Loop-breaking Approaches for Vehicle Route Planning with Multi-agent Q-routing

Jaap Meerhof
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
j.j.meerhof@student.utwente.nl

ABSTRACT

Computation of vehicle routes between locations in urban road networks is challenging due to the highly changing dynamics of vehicle traffic patterns. Reinforcement Learning (RL) is a powerful machine learning approach that can be exploited to develop autonomic control components in dynamic environments. Q-routing is an RL-based adaptive routing algorithm, originally proposed to improve packet routing in network communications. Q-routing can be exploited to self-adaptively compute vehicle routes in dynamic traffic scenarios. The Q-routing algorithm updates Q-tables to learn and predict travel times between road junctions. During the exploration stage, path tracking from the Q-tables may behave as a random walk from the source to the destination. However, if a loop is formed in multi-agent Q-routing, path tracking might loop forever until Q-table entries for the offending "loop nodes" are updated by another agent. Therefore, the formed loop must be broken somehow in order to find a simple (loopless) route path. In this research, four different approaches will be experimented with. This is done with the goal of breaking loops in Q-routing vehicle routing. The four different approaches are Loop-Erased Self-Avoiding Random, Negative Reward Function, Dual Reinforcement Learning and N-Learning. This is done with the goal of finding the most effective approach that leads to the shortest vehicle routes. In this paper the self defined N-learning algorithm combined with Dual Reinforcement Learning and the Negative Reward Function proved to be the most effective at lessening route loops and provided the vehicles with the shortest routes out of the different variations tested.

Keywords

Q-learning, Q-routing, Reinforcement Learning, Routing Loop Breaking, Vehicle Routing Planning

1. INTRODUCTION

Finding the shortest path in vehicular networks has been a topic of research for a long time, it continues to be an art that is short of research. Finding the optimal path of a vehicle in a network is called Vehicle Routing Planning (VeRP). An optimal path would be a path that allows a vehicle to reach its destination the fastest. VeRP is made more difficult as other vehicles cause congestion that make it so that the shortest path might not be the fastest path. There are algorithms that provide an answer to VeRP. An example of a popular shortest path algorithm is Dijkstra's algorithm, this is not an algorithm that would be used in realistic vehicular routing networks as road networks nowadays are getting too complex. Therefore new shortest path algorithms have been developed [1]. RL could be the next improvement when it comes to the development of routing algorithms. Boyan and Littman [6] used RL to route packets around in an adaptive (realistic) environment using Q-routing. Q-routing is based on Q-learning which is an algorithm proposed by Watkins and Dayan [15]. This application of Q-learning into the computational network setting can be taken further to tackle VeRP. Q-routing is normally used to route a packet through a cable to another router, this can be compared to sending a vehicle from a junction via a road to another junction. The original Q-routing algorithm has one Q-table that contains all the information gained in the network. Garcia-Robledo [8] proposed a multi-agent event-stream vehicle routing scheme, this means that the Q-routing algorithm was changed so that every junction is an agent with their own Q-table. For example, if a car arrives at a junction an event will be called that interacts with the Q-table. Having the system distributed means that the computations of the network are automatically distributed as well, this makes the system more scalable and faster in theory. The main problem is that current research on Q-routing focuses on the Vehicle Routing Problem (VRP). The VRP can be defined as the combinatorial optimization problem of finding a set of optimal paths for a group of vehicles in order to service numerous customers [8]. This is a well known problem in the fields of transportation [13] and computer science. However, this paper will focus on assisting a Q-routing-based approach to break loops and converge faster to shorter vehicle routes when solving VeRP. This paper proposes four different variations of the Q-routing algorithm that reduce the number of loops in routes produced by Q-routing during its exploration stage. This is a problem where vehicles get stuck in loops that cause congestion and long travel times. These loops are eventually broken as the Q-routing algorithm converges, they nonetheless need to be broken faster to assist Q-routing in converging faster to shorter vehicle routes. Variations of the Q-routing algorithm that should lessen the looping problem will be put forward. These variations will provide answers to the Research Questions (RQ) of this paper:

1. What are possible variations of the Q-routing algorithm that reduce the number of route loops during the Q-routing exploration stage?

2. Which Q-routing algorithm is most effective regarding VeRP?

35th Twente Student Conference on IT July 2nd, 2021, Enschede, The Netherlands.

(a) Which Q-routing variation reduces the number of vehicle route loops the most?

(b) Which Q-routing variation produces the shortest vehicle route lengths?

The paper will first look into the available literature that is related to the Q-routing loop problem. Four different variations of the Q-routing algorithm will be explained and explored. These four algorithms are called Loop-Erased Self-Avoiding Random Walk, Dual Reinforcement Q-Routing, Negative Reward Function and N-Learning. A recommendation will be made as to what variation of the Q-routing algorithm is the one that is most effective regarding VeRP.

2. RELATED WORK

The Q-routing algorithm as first described by Boyan and Littman was initially made to improve packet routing. The variation of this algorithm that will be used in this research can be found in Equation 1. Q-routing uses a single Q-table represented as Q that holds all the Q-values. A Q-value represents the expected time a packet would be traveling if it took a certain decision. The network in the normal packet routing setting consists of multiple routers called nodes. These nodes can reach certain other nodes and transmit packets. In Equation 1 $Q_x(d, y)$ is the Q-value of node x where a packet has destination d and wants to go to node y. The Q-table is initialized with small random values so the algorithm will randomly move in the beginning. When a packet is at a node it will look for the best Q-value for its destination and take this route until it reaches its destination. When a packet has moved from one router to another it will update the Q-table using Equation 1. This equation updates the Q-value with a new value that represents the expected time to move a packet. η is the "learning rate", this will influence how much a Q-value adapts to an update. s is the time it took to travel from node x to node y (including queue time). In normal packet routing $q + s$ is used where s is the travel time and q is the queue time. However in Garcia-Robledo's paper s is used to specify the time traveled on a road. Variable t can be retrieved via Equation 2. This equation retrieves the best (minimum) Q-value to reach d via y. This is done so that the old value $Q_x(d, y)$ is updated by adding the change after multiplying it by the "learning rate". It was pointed out by Boyan and Littman that the this algorithm could be seen as a variation of the Bellman-Ford shortest path algorithm that asynchronously performs its path relaxation step.

$$Q_x(d, y) = Q_x(d, y) + \eta(s + t - Q_x(d, y)) \tag{1}$$

$$t = \min_{z \in \text{neighbours of } y} Q_y(d, z) \tag{2}$$

There are plenty of working shortest paths algorithms for static network systems [1]. However, the algorithm proposed in this paper as well as in Garcia-Robledo can adapt to changes in the network.

The problem of packets traveling in loops is also acknowledged in Bitaillou et al. [2]. Bitaillou et al. gives the following thoughts on how the loop problem could be fixed in Q-routing: "Another possibility is to change the reward in the Q-function. Indeed, the distance (in hop count) could be considered in order to help the choice of the best route. The reward can also take account of the number of dropped packets." The problem is also acknowledged in Kiadi et al. [11] as they conclude that the looping problem "can be fixed by adding more logic to the code to identify the loops (that is more expensive to do) or to change the stop criteria".

To the best of the knowledge of this author, this paper is the first attempt to address the Q-routing route loop problem when solving VeRP.

3. MULTI-AGENT Q-ROUTING

The Q-routing algorithm for solving VeRP used in this paper is a simplification of Garcia-Robledo's Multi-Agent Event-Stream Vehicle Routing Scheme. This is an adaption of the Q-routing algorithm proposed by Boyan and Littmann to work in road networks. One change is that Garcia-Robledo's algorithm has multiple agents. This means that there are multiple Q-tables instead of one. These Q-tables are stored inside of the different junctions/nodes. In this road network, routers now are junctions and network links now are roads. The Event-Stream part of the algorithm symbolises the fact that these agents will have to stream events to each other to update the Q-tables. In this paper it is assumed that event-streaming does not affect the amount of loops made and will therefore not be simulated. Both Garcia-Robledo's, as the system used in this paper, uses Python, TraCI, and SUMO. SUMO provides a traffic simulation package and Python can interface with SUMO via TraCI to update vehicle's routes and get information on the road network. A SUMO network has junctions and roads which TraCI calls nodes and edges respectively, these terms will be used interchangeably.

The algorithm has a simulator that creates a junction object for every junction in the SUMO road network. Every junction x in the simulator is initialized with Algorithm 1. This algorithm uses set N_x, this is a set that contains all neighbouring nodes of node x. The different sets and variables are explained in Table 1. Algorithm 1 creates a Q-table for every Junction, it does this by assigning a small random float for all available Q-values. The Q-table has multiple Q-values for every outgoing road, every outgoing road holds a Q-value for every possible destination. This way a vehicle can request a Q-value for every possible road at a junction with the random destination.

Algorithm 1: Init Q-Table

```
Input  : -
Output: sets a Junction's Q-table

for $x$ in J do
    for $outgoing\_road$ in $\mathrm{N}_x$ do
        for $destination$ in J do

            $\mathrm{J}_x$.Q[$outgoing\_road$.getID()][$destination$.getID()]
            ← small random float
        end for
    end for
end for
```

This work will use TraCI to retrieve all vehicles in the SUMO simulation. Every new vehicle in the simulation will be given a new route by the Simulator. The implementation creates this route by taking the best road that the different junctions advertise. The routing protocol can be found in Algorithm 2, in this algorithm the *best_edge* can be changed before adding it to route R. This is useful for the different variations that can change the routing algorithm using *best_edge*.

When a new vehicle first appears on a random road it will set its destination to be a random junction. If a vehicle is on a new road (or on the road it first appeared on) it will request a new route from the junction at the end of the road. The Q-table of the next junction will first be queried to find the *best_edge* to reach the vehicle's destination. The set B is used to hold the different outgoing edges by storing 2-tuples with the edge and Q-value sorted by Q-value. If there are multiple outgoing edges with the same Q-value than a random one will be chosen. The next junction at the end of the chosen road will then return its *best_edge*. This process will continue until the vehicle has a route that reaches the destination or when the road assembled is longer than 2∗|J|. The original Q-routing algorithm will not always be able to backtrack a route to the destination due to route loops, for this reason the extra condition 2∗|J| is added to break the route backtrack algorithm. After a vehicle has received a new route the Q-value of the last junction will be updated. This is done by using Equation 1 and 2. The time spent (s) on a road is retrieved via TraCI . The best estimated travel time from the next junction is retrieved via Equation 2.

```
Algorithm 2: Route
Input  : vehicleID, current_edge, destination_id
Output: route R for a vehicle with vehicleID

  best_edge = current_edge
  R = set(current_edge)
  next_node = current_edge.getToNode()
  while next_node.getID() is not destination_id and
  length of R < 2*|J| do
    B = set()
    for outgoing_edge in J_x.outgoing_edges do
      Q − value =
      J_next_node.Q[outgoing_edge.getID()][destination_id]
      B.append((outgoing_edge, Q − value))
    end for
    sort B in ascending order by Q-values
    // randomize if there are multiple choice's for
    best_edge
    A = set()
    best_cost = B[0][1]
    for tup in B do
      if tup[1] > best_cost then
        break
      end if
      A.append(tup[0])
    end for
    best_edge = Choose (e.g. random) from A
    best_edge = B[0][0]
    next_node = best_edge.getToNode()
    R.append(best_edge)
  end while
  return R
```

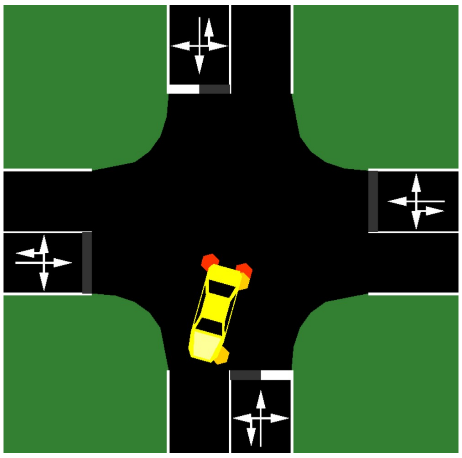
Figure 1. Vehicle requesting for a new route at next junction

Table 1. Variables

Symbol	Meaning
J	Set of all junctions
Q_x	Q table of junction x
P	Path so far
N_x	Outgoing Neighbouring roads from Junction x
R	Route taken by a vehicle so far
L	Loop-causing edges that will cause A random choice of edge the next time reached
B	Set with 2-tuple (edges, Q-value) with the best edges
α	Punish factor = 1.05
β	Punish severery factor = 0.05
η	"Learning rate" parameter of Q-function = 0.5

4. VARIATIONS

All variations of the Q-routing algorithm change the behaviour of the routing protocol in Algorithm 2. The variations can change the *best_edge* and edit the new expected route to change how the algorithm functions.

4.1 Loop-Erased Self-Avoiding Random Walks

In the beginning the vehicles will move around mostly randomly. When a new vehicle appears, a loop-less route can be made by following a Loop-Erased Random Walk (LERW). This is a mathematical term that is normally used in a mathematical lattice. It is created by doing a Simple Random Walk (SRW), an SRW can be compared to a vehicle taking random directions until it reaches its destination. A LERW is an SRW. However, after a loop is formed the nodes in the loop will be removed from the route. This will make a walk that reached its destination without intersecting itself [14]. This Loop-Erasing property is illustrated in Figure 2. Figure 2 explains how the LERW is used if Q-routing fails.

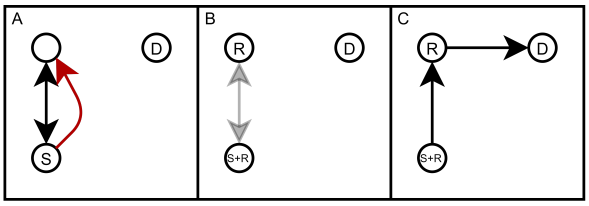

Figure 2. A: Vehicle is in a position where it would have to make a loop. B: The created loop is deleted and the nodes will now ignore Q-values. C: vehicle will be able to reach destination as loop causing nodes are random.

Then there are also Self-Avoiding Random Walks, this is an SRW that will avoid going to the same node twice by choosing different paths. If there is no path available to continue the walk the walk will terminate [9]. This Self-Avoiding property is illustrated in Figure 3. In this paper a variation of the Q-routing algorithm was created that will be called a Loop-Erased Self-Avoiding Random Walk (LESARW). This variation is not the same as the Loop-Erased Self-Avoiding Walk as defined in B. Duplantier [7]. This is because the variation is not only a random walk that removes loops. The implementation of the self defined LESARW variation of the Q-routing algorithm first checks if the *best_edge* given by the Q-routing algorithm will not create a loop. If this is not true then the algorithm will attempt to find a different edge that does not create a loop with the best Q-value. If this also fails the original *best_edge* will be used and the loop that will be created will be removed. All nodes that were in the loop will be added to the set L that holds all nodes that were in a loop. All edges that go to a node that has been in a loop will instead route the vehicle randomly. This brings back the randomness to the Q-routing algorithm to assure that the vehicle can reach the destination. This algorithm can be found in Algorithm 3, and is illustrated in Figure 2.

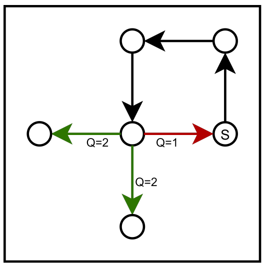

Figure 3. Vehicle will avoid the best edge with the lowest Q-value to avoid loop

4.2 Negative Reward Function

The loop problem might be lessened by changing the reward function of the Q-routing algorithm [2]. If an agent is stuck in a loop the Q-values of the edge that made the loop could be made higher so that this edge might not be taken in the future. This has been done with seemingly good results in the Deep Q-Routing algorithm proposed in Jalil et al. [10]. The algorithm used in the simulations can be found in Algorithm 5. This algorithm will call the punish algorithm (Algorithm 6) on all roads that were in the loop. The punish algorithm highers the Q-value of this road with the punish factor α. The roads that were in the loop are removed from the route so that the routing algorithm can continue with a loop-less route. The Q-values will not be punished to have a higher Q-value than the other edges. One node might end up with all outgoing edges to have the same maximum Q-value for a given destination. The algorithm will in that case pick a random edge with the best Q-value.

Algorithm 3: Loop-Erased Self-Avoiding Random Walk

```
Input  : best_edge from normal Q-routing
Output: new best_edge

if best_edge is in L then
    best_edge = Choose an element (e.g. random) from $N_x$
else if (best_edge.getToNode() is in R) or (best_edge is in P) then
    found_new_option = false
    for neighbour in $N_x$ do
        if neighbour is not in R or P then
            best_edge = neighbour
            found_new_option = true
            break;
        end if
    end for
    if not (found_new_option) and not (best_edge in P) then
        R, L = delete_until(R, best_edge, L)
    end if
end if
return best_edge
```

Algorithm 4: delete_until

```
Input  : Route R, best_edge and loop causing nodes L
Output: R without a loop up until best_edge and L

for $i$ in enumerate(reversed(R)) do
    route.pop($i$)
    L.append(R[$i$])
    if R[$i$] is best_edge.getID() then
        return R, L
    end if
end for
return R, L
```

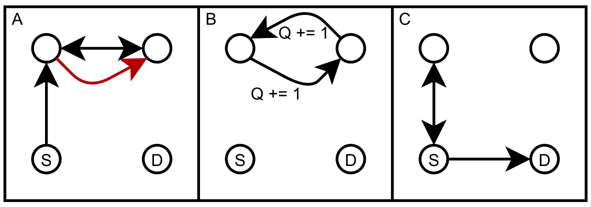

Figure 4. A: Vehicle is routed to a position where the best decision will make a loop. B: The two edges in the loop are punished. C: The vehicle can now make a route.

4.3 Dual Reinforcement Q-Routing

The original Q-routing algorithm uses forward exploration. In forward exploration the sending junction updates its Q-table when it sends out a vehicle. In S. Kumar's Dual Reinforcement Q-Routing (DRQ-Routing) [12] network nodes also use backwards exploration. With backwards exploration the receiving network node also updates its Q-table. Having two kinds of exploration doubles speed at which the system converges. This in term will cause the system to have fewer loops. One important change was made to the DRQ-Routing algorithm as described in S. Kumar's paper. For DRQ-Routing to work effectively in the network made the algorithm will use the same destination d for backwards and forwards exploration. In the origi-

Algorithm 5: Negative Reward

```
Input  : $best\_edge$ from q-routing and $destination$
Output: Route

  if $best\_edge$ is in R then
    for $edge$ in reversed(R) do
      R.pop($edge$)
      $from\_node$ = J.get($edge$.getFromNode())
      $from\_node$.punish($edge$, $destination$) // algorithm 6
      if $edge$ is $best\_edge$ then
        return R
      end if
    end for
    return R
  end if
```

Algorithm 6: Punish

```
Input  : $route\_edge\_id$, $destination\_node\_id$, $x$
         (current node), $s$ severity of punishment
Output: updates the Q-table by "punishing" node $x$

  $old$ = $J_x.Q[route\_edge\_id][destination\_node\_id]$
  $new$ = $old * (\alpha + (\beta * s))$
  $max$ = $J_x$.get_max_q_value($destination\_node\_id$)
  if $new > max$ then
    $new = max$
  end if
  $J_x.Q[route\_edge\_id][destination\_node\_id] = new$
```

nal DRQ-Routing the backwards exploration will use the starting junction s as a destination to update a Q-value. This makes it so that a single vehicle with destination d will learn its environment two times faster, if it were to update a Q-value with s as a destination this would not be the case. The only possible problem with DRQ-Routing is that DRQ-Routing is perfect for networks where the same travel time is experienced when sending a packet the other way around. This can be done in the original Q-routing setting as a wireless will have the similar speeds and reliabilities when sending the other way around. When a vehicle network is under a higher load a road might be congested on an outgoing road. However, it might be clear on the ingoing road.

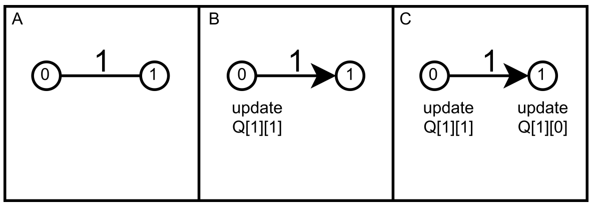

Figure 5. A: a road network with two nodes 1 and 2 connected via two way edge 1. B: normal forward exploration where node 0 updates its Q-table to node 1 via edge 1. C: forward and backwards exploration, node 1 now also updates its Q-table to node 0 over the outgoing edge 1.

4.4 N-Learning

After a vehicle has traveled over a road it can update the Q-value of the junction it came from with the parameter s from Equation 1. The original Q-routing will only retrieve one value t via Equation 2. However, the travel time s could be used for all minimum Q-values t of the next junction. This is done in the N-Learning variation, the junction at the end of the road on which a vehicle experienced travel time s will provide the junction at the beginning of the road with all N minimum Q-values for all destinations available. This way N Q-values can be updated instead of one. This will lessen the time for the system to converge N times faster. One problem with N-Learning is that packets with N Q-values will have to be sent each time an update needs to happen if a network has N junctions. Each junction in N-Learning would require more bandwidth the larger the network. Algorithm 7 shows the added for loop in the update function.

Algorithm 7: N Learning

```
Input  : $s$ (travel time), $next\_node$, $x$ (curent node),
         $edge\_id$
Output: Updates all Q-values over the taken road
         with id $edge\_id$

  for $node$ in J do
    $destination\_id$ = $node$.getID()
    $old = Q_x[edge\_id][destination\_id]$
    if $next\_node$.getID() == $destination\_id$ then
      $t = 0$
    else
      $t$ = $next\_node$.getmin($destination\_id$ // Equation 2
    end if
    $new = old +$
    $\eta$((s+t)−old)$Q_x[edge\_id][destination\_id] = new$
  end for
```

5. ASSESSMENT AND RESULTS

To answer research questions RQ1 and RQ2, simulations were done in an irregular grid road network first found in Boyan and Littman's paper on Q-routing [6]. In this network, a vehicle is able to move in all possible directions as illustrated in Figure 1. The types of roads most resemble collector roads as they have two lanes. The irregular 6 x 6 grid used can be seen in Figure 6. The junctions are not controlled by traffic lights. In this paper's simulations, vehicles appear every 15 full simulation steps resulting in 2000 vehicles going trough the network after 30000 steps. The simulations, however, checked on the vehicles 5 times per SUMO step. Resulting in 150000 simulation steps being done. The network was on a low load with these settings.

5.1 Loop Performance

Figure 7 reveals the amount of loops made over time for the different variations. The Original Q-routing algorithm performed the worst in terms of loops being made followed by DRQ-R (DRQ-Routing). The LESARW already more than halved the amount of loops. The LESARW with DRQ-Routing was quick to converge and avoid loops. The negative Reward Function with DRQ-Routing performed better than the LESARW with DRQ-routing. However, not by a significant margin. It can also be noted that the amount of new loops being made by LESARW with DRQ-Routing in the end was lower as the Negative Reward Function creates more loops in the later half of the simulation. The variation that revealed the most promising results was N-Learning. N-learning combined with DRQ-Routing and the Negative Reward Function produced only 24 loops. Combining the different variations produced the best results throughout the testing.

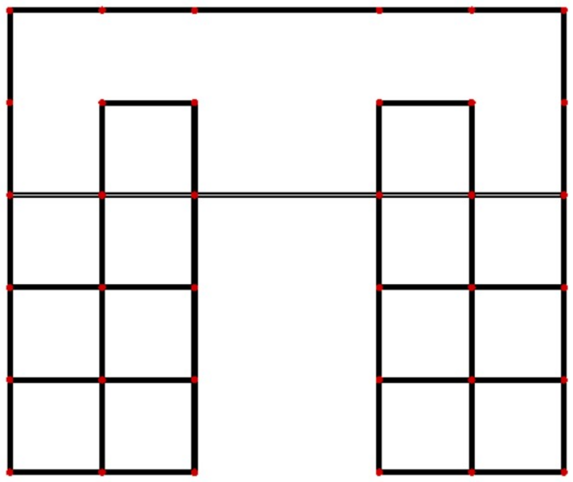

Figure 6. The Irregular 6 x 6 Grid Road Network used to carry out the simulations. First used by Boyan and Littman, and later used in most Q-routing papers.

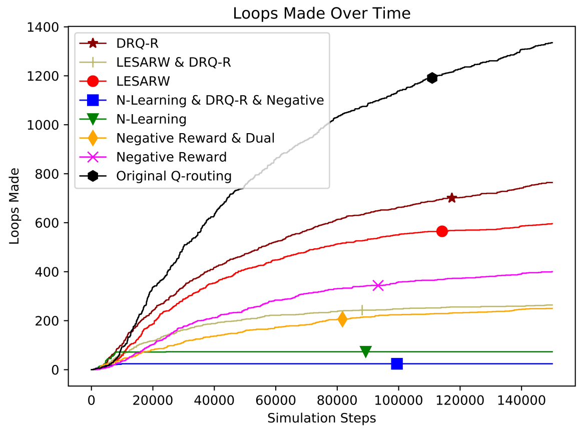

Figure 7. Loops made over simulation time by the different Q-routing variations

5.2 Routing Performance

Figure 8 and 9 reveal the different Q-routing variations and their performance. In Figure 8 the Y-axis contains the average vehicle route length in steps. If a vehicle had to travel 60 roads to get to their destination 60 will be added to the total and the average will be measured over time. This gives an indication as to how fast an variation guides the vehicle to its destination. The LESARW performed better than the standard Q-routing algorithm. However, not by a significant margin. DRQ-Routing learns about the network twice as fast and is therefore quick to minimise route lengths, however, loops are still being made resulting in extra routing time. The LESARW with DRQ-Routing demonstrated a small insignificant performance increase in respect to DRQ-Routing. The Negative Reward Function variations performed better than the LESARW variations. This can be explained by the fact that loop causing roads are punished until they are not viable anymore fast in the beginning of the algorithm. The N-learning variations performed the best as it is able to converge the fastest out of all variations. Figure 9 shows a simple moving average where the vertical axis holds a value that combines 10000 route lengths to the left and right of a point and averages these 20000 values, this averages out the graph and makes it easy to understand. It reveals that all algorithms end up with about the same routing performance, the most significant differences are in the early part of the exploration phase.

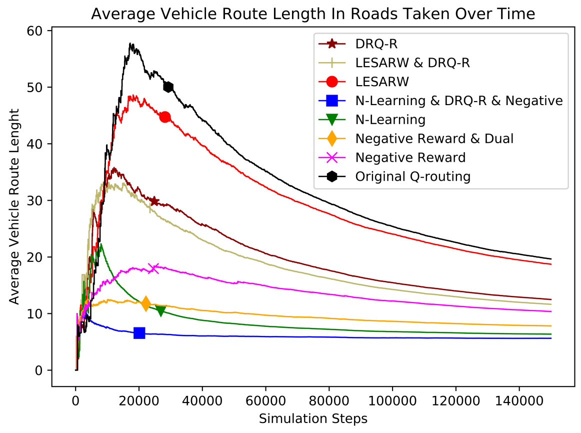

Figure 8. Average Route Lengths made by vehicles over simulation steps

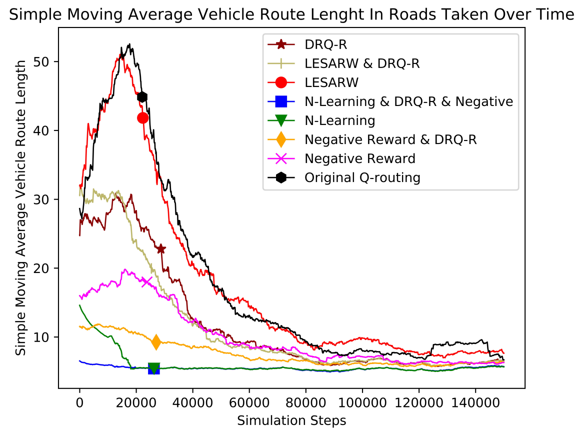

Figure 9. Simple Moving Average Vehicle Route Lengths that better displays the route lengths achieved by every variation over time.

6. CONCLUSION

N-Learning together with DRQ-Routing and the Negative Reward Function demonstrated to have the best performance in minimizing loops and routing lengths. This variation was therefore the best at VeRP. The importance of the rate of exploration with respect to the amount of loops being made is highlighted. Algorithms that are able to converge faster will experience significant decreases in the amount of loops being made. N-Learning allows the Q-tables to converge N times faster by updating N Q-values instead of 1. DRQ-Routing improved every variation by allowing algorithms to converge around twice as fast. The LESARW reduced loops significantly. However, it did not provide significant improvements in routing lengths as its Self-Avoiding property can lengthen routes significantly when it is in a position where it will eventually be forced to make a loop. The Negative Reward Function also proved to be a loop-breaking algorithm with comparable results to LESARW in terms of loops being made. However, the Negative Reward Function did perform superior in terms of the routing lengths when compared to the LESARW.

6.1 Answering Research Question 1

There are still many different unexplored variations of the Q-routing algorithm that can lessen the looping problem. However, the variations tested in this paper did all reduce the amount of loops. N-Learning provided the most significant improvement in breaking loops, combining N-Learning with DRQ-Routing and the Negative Reward Function was tested to be the best combination of variations. Small changed could possibly be made to the LESARW and Negative Reward Function variation to improve their loop-breaking properties.

6.2 Answering Research Question 2

N-Learning combined with DRQ-Routing and the Negative Reward Function was best at VeRP. This variation produced the least amount of loops and provided vehicles with the shortest routes out of the different variations.

6.2.1 Answering Research Question 2a

N-Learning combined with DRQ-Routing and the Negative Reward Function reduced the amount of vehicle route loops the most. N-Learning on its own provided the most significant decrease in loops.

6.2.2 Answering Research Question 2b

N-Learning combined with DRQ-Routing and the Negative Reward Function produced the best vehicle route loops. N-learning on its own provided the most significant decrease in vehicle route lengths. All variations tested reduced the route lengths in the exploration phase. DRQ-Routing can be combined with any variation to improve route lengths under a low network load. It was also gathered that the more a variation reduces loops the better the route lengths were.

6.3 Future Work

There are still many other variations to think of that could improve the loop avoiding properties found in this paper. The variations tested could also provide better results with small alterations. The LESARW could for example be tested without the Self-Avoiding property. Future work could look into algorithms that look forward enough to always create a loop-less route by assuring that the vehicle can reach the destination without making a loop with every possible move. Different punish factors could be experimented with in the Negative Reward Function. The algorithms were solely tested on a low road network load, performance can differ on high network loads. Especially with DRQ-Routing as the Q-tables might not be updated with correct Q-values when one road is more congested than its reverse. The network tested upon could also be altered, more sophisticated realistic networks could be retrieved from the OSMmx library. This library can be used to extract real road networks from OpenStreetMap [3, 4, 5]. However, these simulations do require a significant amount of computing power to simulate in SUMO. Another small change that should be tested is to first update the Q-values and then request a new route instead of doing it the other way around. This could have a big impact on DRQ-Routing and the Negative Reward Function. Vehicles will expectantly move less backwards the road behind a vehicle has also been updated via backwards exploration or punishments from the Negative Reward Function.

7. REFERENCES

[1] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, Cham, 2016.

[2] A. Bitaillou, B. Parrein, and G. Andrieux. Q-routing: From the algorithm to the routing protocol. In S. Boumerdassi, É. Renault, and P. Mühlethaler, editors, *Machine Learning for Networking*, pages 58–69, Cham, 2020. Springer International Publishing.

[3] G. Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[4] G. Boeing. The morphology and circuity of walkable and drivable street networks. In *The mathematics of urban morphology*, pages 271–287. Springer, 2019.

[5] G. Boeing. Planarity and street network representation in urban form analysis. *Environment and Planning B: Urban Analytics and City Science*, 47(5):855–869, 2020.

[6] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 671–678, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[7] B. Duplantier. Loop-erased self-avoiding walks in two dimensions: exact critical exponents and winding numbers. *Physica A: Statistical Mechanics and its Applications*, 191(1):516–522, 1992.

[8] A. Garcia-Robledo. Research progress report: An event-streaming platform for self-adaptive distributed vehicle route guidance in urban networks. 2019.

[9] O. B. Ido Tishby and E. Katzav. The distribution of path lengths of self avoiding walks on erdős–rényi networks. *Journal of Physics A: Mathematical and Theoretical*, 49, 2016.

[10] S. Q. Jalil, M. Husain Rehmani, and S. Chalup. Dqr: Deep q-routing in software defined networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[11] M. Kiadi, Q. Tan, and J. R. Villar. Optimized path planning in reinforcement learning by backtracking. 2019.

[12] S. Kumar and R. Miikkulainen. Dual reinforcement q-routing: An on-line adaptive routing algorithm. 1997.

[13] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[14] O. Schramm. Scaling limits of loop-erased random walks and uniform spanning trees. *Israel Journal of Mathematics*, 118(1):221–288, 2000.

[15] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.