is around 6.10 times faster to compute than naive Q-learning when considering the test data with 10 parcel lockers and around 10.6 times faster to execute than naive Q-learning for 15 parcel lockers and 50 parcel lockers. Through component contribution analysis, these speed boosts for run time further increase by eliminating Q-learning altogether. A standalone run of FLURRY on a zero Q matrix outputs the same results but at a much faster run time, ranging from 26.5 times to 117.5 times (when comparing naive Q-learning vs. standalone FLURRY).

VI. Future Work

For future iterations of FLURRY, the single DVRPPDC can be expanded to multiple vehicles and multiple depots, and can even incorporate capacity.

Acknowledgements

The authors would like to thank Linda Sarmiento for introducing reinforcement learning and Nathaniel Oco for his invaluable insights.

References

[1] B. Moghaddam, A. A. Fatahi, H. Kazemipour and B. Ashtiani, ”A mixed mathematical model of blending and routing problems,” Journal of Applied Science and Engineering Management, vol. 1, pp 46-51. 2013.

[2] A. K. Kalakanti, S. Verma, T. Paul and T. Yoshida, ”RL SolVeR Pro: Reinforcement Learning for Solving Vehicle Routing Problem,” 2019 1st International Conference on Artificial Intelligence and Data Sciences (AiDAS), pp. 94-99, doi: 10.1109/AiDAS47888.2019.8970890. 2019.

[3] L. M. Gambardella, E. Taillard, and G. Agazzi, ”Macs-vrptw: A multiple colony system for vehicle routing problems with time windows,” New ideas in optimization. 1999.

[4] Z. Ozyurt, D. Aksen, and N. Aras, ”Open Vehicle Routing Problem with Time Deadlines: Solution Methods and an Application,” Operations Research Proceedings 2005, 73–78. doi:10.1007/3-540-32539-5_12. 2006.

[5] G. Nagy and S. Salhi, ”Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries,” European Journal of Operational Research, vol.162, pp. 126–141. 2004.

[6] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, ”Ant colony system for a dynamic vehicle routing problem,” Journal of combinatorial optimization, 10(4), pp.327-343. 2005.

[7] G. Laporte and Y. Nobert, ”A branch and bound algorithm for the capacitated vehicle routing problem,” Operations Research Specktrum 5: 77-85. 1983.

[8] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, D. Naddef, and G. Rinaldi, ”Computational results with a branch and cut code for the capacitated vehicle routing problem,” Tech. Rep. 1 RR949-M, ARTEMIS-IMAG, Grenoble France. 1995.

[9] R. Fukasawa, J. Lysgaard, D. M. Poggi, M. Reis, E. Uchoa, and R. F. Werneck, ”Robust branch-and-cut-and-price for the capacitated vehicle routing problem,” Proceedings of the X IPCO, Lecture Notes in Computer Science 3064: 1–15. 2004.

[10] M. Zirour, ”Vehicle routing problem: models and solutions,” Journal of Quality Measurement and Analysis JQMA 4.1: 205-218. 2008.

[11] B. J. Cox and S. Krukowski, ”Reinforcement learning for adaptive routing of autonomous vehicles in congested network,” s. Stanford. edu. 2016.

[12] A. Bdeir, S. Boeder, T. Dernedde, K. Tkachuk, J. K. Falkner, and L. Schmidt-Thieme, ”RP-DQN: An application of Q-Learning to Vehicle Routing Problems,” arXiv preprint arXiv:2104.12226. 2021.

[13] M. T. Nazari, A. Oroojlooy, and L. V. Snyder, ”Reinforcement Learning for Solving the Vehicle Routing Problem,” NIPS. 2018.

[14] A. Pradhan and G. Mahinthakumar, ”Finding all-pairs shortest path for a large-scale transportation network using parallel Floyd-Warshall and parallel Dijkstra algorithms,” Journal of Computing in Civil Engineering, 27(3), pp.263-273, 2013.

[15] C. J. Watkins, ”Learning from delayed rewards,” PhD thesis, University of Cambridge, England. 1989.

[16] M. M. Solomon, ”Algorithms for the vehicle routing and scheduling problems with time window constraints,” Operations research, 35(2), 254-265. 1987.

[17] A. Newell. D. Raj Reddy (ed.). ”A Tutorial on Speech Understanding Systems,” In Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium. New York: Academic. 43. 1975.