Optimization of Message Ferrying UAV Paths using Monte Carlo Tree Search and Reinforcement Learning

Miguel Saraiva, Instituto Superior Técnico, Universidade de Lisboa

***Abstract*—Nowadays, communication is an indispensable element for the vast majority of human activities, such as economic, social, or other. Although there are several methods for this purpose, such as the Internet and mobile phones, there are situations where the connection may fail, as in natural catastrophes or in remote villages where there is no telecommunications infrastructure. In these cases, an alternative method of message transport is required, such as the use of Unmanned Aerial Vehicles (UAVs). We propose the adaptation of two groups of algorithms, one of planning (Monte Carlo Tree Search (MCTS)) and another of Machine Learning (Reinforcement Learning (RL)), for the search of efficient paths in Delay Tolerant Networks (DTNs). This communication was considered in two distinct network topologies: (1) from all nodes to a common node and (2) from all nodes to all nodes. The results show that both RL and MCTS methods present some improvements over the Travelling Salesman Problem (TSP), in smaller networks, for both topologies. RL methods proved to have more consistent results and require less memory when comparing to MCTS. For larger networks, some optimizations are still needed in order to obtain significant improvement.**

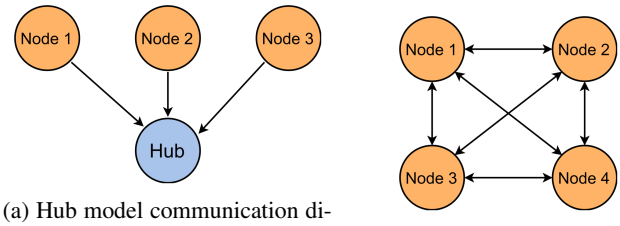
***Index Terms*—Delay Tolerant Network, Monte Carlo Tree Search, Reinforcement Learning, Unmanned Aerial Vehicle**

I. Introduction

A DTN is a network where nodes are not connected continuously but can be connected by employing a transporter that does not travel instantaneously, such as an Unmanned Aerial Vehicle (UAV). The focus of this report is in the use of UAVs as ferries, for the delivery of messages in static DTN environments. This means the UAV will directly retrieve the message from node A and deliver it to node B, where both A and B are static, and the UAV acts as a messenger. The use of drones in this context is quite suitable since they can use their maneuverability to access steep terrains such as mountainous terrain, tunnels, or even a congested city center, which might be hard to access in an emergency. Several algorithms have been developed with the objective of directing the drone in an efficient path.

For the remaining of this report, the following representation will be used to describe the problems we are trying to solve. Considering a DTN composed of a number N of static nodes, which can communicate via a mobile node (in this case, a single UAV), we will analyze the following scenarios (Figure 1):

- Scenario 1: Hub Node (Figure 1a)
  Apart from the N nodes of the network, there will be one new node that will be named as Hub Node, and which will only receive messages, whilst all the other N nodes will only send messages.
- Scenario 2: All to All (Figure 1b)
  This is the most general scenario, where all the N nodes that belong to the network will be able to both send and receive messages.



(a) Hub model communication directions between nodes

(b) All to All communication

Fig. 1: Communication modes between nodes. Arrows point from Sender to Receiver, in the case of bi-directional arrow both are senders and receivers.

In any of these two scenarios, the problem we propose to solve is the following: what is the best path a messenger can take, in order to deliver the largest amount of messages in a predefined interval of time? The only restriction for the message delivery is that all messages have a deadline before which they have to be received by the recipient, or else they expire.

The goal of our work is to create an algorithm that is able to solve this problem. Two approaches will be taken, one based on MCTS and another one based on RL. A simulator will be developed using Rust Lang in order to compare the results of these algorithms with each other and with previous research.

II. State of the Art

In this section we will analyze the operation of the algorithms and give a general idea of what was achieved in the past.

A. Optimizing UAV routes in DTNs

UAVs [1], also commonly known as drones, are aircraft that do not have a pilot on board. They can be controlled by a human from the ground, or there might be a preprogrammed algorithm that determines the path and actions it takes.

A DTN, as introduced before, consists in a communication environment where the nodes do not rely on having a continuously available bidirectional connection between each other. Apart from not being always connected, they might also have a considerable delay to communicate with one another. DTNs are divided into two types: DTNs with dedicated node-based routing protocols and without dedicated nodes. The first one

has a special kind of nodes that are moving and are the ones that transmit the messages between the other nodes. In the second case, this kind of nodes does not exist as every normal node can store, carry and forward information by itself [2]. Some research in this theme has been done, both with non-learning methods [3], [4], [5], [2] and learning methods [6], [7], [8]. This work will be focused on dedicated node-based routing protocols where the carrier is a UAV.

B. Monte Carlo Tree Search

MCTS [9], [10], [11] is an algorithm used in decision processes, most notably in 2-player games. The general idea revolves around creating an unbalanced tree by exploring nodes associated with actions that have a high probability of winning the game for the agent (from previous experience).

Each round consists of a certain number of iterations, and in each iteration, there are four key steps [12]. The descriptions provided below are relative to the standard MCTS algorithm (Figure 2):

1) Selection:
   Starting at the root node, and considering the already explored tree of the game, select successive nodes by employing the Tree Policy until a node that is not fully explored is reached;
2) Expansion:
   From the node selected previously, expand the tree by adding one child node corresponding to one of the unexplored actions, randomly chosen;
3) Simulation:
   Simulate a complete episode of the program from the newly created node on the previous step, by following the Default Policy, which usually consists of choosing actions at random. The termination of the simulation may happen due to reaching a terminal state or due to a limit depth imposed on the simulation, depending on the implementation. This is the step that resembles Monte Carlo Methods as the random generation of moves will give a probabilistic result of which action is best, after having completed a significant number of iterations;
4) Backpropagation:
   Use the return of the simulation to update the values in the nodes, from the node where the simulation begins, until the root of the tree. The nodes that are part of the simulation are forgotten (not added to the algorithm tree). They serve the sole purpose of reaching a value at the end of the simulation.

At the end of a round, the best action can be selected by choosing the child of the root node with the highest associated value. After that, another round can begin from the recently chosen child in order to select the next best action, and so on.

C. Reinforcement Learning

Reinforcement Learning (RL) is one of the three main topics of Machine Learning (ML), next to Supervised and Unsupervised Learning; ML by itself is a small part of Artificial Intelligence (AI). The basic idea behind RL is to create an agent that takes actions in an environment with the objective of maximizing an accumulated discounted reward. The actions taken by the agent are in the beginning randomly chosen, but as the agent starts learning, it starts being more confident that a certain action is better in a certain situation (state).

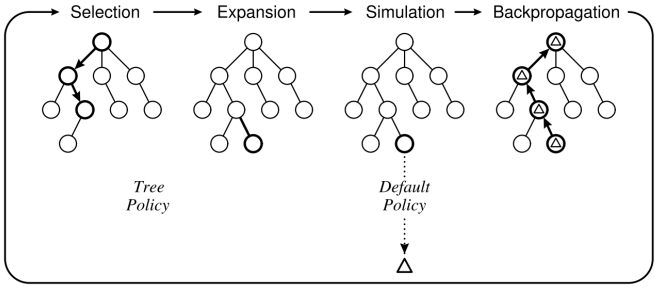


Fig. 2: The basic process of the MCTS algorithm. Image taken from [13].

An RL problem can often be described with the help of Markov Decision Processes (MDPs). An MDP is classically a formalization of sequential decision-making problems where actions at time t influence the reward not only at time t but also subsequent situations, through future rewards for time $t + n$ [12]. The objective is to estimate a value for each element of the function $Q_*(s, a)$ or for each element of $V_*(s)$. The former corresponds to the value for each state-action pair whilst the latter associates a value to each state, both given by following the optimal policy. If the policy followed is not optimal, but instead is some policy π, these functions are defined as $Q_\pi(s, a)$ and $V_\pi(s)$. The value associated with each state or state-action pair corresponds to the discounted sum of the rewards received in that state and the future ones. The further in the future the reward is given, the larger the discount is.

An MDP can be described as a tuple (S, A, T, R) where S represents the list of all the states, A represents the list of actions for all states, T is the Transition Function defined as $T : S$ x A x $S \rightarrow [0, 1]$ and R is the Reward Function defined as $R : S$ x A x $S \rightarrow R$ [14].

In Figure 3 it is possible to see the process of state transition in an MDP. The agent starts by performing an action A_t in the environment, resulting in a change from state S_t to state S_{t+1} with an associated reward R_{t+1}. This new state and reward are now considered by the agent, in order for it to be able to choose which action should be taken next.

To solve an MDP, a possible way is to use an algorithm called TD(0) or one-step TD, is a TD-Prediction algorithm, which is used in [6]. It is considered as model-based RL since it needs to be able to calculate the next state for every possible action in the current state. As any other RL prediction algorithm, its objective is to find the optimal value function v_* by following a known policy π. The value function $V(s)$ for TD(0) 1 is given by:

$$V(S_t) = V(S_t) + \alpha * [R_{t+1} + \gamma * V(S_{t+1}) - V(S_t)]. \quad (1)$$

This update is done after each step of an episode, or in other

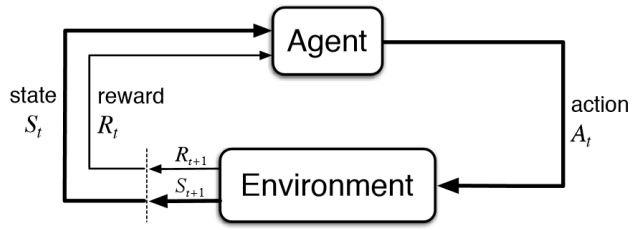


Fig. 3: Interaction between the agent and the environment in MDPs [12].

words, after an action is taken and a new state is reached. The value of the current state at time t is weighted with $1-\alpha$ of the previously stored value and α times the TD-Target. $\alpha \in [0, 1]$ is the learning rate and is used to state the importance of new information, comparing to what was already learned. The TD-Target, $R_{t+1} + \gamma V(S_{t+1})$, corresponds to the reward received upon transiting to state S_{t+1}, plus the value of the next state times a discount factor $\gamma \in [0, 1]$, which is used to control the importance given to future rewards.

Algorithm 1: Tabular TD(0) for the estimation of v_π [12]

```
Initialize Policy π to be evaluated;
Initialize Learning rate α ∈ [0, 1];
Initialize V(s) for all s ∈ S+, arbitrarily, with
  V(terminal) = 0;
for each episode do
    Initialize s;
    for each step of episode do
        a ← action given by π for s;
        Take action a, observe r, s';
        V(s) ← V(s) + α[r + γV(s') − V(s)];
        s ← s';
        until s is terminal;
    end
end
```

III. Proposed Algorithms

The RL and MCTS implementations created will be described in this section, followed by how they were changed in order to deal with deadlines.

A. RL Implementation

We have produced an implementation based on the work described on [6]. In some steps where there was not a clear procedure of the implementation made by the authors, several hypotheses were tested in order to find which one made the most sense or worked better.

This solution implements the TD(0) algorithm, which tries to evaluate each possible state for the drone to be at, by visiting them multiple times. The most critical part of these types of algorithms is the reward. In this case, the reward gives higher values for states which have a smaller number for the sum of the messages in all the nodes, as well as valuing smaller routes taken by the drone. In the next paragraphs, our implementation of their solution is going to be further explained.

To solve this problem with RL, we started with the reward from [6]:

$$r(s, a) = -\int_0^{t_a} (Ft + N_0)e^{-\beta t} dt, \tag{2}$$

where t_a represents the time that the action a takes to be completed and N_0 represents the total number of messages at the beginning of the action being evaluated. The negative integral will give better state-values to states which have a lower amount of accumulated messages.

The policy is given by Equation 3. It represents the way that actions are selected when presented with a certain state. The value function (Equation 4) represents how the value of each state is updated after each time it is visited. $\alpha \in [0, 1]$ is the learning rate and s' is the state resulting from applying the action selected by policy π on state s.

$$\pi(s) = \underset{a}{\operatorname{argmax}}(r(s, a) + e^{-\beta t_a} V_t(s_{t+1})) \tag{3}$$

$$V_{t+1}(s) = V_t(s) + \alpha(r(s, a) + e^{-\beta t_a} V_t(s') - V_t(s)) \tag{4}$$

The pseudo-code used to implement this version of the algorithm can be seen in Algorithms 2 and 3. We started by setting a certain number of episodes, which represents the number of times we will run it to update the value function, where each episode runs for a specific number of iterations. What happens is that the algorithm starts in the initial state where all nodes have empty buffers, then, i sequential actions are performed. At the end of this cycle, the algorithm proceeds for the next episode, resetting the buffers to empty and starting over from the initial state, but with the already filled state-value function.

Inside each of the i iterations, what happens is the following: first, an action is chosen through the $\epsilon - greedy$ algorithm, balancing ϵ chance of choosing a random action and $1 - \epsilon$ odds of choosing the action by the policy at the beginning. Both ϵ and α (from Equation 4) parameters are decayed by half every 20% of the iterations..

B. MCTS Implementation

First, a root node is created with the current state of the network. At the start, all the nodes have their buffers empty, and the UAV is located at the Hub. Then, a node is chosen by exploring an unexplored action. From there, a simulation is rolled out with a pre-determined depth. At the end of the rollout, the final value is backpropagated to the node where the rollout started and all previous nodes. The values for the nodes generated during the rollout are not saved. After a few iterations, when the root node has all its child nodes explored, each with a different action, the Tree Policy is used to select the next node to go to, and so on, until another node that does not have all its actions explored is found. When that happens, again, a rollout is performed, and the same value

Algorithm 2: RL-implementation

```
for N Episodes do
    Initialize State;
    Initialize Action Space;
    if current episode % (N / 5) == 0 then
        α = α * 0.5;
        ε = ε * 0.5;
    end
    for i Iterations do
        Choose a random number 0 ≤ rnd ≤ 1;
        if rnd < ε then
            Select an action randomly;
        else
            Select the best action according to the
             Policy;
        end
        new_state ← Generate state transition
         according to the action selected and current
         state;
        Update state-value function accordingly;
        State ← new_state;
    end
end
```

Algorithm 3: RL-Extracting final result

```
Initialize State;
Initialize Action Space;
for i Iterations do
    Choose best action according to the Policy;
    new_state ← transition(old_state, action);
    results.append(action);
    old_state ← new_state;
    print(results);
end
```

update is done, as explained before. This process is done for a certain number of iterations or until a limit time constraint is reached. In the end, the child of the root node with the best value calculated by the policy is chosen, and the associated action is considered the best action in that state. Then, the same process is repeated with the selected child node being the new root, and so on. After doing this a few times, we have a sequence of actions from which a cyclic result can be extracted.

In Algorithm 6, an action is chosen to be simulated next according to the Tree Policy. When all of them were already explored, an action is chosen by a balanced choice between exploration and exploitation. The value calculated corresponds to the UCT formula, but instead of only using a constant C to adjust the exploration part, the value of the policy is used. The reasoning behind this is that the values that come out of the policy are large negative values, and this choice helps to normalize both terms to the same scale, similar to what was done in [15]. Also, it is possible to notice that a subtraction is made instead of a sum of the terms. The justification is that,

Algorithm 4: MCTS Algorithm applied to the problem described

```
Initialize Tree root with initial State;
for N Episodes - Amount of actions to predict do
    for N Iterations do
        Search(root_node, 0);
    end
    next_node ← best_action(root_node, 0);
    root_node ← next_node;
end
```

Algorithm 5: Search Function

```
if All actions were explored then
    if depth == MAX_DEPTH then
        return rollout();
    end
    tree_policy();
else
    Create node if it does not exist;
    Assign in the TT;
    rollout();
end
q ← reward + e^{-β*t} * search(next_node, depth + 1);
update_value(node, q);
return q;
```

as said before, the policy returns a negative value and as such, it has to be subtracted in order for the term $\frac{\ln(node.t\_visited)}{\sqrt{child.t\_visited}}$ to have the desired effect of incentivizing visits to the lesser-visited nodes.

Algorithm 6: Tree Policy

```
C ← 1;
for child ∈ node.children do
    value ← policy(node, child) − C *
     policy(node, child) * ln(node.t_visited) / sqrt(child.t_visited);
    if value > best_value then
        best_value ← value;
        best_child ← child;
    end
end
return best_child;
```

When a node is reached where not all of the possible actions were explored, the rollout phase begins (Algorithm 7). This step is very simple, the algorithm will start on the newly created node and choose actions at random consecutively until reaching a limit depth ($MAX\_ROLLOUT\_DEPTH$). When it does, it returns the reward for the last visited node and backtracks the value q for all the other states in the path until going back to the newly created node.

In order to update the value of each node after it has been visited more than once, the cumulative moving average (CMA) is calculated (Algorithm 8).

Algorithm 7: Rollout Function

if $depth == MAX\_ROLLOUT\_DEPTH$ **then**
  return $reward$;
end
Choose random action;
$q = r + \gamma * rollout(depth)$;

Algorithm 8: Update Value Function

$node.value = \frac{node.value * node.t\_visited + q}{node.t\_visited + 1}$;
$node.t\_visited+ = 1$;

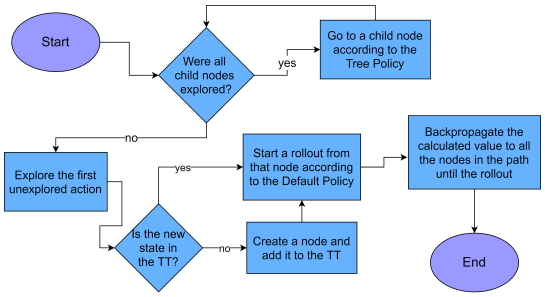


Fig. 4: Diagram of how nodes are selected and stored in the Transposition Table until reaching a leaf node.

One possible way of optimizing MCTS is by the use of Transposition Tables (TT) [10], [16]. A TT was implemented by storing a table linking possible states with references to nodes in the search tree. What happens is that a certain state will only happen once in the tree. If it happens again, the parent node will have the reference of the already existent node with that state as its child. The structure will no longer be a tree since a node can have several parents, or in other words, several previous state-action pairs that lead to it.

C. Deadlines Implementation

In the proposed problem we consider that the messages have a time limit before which they have to be delivered, counting from the time the message was created, or else it does not count as delivered. With the objective of making the algorithms take this new factor into consideration, we propose an alteration to the reward.

The original reward, shown in Equation 2, tries to minimize the number of messages in the network. The main idea proposed for this section is to still consider the previous reward, but at the same time give a lower reward to states that have lower Delivery Rate (DR). In order to adjust the value of the reward, we multiplied a new term by the original reward:

$$DR := \frac{Delivered + \eta}{Delivered + Expired + \eta}. \tag{5}$$

In order to adapt this term into the reward, the formula

$$e^{1-p*DR} + 0.5 \tag{6}$$

was used, where p is an adjustable parameter that depends on the rate of decay desired, and DR is the Delivery Rate as calculated by 5. The value 0.5 is used to stabilize to guarantee that the result is 0.5 when $DR = 1$. Adding one to $-p * DR$ guarantees that the result is decreasing for DR values going from 0 to 1.

In Algorithms 9 and 10 it is shown how the new term is adjusted into the reward. The former corresponds to a variation where the messages from all the nodes are summed, and then the total DR is calculated. The latter corresponds to calculating the individual DR for each node and then selecting the lowest value. The first option has the objective of maximizing the total DR independently of one or several of the nodes performing poorly. On the other hand, the second option tries to optimize the worst-performing node, making the DR have more balanced values between the different nodes.

Algorithm 9: RL - Total DR, how the reward is calculated.

$DR = \frac{\sum_{i=0}^{N} Delivered_i}{\sum_{i=0}^{N} Delivered_i + \sum_{i=0}^{N} Expired_i}$;
$dr\_term = e^{1-3*DR} + 0.5$, *by the use of Equation 6*;
$r(s,a) = -\int_0^{t_a} (Ft + N_0) e^{-\beta t} dt * dr\_term$;

Algorithm 10: RL - Lowest DR, how the reward is calculated.

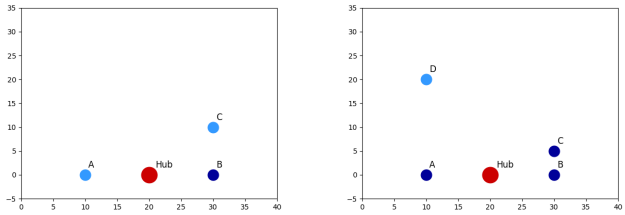
$DR = \min_{i=0..N} \frac{Delivered_i}{Delivered_i + Expired_i}$;
$dr\_term = e^{1-3*DR} + 0.5$, *by the use of Equation 6*;
$r(s,a) = -\int_0^{t_a} (Ft + N_0) e^{-\beta t} dt * dr\_term$;

IV. Evaluation and Comparison

In order to evaluate the performance of the algorithms, they are first compared without taking into account any limitation to the amount of time that each message has to be delivered (deadline). Further on, the new reward system for the deadlines is evaluated and compared with the previous approaches that do not make this consideration. Four maps are considered to verify the efficiency of the algorithms (Figure 5). These scenarios are approximations of the ones used in [6] since no exact values are given. In each of the networks, there is a Hub Node, represented in red, low-rate nodes, in light blue and high-rate nodes, colored with dark blue. The low-rate nodes transmit packets at a rate of 2 per time unit. On the other hand, high transmission-rate nodes transmit at a rate of 7 packets per time unit. Having different rates will create an uneven network, making it more realistic.

A. Evaluated Metrics

In order to compare the algorithms, we evaluate the average delay with which the messages were deleted. For the case where deadlines are considered, two additional terms are compared: the overall DR of the network and the DR of the node with the lowest associated value (Lowest DR).



(a) Network 1 (b) Network 2

(c) Network 3 (d) Network 4

Fig. 5: The four networks used to evaluate the algorithms: Red: Hub; Dark-blue: High generation rate node; Light-blue: Low generation rate node.

B. List of Algorithms Evaluated

In the evaluation tables in the next section, several algorithms will be compared with slight variations on some of them. In this section, we present a list of those algorithms, as well as a short description, in order to ensure a better grasp of the results.

- RL - Original [6]: This represents the original RL solution with the original reward, which does not consider deadlines.
- RL - Total DR*: The first adaptation, which uses the total Delivery Ratio between all nodes to adjust the reward, in order to ensure a better network DR.
- RL - Lowest DR*: The second adaptation used. Now the DR is calculated separately for each node, and then the lowest one will be chosen to adjust the reward calculation. This aims to ensure that no Node is left without visits, and thus corresponds to a fairer system.
- MCTS: The reward and policy used in RL are brought into an MCTS-like algorithm where the search is done by constructing a tree.
- TSP Path: The path given by solving the Travelling Salesman Problem. Since the networks have a small number of nodes, this can be easily calculated by brute force almost instantly. Otherwise, a near-optimal solution could be found by using, for example, a genetic algorithm, between other options.
- Round Robin: The path consists of going to each node once and then coming back to the hub. This path can be easily generated without further calculations.

*These algorithms are only relevant for the section considering deadlines. The evaluation prior to deadlines does not employ them.

C. Results for the Hub Scenario

In this subsection, RL, MCTS, TSP, and Round Robin algorithms are compared with each other. A sum-up of all the used parameters is presented in Table I for RL and in Table II for MCTS.

TABLE I: Parameters used for RL. These parameters do not vary between networks.

Parameter	Initial Value	Decay
Number of Episodes	10 000	—
Depth of Search	20	—
Deadline	100	—
Max Size	300	—
Step Size	20	—
ϵ (Exploration Rate)	0.3	$\frac{\epsilon}{2}$ *for every 20% of the episodes.*
α (Learning Rate)	0.3	$\frac{\alpha}{2}$ *for every 20% of the episodes.*
β	0.01	—
γ	$e^{-\beta * t}$	—

TABLE II: Parameters used for MCTS. These parameters do not vary between networks.

Parameter	Initial Value	Decay
Number of Episodes	20	—
Number of Iterations	50000	—
Max Depth	15	—
Max Rollout Depth	15	—
Deadline	100	—
Max Size	300	—
Step Size	20	—
ϵ (Exploration Rate)	0.3	$\frac{\epsilon}{2}$ *for every 20% of the episodes.*
α (Learning Rate)	0.3	$\frac{\alpha}{2}$ *for every 20% of the episodes.*
β	0.01	—
γ	$e^{-\beta * t}$	—

1) Results without Deadlines

In Figure 6 a sum up of the results for this section is presented. It is possible to verify that both RL and MCTS algorithms perform better than both TSP and RR for the 4 networks tested. RL also has a small edge of around 5%, depending on the network, over the MCTS version. It was possible to verify that with the increase in the size of the networks, by changing the number of nodes, the execution time rises steeply, making larger problems hard to compute in an adequate time frame.

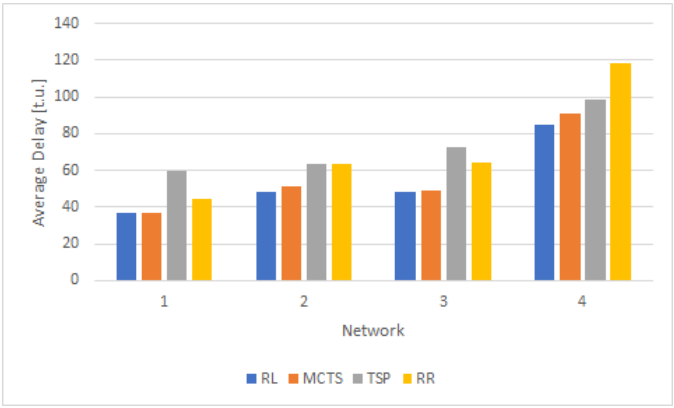


Fig. 6: Hub Scenario average delay without deadlines.

MCTS takes a longer time running in all the maps and ends up using more memory for larger ones, when compared to RL. The relative gain (RG) is calculated as the reduction in average delay compared to the worst-performing algorithm in percentage, for each network

$$RG = \frac{baseline - target}{baseline} * 100, \quad (7)$$

where the $baseline$ is the highest AD value and the $target$ is the algorithm for which the RG is being calculated. RG is represented in Table III.

TABLE III: Relative gain in terms of average delay for each of the 4 Networks. Calculated with Equation 7.

Relative Gain [%]	Network 1	Network 2	Network 3	Network 4
RL	38.9	24.6	33.6	28.3
MCTS	38.9	19.7	32.4	23.0
TSP	0.0	0.0	0.0	16.6
RR	26.1	0.0	11.8	0.0

2) Results with Deadlines

For this phase, the same comparison will be made, but in this case, for the RL and the MCTS algorithms, the reward used is the one that considers the existence of deadlines.

From all the tested networks, Network 4 is the one with the most exciting results. As seen before, this network is the largest one tested, having five nodes plus the hub. It is possible to notice from Figures 7, 8 and 9 that the original reward does not do well, and ends up with a worse DR and not visiting two nodes. On the other hand, both the reward that considers the total DR and the reward that considers the Lowest DR between the nodes, show better results. The former has the highest DR, with 68.7% of the messages being delivered. The latter has a slightly lower DR but has the advantage of visiting all the nodes every once in a while, having a higher lowest delivery rate of 37.5% instead of 0% as in the other two. The average delay is somewhat consistent between them. Still, since it is only related to the messages that were actually delivered, the DR factor shows that the two new rewards are objectively better than the original, at least for this network.

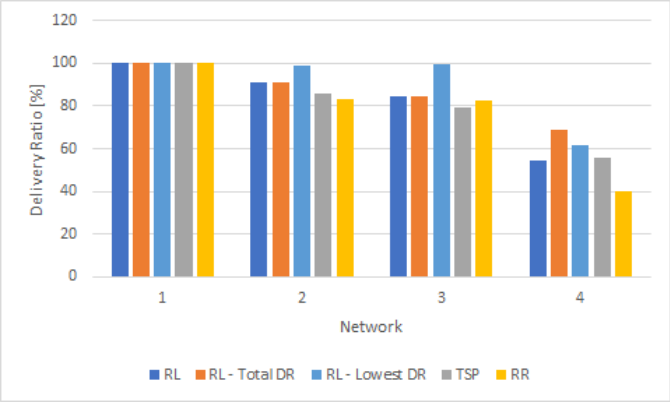


Fig. 7: Hub Scenario DR with deadlines.

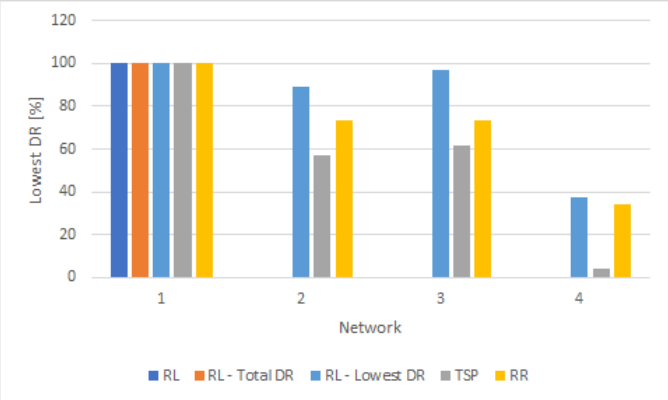


Fig. 8: Hub Scenario Lowest DR with deadlines.

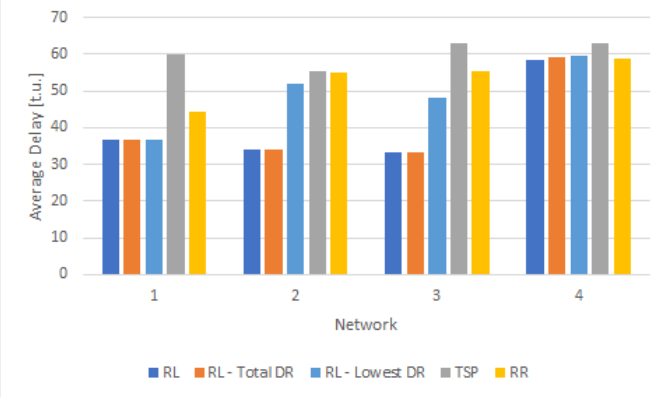


Fig. 9: Hub Scenario Average Delay with deadlines.

D. Results for the All-to-All Scenario

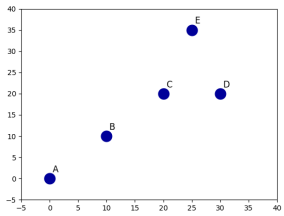
New networks were created in order to evaluate the algorithms in Scenario 2 (Figure 10). Now only the *RL - Lowest DR* version will be tested and compared to the *TSP path* since it was the best performing one.

For this scenario, the parameters are given in table IV. It is possible to notice that the number of episodes and depth of search is higher than the ones used in the Hub scenario. This is due to two factors. First, each action is now simpler to calculate, but more actions are needed in order to form a solution. This leads to an increase in the depth of search. Second, now some of the networks tested have a larger number of nodes than before, and the scenario is more complicated. As such, the number of required episodes needs to be large enough to guarantee that increasing it further would not results in an improvement of the results.

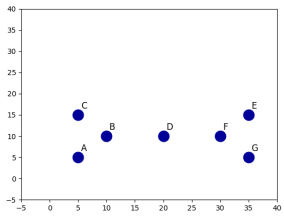
TABLE IV: Parameters used for RL in the All-to-All Scenario

Parameters	Initial Value	Decay
Number of Episodes	100 000	—
Depth of Search	1000	—
ϵ (Exploration Rate)	0.3	$\frac{\epsilon}{2}$ *for every 20% of the episodes.*
α (Learning Rate)	10^{-9}	$\frac{\alpha}{10}$ *for every 20% of the episodes.*
β	0.01	—
γ	$e^{-\beta * t}$	—

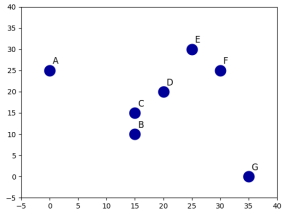
In Tables V, VI, VII and VIII the generation rates for Networks 10a through 10d are shown. On the rows, the node that is generating the messages is represented, followed by its respective message generation rate towards every other node.



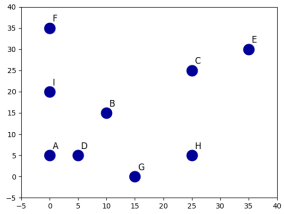
(a) Network 5



(b) Network 6



(c) Network 7



(d) Network 8

Fig. 10: The four networks used to evaluate the all-to-all scenario.

TABLE V: Network 5 generation rates. Nodes A and C have the highest total generation rates.

	A	B	C	D	E
A	0	2	7	2	7
B	2	0	2	2	7
C	3	1	0	8	7
D	2	2	2	0	7
E	2	2	2	6	0

TABLE VI: Network 6 generation rates. Nodes A, B, C and D have high generation rates towards each other.

	A	B	C	D	E	F	G
A	0	9	9	9	2	2	2
B	9	0	9	9	2	2	2
C	9	9	0	9	2	2	2
D	9	9	9	0	2	2	2
E	2	2	2	2	0	2	2
F	2	2	2	2	2	0	2
G	2	2	2	2	2	2	0

TABLE VII: Network 7 generation rates. Nodes B, C, D, E and F have high generation rates towards each other.

	A	B	C	D	E	F	G
A	0	2	2	2	2	2	2
B	2	0	9	9	9	9	2
C	2	9	0	9	9	9	2
D	2	9	9	0	9	9	2
E	2	9	9	9	0	9	2
F	2	9	9	9	9	0	2
G	2	2	2	2	2	2	0

TABLE VIII: Network 8 generation rates. Nodes B, C, D, E and F have high generation rates towards each other.

	A	B	C	D	E	F	G	H	I
A	0	9	9	2	2	2	2	2	2
B	2	0	9	2	2	2	2	2	2
C	2	9	0	2	2	2	2	2	2
D	2	9	9	0	2	2	2	2	2
E	2	9	9	2	0	2	2	2	2
F	2	9	9	2	2	0	2	2	2
G	2	9	9	2	2	2	0	2	2
H	2	9	9	2	2	2	2	0	2
I	2	9	9	2	2	2	2	2	0

1) Results without Deadlines

First, the results without considering deadlines are presented. The Networks evaluated are the four assessed before plus the four corresponding to the Hub Scenario.

The first four networks (which correspond to the ones studied in the Hub Scenario) can be implemented as an All-to-All scenario by outputting a generating rate of 0 from all nodes to each other, except for the Hub Node. The hub, on the other hand, has all generation rates equal to 0. The hub node is now considered as a regular node, and the actions taken are from any node to any other node instead of being complete paths from hub to hub.

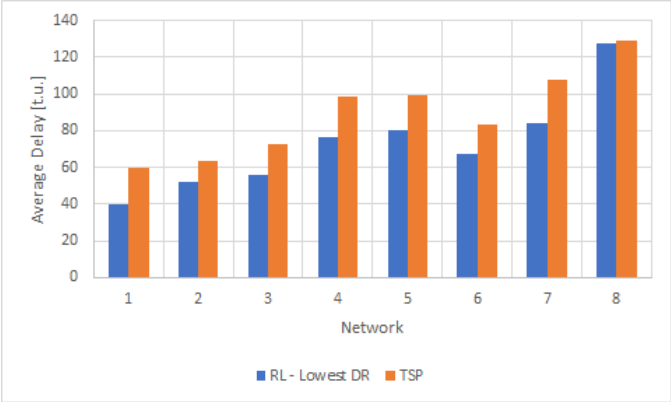


Fig. 11: Average Delay for the 8 Networks.

For the results without deadlines, it is possible to observe that in the networks that corresponded to the Hub Scenario the results are very similar to the ones achieved before (Figure 6), being only slightly worst in the All-to-All implementation of the Hub Scenario (Figure 11). The differences in the results achieved might be related to the method used to approximate the values of the states being linear as well as due to the action space allowing several visits to the same node before reaching the hub node again, which is always harmful for the Hub Scenario.

The results obtained for the four new networks were slightly better than the TSP. We believe the results could still be improved on when considering larger networks by using a non-linear approximation method like Neural Networks, which would be able to better estimate the value of each state considering the features used.

In order to get a better overall evaluation of the algorithm, random networks were simulated with a different number of

nodes, where an exponential distribution gives the generation rates rounded to the closest integer. As before, the generation rate from a node to itself is always 0. The positions of the nodes are randomly generated in a 50 by 50 grid with the restriction that the coordinates of the nodes have to be a multiple of 5 (e.g.: (15,5), (40,50), (0,10), ...).

The parameters used for the test are given in Table IX. Networks were grouped by the number of nodes. One hundred networks were generated for each number of nodes between 3 and 9. In the end, the average delay in a simulation is calculated for both the path obtained from training and TSP. Afterward, the RGs (Equation 7) are calculated where the $baseline$ is the average delay calculated with the TSP algorithm solution, and the target is the path our algorithm produced. Finally, the RGs are averaged between the 100 networks for each number of nodes, and the sample standard deviation is calculated. The results for this process can be seen in Figure 12.

TABLE IX: Parameters used to simulate each of the randomly generated networks.

Parameter	Initial Value	Decay
Number of episodes	100000	-
Depth	1000	-
Deadline	∞	-
ϵ	0.3	$\frac{\epsilon}{2}$ *for every 20% of the episodes.*
α	10^{-9}	$\frac{\alpha}{10}$ *for every 20% of the episodes.*
Exp Distribution factor (γ)	0.5	-
Map dimensions	50 by 50	-

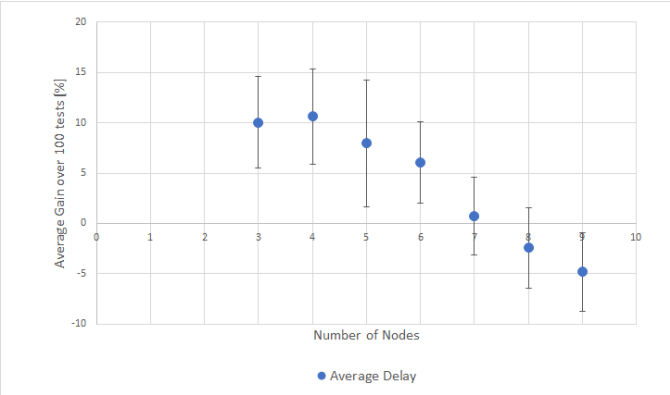


Fig. 12: Average gain per number of nodes in percentage.

From the figure, it is possible to notice that there is a decline in the improvement obtained over the TSP as the number of nodes increases. This can be due to the fact that the linear approximation lacks the capability of capturing the relations between different features, which is aggravated by the fact that the number of features grows polynomially with the number of nodes, as described in the previous chapter.

2) Results with Deadlines

In this section, the results for the simulation with deadlines are presented (Figures 13, 14, 15). Note that for Network 7 and Network 8 the deadline value used was 150 instead of 100 due to the size of those networks being larger.

Overall it was possible to notice that the execution time increases linearly with the number of nodes in the network. An interesting result to extract from the first four networks is that the algorithm was able to have similar results to the previous approach shown for Scenario 1, in equivalent networks.

For the completely new networks, 5 through 8, the results were slightly better than the TSP, apart from Network 8, where the results were slightly worse. As was already seen previously, with an increase in the number of nodes, the algorithm approximator has a higher difficulty finding a set of weights that can adequately represent a value-function for the network.

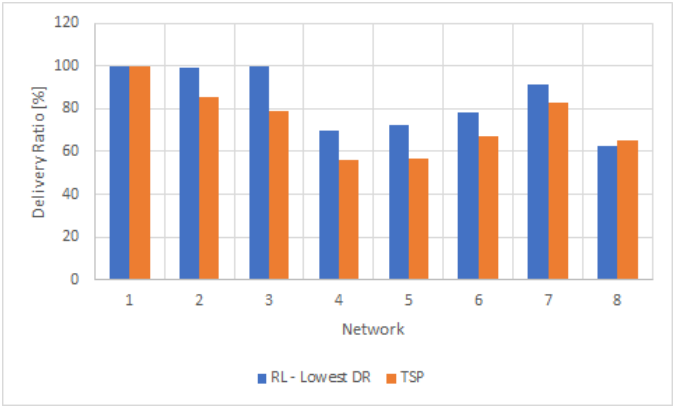


Fig. 13: Delivery Ratio for the 8 Networks.

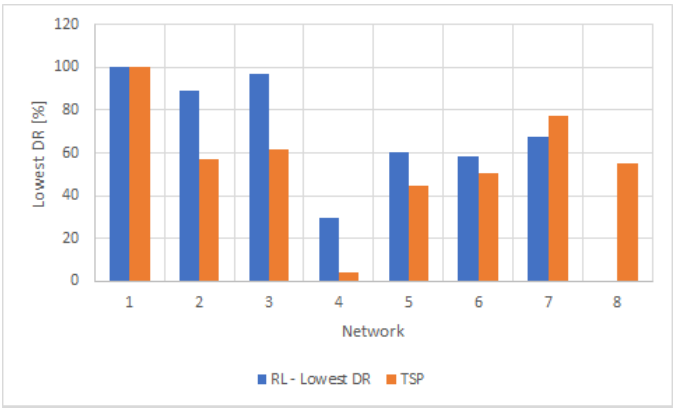


Fig. 14: Lowest Delivery Ratio for the 8 Networks.

In a similar fashion to what was done for the results without deadlines, an exhaustive simulation using random networks was run. The parameters used are the same as before (Table IX), with the only difference being the deadline used now has the value 100.

The results can be seen in Figure 16. For this deadline value, it is possible to notice that there are around 10% to 20% gains in the Average Delay and 20% to 30% gains in the total Delivery Ratio of the Network. For the larger networks, the algorithm ends up ignoring one or more nodes in order to maintain a higher total DR.

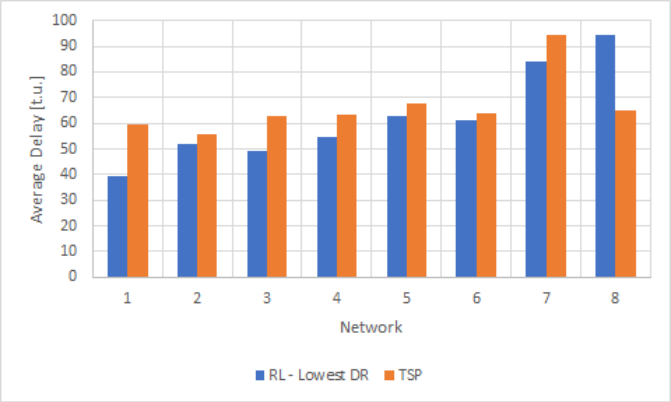


Fig. 15: Average Delay for the 8 Networks.

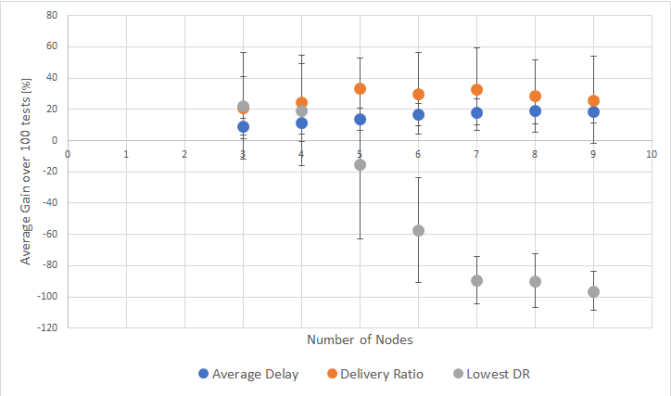


Fig. 16: Average gain per number of nodes in terms of Average Delay,Delivery Ratio and Lowest Delivery Ratio. Simulated with Deadline = 100.

V. Conclusion

A new way of connecting nodes in small networks was studied for two kinds of scenarios. It was proved that the resolution of the second scenario would also bring a solution for the first one since it is a simplification of the All-to-All. The baseline RL algorithm used, called TD(0), proved to have better results than both the MCTS or a more straightforward approach like the TSP for small networks. For larger networks, both algorithms proved to be too slow regarding computing efficiency, and MCTS also proved to use a large amount of memory when the networks start to grow in size (number of nodes). More work needs to be done in order to improve solutions on larger networks. In Section V-A, we present some possibilities for future work in order to improve the results presented.

A. Future Work

Several approaches can optimize the results we obtained. Due to the time frame of the work, it was not possible to test with many different hyper-parameters combinations. As such, we suggest trying out different values for the number of episodes, α, ϵ, depth, β, etc.

The use of another function approximator for the RL approach might also be interesting. Instead of using a linear function approximator, it would be possible to use a non-linear approximation (e.g., Artificial Neural Network), which might improve the results drastically for larger networks. Other feature representations for the approximation can also be used and experimented with.

Further on, it would also be attractive to work out the path without having the knowledge of the generation rates for each node. Finally, the idea of generating paths for multiple UAVs in the same network, which work together for a better result, is fascinating, although such a multi-agent system entails much higher complexity.

References

[1] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*. Springer, 2015.

[2] C. Barroca, A. Grilo, and P. R. Pereira, "Improving message delivery in UAV-based delay tolerant networks," *Proceedings of 2018 16th International Conference on Intelligent Transport System Telecommunications, ITST 2018*, 2018.

[3] W. Zhao, M. Ammar, and E. Zegura, "Controlling the mobility of multiple data transport ferries in a delay-tolerant network," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 2. IEEE, 2005, pp. 1407–1418.

[4] Z. Zhang and Z. Fei, "Route design for multiple ferries in delay tolerant networks," in *2007 IEEE Wireless Communications and Networking Conference*. IEEE, 2007, pp. 3460–3465.

[5] H. Guo, J. Li, and Y. Qian, "Hop-dtn: Modeling and evaluation of homing-pigeon-based delay-tolerant networks," *IEEE Transactions on Vehicular Technology*, vol. 59, pp. 857–868, 2010.

[6] D. Henkel and T. X. Brown, "Towards autonomous data ferry route design through reinforcement learning," *2008 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, WoWMoM2008*, 2008.

[7] P. Smith, R. Hunjed, A. Aleti, and J. C. Barca, "Adaptive data transfer methods via policy evolution for uav swarms," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2017, pp. 1–8.

[8] K. Li, W. Ni, E. Tovar, and A. Jamalipour, "On-board deep q-network for uav-assisted online power transfer and data collection," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12 215–12 226, 2019.

[9] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[10] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved monte-carlo search," *Univ. Tartu, Estonia, Tech. Rep*, vol. 1, 2006.

[11] T. Vodopivec, S. Samothrakis, and B. Ster, "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.

[12] Richard S. Sutton, Andrew G. Barto, *Reinforcement Learning: An Introduction*. the MIT Press, 2018.

[13] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[14] Marco Wiering, Martjin van Otterlo (Eds., *Reinforcement Learning: State-of-the-Art*. Springer Heidelberg New York Dordrecht London, 2012.

[15] R.-K. Balla and A. Fern, "Uct for tactical assault planning in real-time strategy games," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[16] B. E. Childs, J. H. Brodeur, and L. Kocsis, "Transpositions and move groups in monte carlo tree search," in *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 2008, pp. 389–395.