

Nested Rollout Policy Adaptation for Optimizing Vehicle Selection in Complex VRPs*

Ashraf Abdo, Stefan Edelkamp, Michael Lawo

*Faculty of Mathematics and Computer Science- Institute for Artificial Intelligence
University of Bremen - TZI - Centre for Computing and Communication Technologies
Bremen, Germany*

ashraf@uni-bremen.de

{edelkamp, mlawo}@tzi.de

Abstract - The goal of Vehicle Routing Problems (VRP) and their variations is to transport a set of orders with the minimum number of vehicles at least cost. Most approaches are designed to solve specific problem variations independently, whereas in real world applications, those constraints are handled concurrently. This paper describes a novel approach to solve variants of Open VRP, Multi Depot VRP, Capacitated VRP as well as Pickup and Delivery with Time Windows applying Monte-Carlo Tree Search (MCTS) and in particular Nested Rollout Policy Adaptation. For evaluation, real data from the industry was obtained and tested on the developed approach.

Vehicle Routing, Monte Carlo Methods, Nested Rollout Policy Adaptation.

I. INTRODUCTION

In the area of AI search and game playing, the raise of randomized state space search for optimization can be recognized with algorithms that are called Monte-Carlo Tree Search (MCTS). The search is based on random game samples, known as rollouts.

MCTS has impressed with many high scores in programs that play one-player games (for example Klondike Solitaire, SameGame and Morpion Solitaire). It is also very effective in two-player board games (for example Hex, Havannah, Magic: The Gathering, Game of the Amazons, and Arimaa), in real-time video games (for instance PocMan, Fable Legends and Total War: Rome II), and in non-deterministic games (such as Scrabble, Bridge, Magic: The Gathering, or Settlers of Catan) [1]. The probably most visible success is the defeat of Go world champion player Lee Sedol, by a combination of deep learning and MCTS².

In computational biology, challenging multiple-sequence alignment problems were solved with MCTS superior to existing solutions [1]. MCTS algorithms like *nested rollout policy adaptation* (NRPA) [2] have also been successfully ported to logistics applications, solving challenging 3D container packing [3] as well as single-vehicle routing problems with time windows [4] and pickup and deliveries [5]. This paper presents a novel approach to solve complex multi-vehicles routing problem using the NRPA variation of the Monte-Carlo search. The paper is structured as follows: first, we outline the MCTS variants which were used to solve transportation logistics problem, providing an overview of each

algorithm. Next, we introduce the developed approach for solving highly constrained VRP, starting with a mathematical formulation of the problem to proceed with the explanation of the search function and then to the selection procedure for the successor nodes and vehicles. In the results section we discuss two different policy initialization techniques and show how the developed approach could be applied successfully in regular operation of real-world logistics process.

II. RELATED WORK

Academic work in MCTS strives to find different algorithm variants and improvements so that each of these improvements matches its application domain the best and can be further extended to broader perspectives. In this part of the paper, the focus will be only on MCTS variations that have been used to solve transportation logistics problems like TSP and VRP.

A. Nested Monte Carlo Tree Search (NMCTS)

This variation of the Monte Carlo algorithm, introduced in 2009 [6], guides the search at a given level of the search tree by the result of the searches at the lower levels. On the base level of the search tree, random moves are performed, and then the best sequence of moves is memorised to improve the search of the higher levels.

NMCTS combines nested calls with randomness in the playouts and memorization of the best sequence of moves. The NMCS function performs one gameplay (rollout), choosing at each step in the game, a move that has the highest score of the lower level NMCS. At each step, the algorithm tries all possible moves, plays a nested search at the bottom level after each move, and memorises the move associated with the best score of the lower level searches.

Nested rollouts (playouts), the rollouts are based on a heuristic. It implies that nested rollouts always improve the previously performed rollouts, and they also improve on just simply following the heuristic. In the base level, it is possible that the search does not use heuristics, but rather random moves. This in turn arises the possibility of a nested search giving worse results than a lower level search. Therefore, the nested search of a high level will lead without certainty to a better result than the previous ones or even lower level searches.

* This work is part of the Erasmus Mundus Master programme in Pervasive Computing and Communication for Sustainable Development (PERCCOM) [12] of the European Union www.perccom.eu. The authors thank all the partner institutions, sponsors and researchers of the PERCCOM program.

² <https://www.deepmind.com/alpha-go.html>

To keep track of the best moves and of the corresponding sequences already found by a previous search, the algorithm has to keep the best sequence found so far (Algorithm 1 *Nested Monte Carlo Search Algorithm as presented by [6] on a maximization problem*. line 11), in order to follow it when the randomised searches give worse results than the best-known sequence so far. Otherwise, and when the search succeeds in finding a better solution, the best sequence will be updated with the newly discovered sequence and the best move is played [6].

```

1 nested (position, level)
2 best score = -infinity
3 while not end of game
4 if level is 0
5   move = rollout (position, m)
6 else
7   move = nested (sample (position, m), level - 1)
8 if ( score of move > best score)
9   best score = score of move
10  best sequence = seq. after move
11  bestMove = move of best sequence
12  position = play (position,bestMove)
13 return score

```

Algorithm 1 Nested Monte Carlo Search Algorithm as presented by [6] on a maximization problem.

While introducing the NMCTS, Cazenave [6] demonstrated how memorising the best sequence of moves improves the mean results of the search by running the algorithm on three different games. The games were Morpion Solitaire, SameGame, and 16x16 Sudoku. Later, the Nested Monte Carlo search was used to solve the Travelling Salesman Problem with Time Windows (TSPTW). The algorithm was able to compute state of the art results for problems of a size less than 30 stops in a small computation time. For a full overview of this implementation, the reader is referred to [7].

B. NRPA

The MCTS algorithm and its enhancement Nested Monte Carlo Tree Search (NMCTS) uses static uniform random or domain-specific policies to choose the successor moves.

Rosin [2] proposed an enhancement to the NMCTS Algorithm of [6]. This new algorithm dynamically adapts the successors move choice policy during the search, and is called Nested Rollouts with Policy Adaptation (NRPA).

The rollout policy in the implementation is a matrix or a vector of decimal values representing weights that are used to calculate the likelihood of choosing a certain move during the game [8]. Starting from a random node in the search tree, and trying from there to get an effective policy to guide the search towards good results will not work [2]. However, the policy in NRPA is most effective when it builds on the best found solutions in the search space. The policy would direct the search towards the neighbourhoods of such solutions and help driving future rollouts towards them.

Algorithm 2 resembles the pseudocode of the NRPA algorithm with parts in bold highlighting the main difference from the preceding NMCTS.

One can observe that the choice between one of the successors of a node has been replaced from a uniform choice, to a choice based on the policy value of the node (lines 5 and 6), which represents the probability to choose a certain node as a next move starting from the current position.

When NRPA succeeds in finding a better solution, the adapt function is called (line 18) to update the policy; so the search can be directed towards the neighbourhood of this good result.

```

1 NRPA(level,pol):
2   if level is 0: // base rollout policy
3     node = root(), rollouts = 0, seq = {}
4     while num_children(node) > 0:
5       CHOOSE seq[play] = child i with probability
6         proportional to exp(pol[code(node,i)])
7       node = rollout(node,seq[play])
8       rollouts += 1
9   return (score(node),seq)
10
11 else: // for nesting levels>=1
12   best_score = -infinity
13   for N iterations:
14     (result,new) = NRPA(level-1,pol)
15     if (result >= best_score) THEN:
16       best_score = result
17       seq = new
18       pol = Adapt(pol,seq)
19   return (best_score,seq)

```

Algorithm 2 NRPA algorithm as presented by [2] on a maximization problem.

NRPA improved the solutions of the Travelling Salesman Problem (TSP) obtained by [7] using Nested Monte Carlo Search [8].

The algorithm succeeded in giving quality results, which in many cases matched the state of the art solutions, without utilizing any expert knowledge or heuristics while solving the problem. Applying NRPA with expert knowledge of the TSP, the solution provided by the algorithm matched the state of the art solutions for 23 out of 30 problems from a standard benchmark set. Two important works following [8] in using NRPA to solve TSP were the research documented by [4] and [5].

In the first research, several enhancements for the algorithm were discussed, ranging from simple steps like avoiding the usage of copy constructors, to reducing the memory overhead, in addition to applying other algorithm engineering measures. The solver method presented in the paper was later adopted in an actual implementation of a multi-agent based system dealing with dynamic scheduling problems. In that system, each agent encompasses a version of the solver so that the agent can compute its tour independently and later negotiate with other agents based on the result obtained.

The second research succeeded in using NRPA to solve “hybrid” TSP with pickup and deliveries(TSPPD) problems that also involved other constraints like capacity and time window. It also implemented all the enhancements suggested by the first research into the solver.

Other examples of the usage of Monte Carlo methods in logistics domain were presented by [3].

We propose a simple centralized solution that optimizes the travel cost function (usually representing time or distance covered) while also reducing the number of vehicles. In the exposition we will give a formal definition of the complex VRP, explain the approach's core aspects and then we present results of experiments run on the developed solution.

The main incentive of using MCTS, however, is its flexibility, its slim implementation, and the limited amount of expert knowledge that has to be put in. Besides time windows, service times, and uniform capacity constraints that are found in classical benchmarks, the general vehicle routing planner based on MCTS we propose is much more flexible. It

- supports a heterogeneous fleet of vehicles
- dynamically reduces the number of vehicles
- respects individual volume and capacity restriction
- satisfies pickup and delivery constraints
- finds Hamiltonian paths (rather than only cycles)
- handles several individual depots
- supports user-defined selection rules

For this approach, we exploit a refined implementation of NRPA. One research question addressed in this paper is on how to improve move selection in a Monte-Carlo algorithm.

III. ALGORITHM DESCRIPTION

A. Mathematical formulation

The terminology presented here extends the one used by [9] which is in turn based on the book of [10].

For a VRP problem that considers pickup and delivery, multiple depots, capacity constraints, time windows, free tour ends, and heterogeneous vehicles (in terms of starting locations, capacity, and operating times).

There are n number of requests which will be served by a maximum of m available vehicles.

The problem is defined on a graph where:

$K = \{0, \dots, m-1\}$ is the set of vehicles.

$P = \{2(m+i), \dots, 2(m+n)-2\}$ for $i \in \{0, \dots, n-1\}$ is the set of pickups.

$D = \{2(m+i)+1, \dots, 2(m+n)-1\}$ for $i \in \{0, \dots, n-1\}$ is the set of deliveries.

Each vehicle k belongs to the set V and has a start and an end terminal represented in the graph as nodes: t_k and t'_k respectively.

$$\begin{aligned} t_k &= 2k \text{ for } k \in \{0, \dots, m-1\} \\ t'_k &= 2k+1 \text{ for } k \in \{0, \dots, m-1\} \end{aligned}$$

Each request i is represented by two nodes on the graph, the pickup location $P_i = 2(m+i)$ for $i \in \{0, \dots, n-1\}$ and $P_i \in P$ and delivery location $D_i = 2(m+i)+1$ for $i \in \{0, \dots, n-1\}$ and $D_i \in D$.

Figure 1 An example of nodes representations for a problem with 5 available vehicles and 40 orders illustrates the memory layout of the nodes represented in case $m=5$ and $n=40$

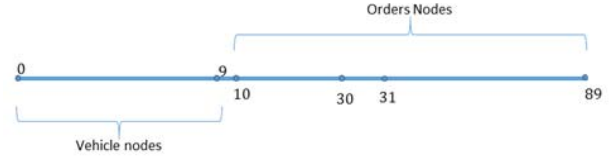


Figure 1 An example of nodes representations for a problem with 5 available vehicles and 40 orders

As one vehicle might not be able to serve all requests, one has to maintain that the vehicle which served the pickup would be the one to serve the delivery. These kinds of limitations are modelled by letting $P_k \subseteq P$ and $D_k \subseteq D$ be subsets of pickups and deliveries served by vehicle k . Since every request is serviced by the same vehicle, it can be assumed that $i \in P_k \Leftrightarrow i+1 \in D_k$, i.e. that both the pickup and delivery can be serviced by vehicle k .

Define $N = P \cup D$ and $N_k = P_k \cup D_k$ the directed graph $G = (V, A)$ consists of the nodes $V = \{t_0, \dots, t_{m-1}\} \cup \{t'_0, \dots, t'_{m-1}\} \cup N$ and the arcs $A = V \times V$.

For each vehicle, exists a subgraph $G_k = (V_k, A_k)$, with $V_k = \{t_k\} \cup \{t'_k\} \cup N_k$ and $A_k = V_k \times V_k$. For each edge $(i, j) \in A$ a travel time $t_{ij} \geq 0$ is assigned.

It is assumed that all travel times satisfy the triangle inequality i.e. $t_{ij} \leq t_{il} + t_{lj}$ for $i, j, l \in V$.

A service time s_i and a time window $[a_i, b_i]$ is assigned to each node $i \in V \setminus (T_k \cup T'_k)$.

This means that service time is considered at all locations except at the start and end of each individual tour. The service time represents the time needed for loading and unloading and the time window indicates when the visit at the particular site should start. A visit to node i can only take place between time a_i and b_i . A vehicle is allowed to arrive at a site before the start of the time window, but it has to wait until the start of the time window before the visit can be performed.

For each node $i \in N$, a weight w_i is assigned for goods that should be loaded onto the vehicle at the particular node.

Providing that $w_i \geq 0, i \in P$, and $w_i = -w_{i-1}, i \in D$, each vehicle $k \in K$ has room for a certain amount of goods; this capacity is given by C_k . Each vehicle k should follow a legal route from its start terminal t_k to its destination t'_k . A legal route r is a simple (loop-free) path:

$$\bar{r} = (t_k, v_1, v_2, \dots, v_h, t'_k)$$

That loop-free path satisfies the time window constraints set by the customers, the capacity of the vehicle, the precedence of a pickup over delivery, and only requests that can be serviced with vehicle k .

$$i \leq j, v_i \in P_k, \quad v_j \in D_k, \quad v_j = v_i + 1$$

The previous condition represents the pickup and delivery precedence condition and that a pickup-delivery pair should be served by the same vehicle.

To ensure that time windows are satisfied, $S_i \in Z_0^+$ is defined to denote when the vehicle starts the service at site v_i . The above will be translated into the following constraints

$$\bar{r} = (t_k, v_1, v_2, \dots, v_h, t'_k)$$

with

$$\begin{aligned} a_{v_i} &\leq S_i \leq b_{v_i} \quad i = 1, \dots, h \\ S_{i+1} &\geq S_i + s_i + t_{v_i, v_{i+1}} \\ a_{t_k} &\leq S_1 \leq b_{t_k} \\ a_{t'_k} &\leq S_h \leq b_{t'_k} \end{aligned}$$

Where $[a_{t_k}, b_{t_k}]$ is the time window of terminal t_k (the depot) and $[a_{t'_k}, b_{t'_k}]$ is the time window of terminal t'_k . Finally, the capacity of the vehicle should be respected throughout the path. For this purpose, $L_i \in Z_0^+$ denotes the load of the vehicle at node i after serving this node i . We have:

$$\begin{aligned} L_i &\leq C_k \quad i = 1, \dots, h \\ L_{i+1} &= L_i + l_{i+1} \\ L_1 &= 0 \\ L_h &= 0 \end{aligned}$$

The travel cost of a given route $\bar{r}$ is

$$c_{\bar{r}} = \sum_{i=0}^h t_{v_i, v_{i+1}}$$

If there were still any unmet requests after the search ends, they would be added to the end of the tour without having their cost calculated.

The whole problem can now be formulated as follows: let R be the set of all feasible routes. The Boolean matrix $(\alpha_{j\bar{r}})$ for $\bar{r} \in R$ and $j = 1, \dots, n$ denotes whether request j is serviced using route $\bar{r}$. The Boolean matrix $(\beta_{k\bar{r}})$ for $\bar{r} \in R$ and $k = 1, \dots, m$ denotes whether the route $\bar{r}$ is carried out by vehicle k . Using binary variables $x_{\bar{r}}$ to indicate whether route $\bar{r}$ is used in the solution, the following model is introduced:

$$\begin{aligned} \min f(x) &= \sum_{\bar{r} \in R} c_{\bar{r}} x_{\bar{r}} \\ \text{Subject to } \sum_{\bar{r} \in R} \alpha_{j\bar{r}} x_{\bar{r}} &= 1 \quad j = 0, \dots, n-1 \\ \sum_{\bar{r} \in R} \beta_{k\bar{r}} x_{\bar{r}} &= 1 \quad k = 0, \dots, m-1 \\ x_{\bar{r}} &\in \{0,1\} \quad \bar{r} \in R \end{aligned}$$

B. Search and Rollout

The algorithm we propose extends NMCTS with policy adaptation [5] for solving the TSP. The core function in the optimisation is the search procedure presented in *Algorithm 3 NRPA search function* which is a

recursive function and is the main procedure of this implementation. It shows how the level-specific search policy of a level l is adapted if a better solution at level $l-1$ has been found. As soon as all iterations have been executed, the global policy applied by the *rollout* function is overwritten. If the algorithm reaches the lower level 0, the search function performs a rollout. A simplified pseudocode of the rollout function is available in the appendix.

The procedures executed by the rollout function are explained in the following sections. In order to understand how the search and the rollout functions work, we introduce some variables:

- *Global Policy*: $\text{levels} \times \{0, \dots, N-1\} \times \{0, \dots, N-1\} \rightarrow a_{\text{level}, i, j}$
- Global policy is a three-dimensional policy matrix that keeps a copy of the search policy at each level of the search tree.
- Policy vector for vehicle choice $\{0, \dots, m-1\} \rightarrow a_i$

Instead of a policy matrix for selecting vehicles, a policy vector of length m has been implemented. The reason behind choosing an one dimensional vector is that there is no direct correspondence between vehicles, unlike the case between nodes representing stops, where the distance between the nodes and the type of the stop contributes to the values of the policy matrix.

- N refers to the total number of nodes in the search graph. $N = 2(m+n)$.
- **Checked**: is a vector with the size $[0, \dots, 2(m+n)-1]$ is a vector designed to prevent having to do a costly backtrack operation in the case the inclusion of a node in a tour resulted in a new violation. The checked vector marks the nodes that has already been evaluated during search for candidate nodes to include in a vehicle's tour. This variable is reset each time a new vehicle is selected.
- **Visited** vector with the size $[0, \dots, 2(m+n)-1]$ is a Boolean vector that marks all the nodes that have already been visited by the solver and they are now part of the tour that represents the current solution.
- **Weight** vector according to the definition, each pickup node has a weight $w_i \geq 0$ for $i \in P$ and $w_i = -w_{i-1}$ for $i \in D$. The Successor set contains all the candidate nodes that can be selected to extend the search starting from the current node. The choice between these nodes will be based on the policy and will be explained more in detail in
- **Violations counter**. The algorithm will count the number of the times when any of the problem constraints is violated during the search process.

```

1 procedure search(level, iterations)
2   TOUR R
3   R.score = + Infinity
4   if level = 0 then
5     R = rollout()
6     Runs ++
7   else
8     policy[level] ← polGlobal // polGlobal denotes the global
    policy
9   for all i = {1, ..., iterations} do // for all iterations in current
    level
10    R ← search(level - 1, iterations)
11    if R.score < best.score then
12      best.score ← R.score
13      best.tour ← R.tour
14      adapt(best.tour, level)
15    end if
16    if (Time Limits exceeded or Max Number of Runs
    exceeded) then
17      return best
18    end if
19  end for
20  polGlobal ← policy[level]
21  end if
22  return best
23 end procedure

```

Algorithm 3 NRPA search function on minimization problem

C. Successors Selection

After the starting location of the vehicle, each following graph node in a vehicle's tour has to be selected according to the problem rules in a way that will guarantee a maximum number of customer's requests met in the most efficient way. Here we are going through the nodes selection process of the implemented algorithm.

In order for a node i to be considered as a valid move from current node n it has to be either

- a pickup node that has not been checked yet; or
- a delivery node for an order that has already been picked up.

Considering the specificity of the problem tackled helps identifying areas where performance can be enhanced. One of these improvements for the VRP in our case is to simply prioritise choosing a successor node to the current node by first looking for candidates among the pickup or delivery nodes in the same geographical position as the current node:

```

if (number of successors is 0)
  for all nodes  $i < N$ 
    check_valid_move(i)
    if node  $i$  location = currentnode location
      Add  $i$  to the possible moves

```

If there were no graph nodes found in the same geographical location as the current node, this would lead to

having an empty successors set. Therefore, the search for successors will be directed toward any valid possible move possible starting from the current node as follows.

```

if (number of successors is 0)
  for all nodes  $i < N$ 
    check_valid_move(i)
    Add  $i$  to the possible moves

```

If the successor set is still empty of no candidate nodes satisfying the valid moves check, this means that the search for nodes to extend the tour of the current tour has come to an end, and the next step would be choosing the next vehicle. However, if there are no more vehicles available, the number of violations increases with respect to the number of the nodes not included in the overall tour as follows:

```

violations += N - tourSize;
break SearchLoop;

```

Now assuming that the successor set is not empty, a choice between the one of the successors will be made according to the current policy. This inclined selection of a successor node is similar to the roulette wheel (sometimes called fitness selection) in genetic algorithms (more on this selection method can be found in the work of [11]).

```

for all successors do
  probability[i] = Math.exp(policy[node][moves[i]])
  sum += probability[i];
end for
mrand = random.nextDouble() * sum
i = 0
sum = probability [0]
while (sum < mrand)
  sum += probability [++i]
  node = moves[i]

```

If the successor is eventually determined, the tour will be extended by one city (either pickup or delivery location), and all violations are counted.

Adding the new node to the tour might lead to violation of one of the rules, therefore, a check will take place to determine the consequences of adding this node on the maximum vehicle's capacity or the order's time windows. Sometimes, extending the tour with a node might not immediately result in a violation. That is exactly the case when there are several orders located in the same geographical place needed to be handled together. This increases the service time, which is proportional to the number of these orders in the same place. In case that extending the tour with the selected node will result in a violation increment. The added node needs to be removed by directly eliminating it from the tour if it was a pickup node. Whereas, in the case of a delivery node, the node will be eliminated from the tour as well as its corresponding pickup node of the tour. Removing a delivery node requires looking up its pickup node in the tour $\bar{r} = (t_k \rightarrow v_1, v_2, \dots, v_h)$ which is

located in a position $x < h$ and removing it. Doing so means that the makespan of the tour has to be recalculated and the load of the current tour has to be adjusted. The distance between the v_{x-1} and v_k and between v_x and v_{x+1} will be substituted by the distance between v_{x-1} and v_{x+1} . The process is guaranteed not to increase the makespan of the tour providing that the triangle equality is assured as we first assume.

D. Rollout Policy

The policy matrix has size $N * N$ and consists of floating-point values. That matrix's values are updated by the adapt function whenever the search led to a better solution than the current best. The adapt function follows a procedure similar to the ones in [2], [5] and [4], as it adjusts the policy values representing the qualification to choosing each of the nodes based on the best-known solution so far. However, a noted difference is that the adapt function also performs the policy adaptation processes for the vehicle choice policy vector as well.

E. Vehicle Selection

Once one vehicle finishes its tour, the algorithm will try to complete the tours of the remaining vehicles in the fleet if there were any left. As the set of possible successors remaining vehicles is fixed, a decision based on the current vehicles policy vector is applied also using a choice based on roulette wheel selection just like the case with node selection. Once a new vehicle has been determined, the overall tour is extended by one stop (the new vehicle start location stop). The tour parameters will be reset (all violations are reset, makespan or the time elapsed so far, as well as the vehicle load will all be reset to zero). Also, all the checked nodes not visited can be set back to 'not checked'. Performing this action means that the nodes not integrated into the tours of the previous vehicle(s) can be reassessed for inclusion in the new vehicle tour. In case there were no more vehicles left in the fleet; so that the search terminates. Once the search terminates, any node that was not checked is counted as a violation. Also, the difference between the total number of nodes and size of the tour resulting from the search will be counted as violations.

IV. RESULTS

To validate the performance of the implemented algorithm, we discuss the results of running the solver on datasets obtained from industry. For this purpose, several industrial datasets representing complex VRP problems were obtained from XTL Kommunikationssysteme GmbH in Bremen.

A. Parameter configurations

A series of pre-tests has been conducted to determine the parameters that give the best results on different series of tests; i.e. the parameters that provide the balance between exploration and exploitation rates.

The approach minimizes the number of vehicles involved in the transportation process as its priority number one. Meeting the maximum number possible of demands comes next.

Due to the reasoning above, the penalty cost of adding a new vehicle was set to six times the penalty of missing an order. This difference helps in directing the algorithm towards minimizing the number of vehicles in the early stages of calculations first and then concentrating -during the remaining number of rollouts- on fitting the maximum number of orders in the tour of each vehicle.

The pre-tests did not consider changing the learning rate parameter of the policy used in the adapt function. Instead the value of the learning rate alpha that has been used in a previous publication using NRPA was adopted, see [5], [8] and [2].

The pre-tests led to the adoption of the following configuration:

- Number of the iterations in the search tree: 128
- Depth level of the search tree: 5
- Missing an order violation cost: $1 * 10^8$
- Violation of adding a vehicle: $6 * 10^8$
- Learning rate in the adapt function: 1

All experiments were executed on a PC Laptop equipped with a 5th generation Core i5 Intel® processor model 5200u with two cores running at 2.2 GHz and 16GB of RAM and Win 10 Professional OS.

B. Scenarios and results

In every scenario listed below, the results are obtained from running the algorithm with the exact same parameters and configurations settings.

Results presented of scenarios 2 to 5 are 20 randomized samples of multiple runs.

Scenario 1 Investigating policy matrix initialization

Number of orders: 18

Number of available vehicles: 2 identical vehicles.

After investigating in the pre-tests the best parameters to run the algorithm, this scenario evaluates the effects of policy matrix initialization on the quality of the solution.

The assumption is that giving the initial policy some "guidance" would result in a quicker convergence and an improved overall performance reflected by higher quality solutions.

To assess this assumption, the developed solver is given a scenario with two identical vehicles (so the effects of vehicle policy choice will be neutralized). However, two different policy initializations are applied each time. First, the policy will be initialized with zero values. Second, we apply a heuristic in every node's entry in the policy matrix. The applied heuristic gives small positive values to the closest corresponding nodes so that policy can guide the search, in the beginning, to look for solutions in the neighbourhood of these closer nodes.

The experiments were set to run 100 times for each initialization technique and results are presented in *Figure 2 Results obtained from initialization based on a closest distance heuristic.* and *Figure 3.*

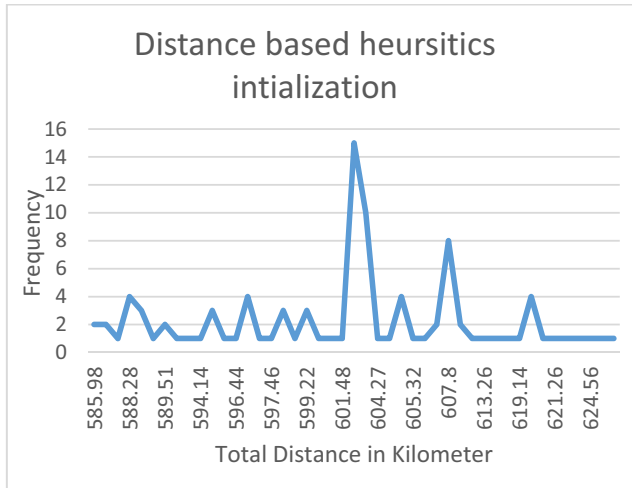


Figure 2 Results obtained from initialization based on a closest distance heuristic.

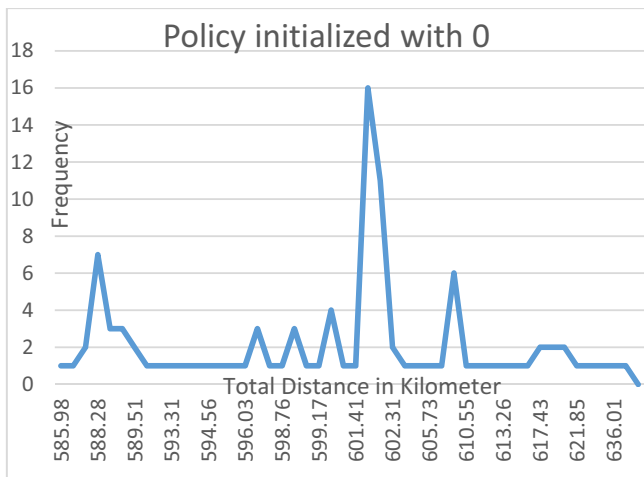


Figure 3 Results achieved with initializing policy with 0 values

Comparing the two graphs, one can see that they show similar overall behaviour in sense that many results are obtained around a total distance of 600 km. However, a major difference is the number of obtained solutions in the first half of the graph that stands for lower values of the distance covered. Here, more results were obtained when initializing policy with zeros in contrast to the other initialization.

The explanation is that looking in the neighbourhood of nearest next city may offer better results for small sized problems. However, it is in the interest of the search to find a solution in unexamined neighbourhoods of larger problems (increasing exploration rate), especially in highly constrained problems. Since a minimized overall distance covered in tours is preferred, the conclusion drawn is that starting the search with a policy initialized with 0 led to more occurrences of preferred results. Based on that, we adopted this initialization in all further runs of the solver.

In the coming scenarios 2 to 4 the number of available vehicles is 10 vehicles which differ in starting locations, end locations, operation times and capacities, but all have the same average speed.

Scenario 2

Number of orders: 13

Number of vehicles in the resulting solution: 2.

Scenario 2 utilizes two vehicles out of the available pool of ten. This scenario contains a relatively small number of orders. The samples taken show – see Figure 4 – that the algorithm could find a valid solution that satisfies all orders using two vehicles in 16 out of 20 cases. In the remaining four cases, not all orders were met, three of the remaining four cases produced results with two unserved orders and one solution with one unserved order. Those cases where all orders were met represent local optima in the neighbourhood of the search, whereas missing one order or two would result in a less overall cost -during evaluation- than adding a new vehicle.

Scenario 3

Number of orders: 18

Number of vehicles in the resulting solution: 2

A similar observation as in scenario two is witnessed here (Figure 5 Scenario 3). The solver was able to compute solutions using two vehicles out of the available ten. In the two local optima, one has 4 unserved orders, and the other missed 8. It can be noted here that the increased number of orders in the problem led to an increase in the number of unserved orders in the computed solutions.

Scenario 4

Number of orders: 22

Number of vehicles in the resulting solution: 2

With 22 orders to serve, the differences between solutions meeting all the demanded orders decrease as they all concentrate in a relatively small area(range) – see Figure 6 scenario 4 – regarding the total distance of the tour. In the same time, more orders to be transported, meant a decreased possibility to have all orders met with more occurrences of local optima missing between six (maximum) to two (minimum) orders.

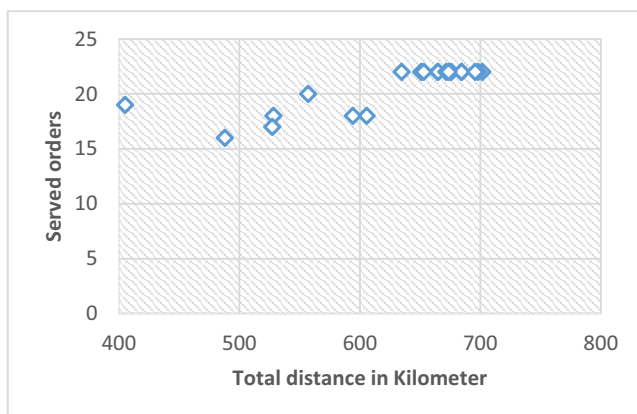
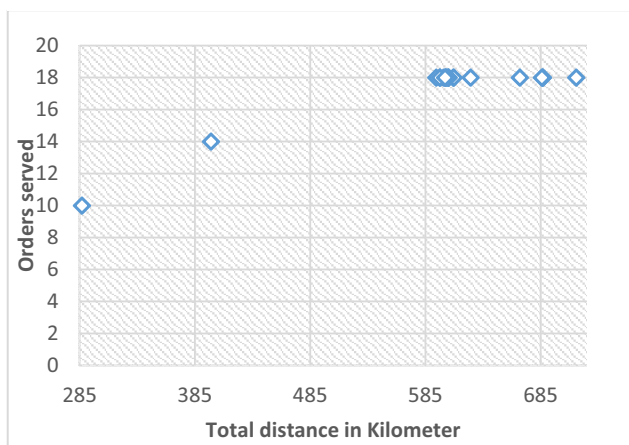
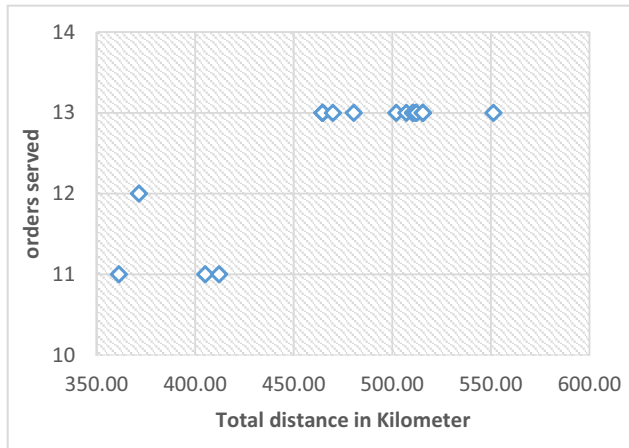
Scenario 5

Number of orders: 305

Number of available vehicles: 34 differ in starting locations, end locations, operation times and capacities, but all have the same average speed.

This scenario was run to test the ability of the developed algorithm to solve large problem sets. However, the implemented solver could not find any valid solution to the given problem within the number of allowed runs.

One direct criticism to the tests performed would be that they do not include tours with orders in the range between 50 to 200 orders for example. Orders in this range of sizes would be necessary to better understand the performance of the algorithm in midsized problems.



V. CONCLUSION

This research has outlined the usage of Monte Carlo Search and in particular, the NRPA to solve complex VRP. Test data from industry was used to tune the algorithm parameters, and then to evaluate the effect of policy initialization and finally to test the applicability of the proposed approach. Future work might consider benchmarking the performance of the developed approach against other “industrial” solvers, looking into the parallelization of the algorithm’s execution to compute better results, as well as considering dynamic problems where the distances between vertices might vary during runtime.

APPENDIX

```

procedure rollout()
    RESET: tour, tourSize, Visited, Checked, vehiclesUsed
    RESET: Makespan = currentload = violations = cost = 0
    RESET: validtourSize:
        Vehicle CurrentVehicle = ChooseVehicle()
    Node CurrentNode = StartOf (CurrentVehicle)
    Node PrevNode = CurrentNode
    Visited[CurrentNode] = true
    tour[0] = CurrentNode
    tourSize = 1
    TourLoop: while (true)
        Successors = Find_Successor_Candidates ()
        if (No Successors)
            if capacity constraints violated of (CurrentVehicle)
                violations ++
            tour[tourSize++] = EndOf(CurrentVehicle)
            if (CountRemainingVehicles > 0)
                CurrentVehicle = ChooseVehicle()
                CurrentNode = vehiclesStart[CurrentVehicle]
                visited[CurrentNode] = true
                checked[CurrentNode] = true
                tourlengthOfV[CurrentVehicle]++
                tourSize++
                Prev = vehiclesStart[CurrentVehicle]
                Makespan = EarliestVisitingTime[CurrentVehicle]
                for all checked nodes && not visited do
                    Checked[] = false
                RESET: violations, CurrentLoad
                // After picking up new vehicle
                // Proceed to the main search loop
                continue TourLoop
            else //No More vehicles available
                for all nodes not checked do
                    violations++ // all non checked nodes are
missed orders!
                    Violations += N - TourSize
                break TourLoop
            end if
            // selecting next node from candidates
            CurrentNode = Select_Next_Move(Successors)
            checked[CurrentNode] = true
            tourSize++
            define: oldviolations = violations

            if capacity constraints violated of (CurrentVehicle)
                violations ++
            end if
            if CurrentNode time window violated
                violations ++
            end if
            if CurrentVehicle window violated
                violations ++
            end if
            // Add the checked node to Tour
            // only if it does not violate
            if (oldviolations equals to violations)
                extend the tour with node
                prev = node
                visited[node] = true
                visitingVehicle[node] = CurrentVehicle
            else if pickup node
                checked[node + 1] = true
                violations ++
            end if
            else if delivery node
                tour = remove_correspondent_pickup(node, tour)
            end if
            end if
        end while
        NonUsedVehicles = CountRemainingVehicles()
return ( - Vehicle violation penalty * (NonUsedVehicles - V)
        + Order violation penalty * violations + cost)

```

REFERENCES

- [1] 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 Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [2] C. D. Rosin, "Nested rollout policy adaptation for Monte Carlo tree search," *IJCAI Int. Jt. Conf. Artif. Intell.*, no. line 22, pp. 649–654, 2011.
- [3] S. Edelkamp, M. Gath, C. Greulich, M. Humann, O. Herzog, and M. Lawo, "Monte-Carlo Tree Search for Logistics," Springer International Publishing, 2016, pp. 427–440.
- [4] S. Edelkamp, M. Gath, T. Cazenave, and F. Teytaud, "Algorithm and knowledge engineering for the TSPTW problem," *Proc. 2013 IEEE Symp. Comput. Intell. Sched. CISched 2013 - 2013 IEEE Symp. Ser. Comput. Intell. SSCI 2013*, pp. 44–51, 2013.
- [5] S. Edelkamp and M. Gath, "Solving Single Vehicle Pickup and Delivery Problems with Time Windows and Capacity Constraints using Nested Monte-Carlo Search," *5th Int. Conf. Agents Artif. Intell.*, pp. 324 – 329, 2013.
- [6] T. Cazenave, "Nested Monte-Carlo Search," *Int. Jt. Conf. Artif. Intell.*, pp. 456–461, 2009.
- [7] A. Rimmel, F. Teytaud, and T. Cazenave, "Optimization of the nested Monte-Carlo algorithm on the traveling salesman problem with time windows," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6625 LNCS, no. PART 2, pp. 501–510, 2011.
- [8] T. Cazenave and F. Teytaud, "Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7219 LNCS, pp. 42–54, 2012.
- [9] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2403–2435, 2007.
- [10] P. Toth and D. Vigo, *Vehicle Routing Problem, Methods, and Application*. Society for Industrial and Applied Mathematics, 2014.
- [11] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Phys. A Stat. Mech. its Appl.*, vol. 391, no. 6, pp. 2193–2196, 2012.
- [12] A. Klimova, E. Rondeau, K. Andersson, J. Porras, A. Rybin, and A. Zaslavsky, "An international Master's program in green ICT as a contribution to sustainable development," *J. Clean. Prod.*, 2016.