Simulation-based approach to Vehicle Routing Problem with Traffic Jams

Jacek Mańdziuk
Faculty of Mathematics and Information Science
Warsaw University of Technology, Warsaw, Poland
Email: j.mandziuk@mini.pw.edu.pl
and
School of Computer Science and Engineering
Nanyang Technological University, Singapore
Email: j.mandziuk@ntu.edu.sg

Maciej Świechowski
Systems Research Institute
Polish Academy of Sciences
Warsaw, Poland
Email: m.swiechowski@ibspan.waw.pl

Abstract—Capacitated Vehicle Routing Problem (CVRP) is a well-known NP-hard optimization problem. In this paper, we transform it into a non-deterministic dynamic version by introducing traffic jams (TJ).

The paper is the first attempt of applying the Upper Confidence Bounds applied to Trees (UCT) algorithm in the domain of dynamic transportation problems. In short, UCT is an extension to the Monte Carlo Tree Search (MCTS) method, however, unlike MCTS which makes use of uniformly distributed simulations, the UCT algorithm aims at maintaining an optimal balance between exploration and exploitation. The MCTS/UCT algorithm is enhanced by the usage of knowledge-based actions, which shares common traits with the human way of solving this task.

Our solution is compared with an Ant Colony Optimization method showing its upper-hand and raising hope for a cross-domain applicability of the proposed approach.

I. INTRODUCTION

Vehicle Routing Problem (VRP) [1] represents a rich family of problems concerning combinatorial optimization in transportation systems. They share a common concept of graph-based representation of a set of customers which are to be served by a fleet of homogenous vehicles. The goal is to minimize the total length of vehicles’ routes under certain conditions describing the way in which the customers need to be served. A description of a particular VRP version considered in this paper is presented in the next section.

VRP is an NP-hard problem [2] typically approached using meta-heuristic methods. Due to variety of VRP formulations used in practice, there exist multiple approximation algorithms for solving the problem, most of them designed to address specific real-life requirements or constraints, e.g., Savings algorithm [3], Multi-route improvement algorithm [4], Sweep algorithm [5], Ant Colony Optimization [6], Memetic Algorithm [7], [8], Granular Tabu Search [9] or Particle Swarm Optimization [10], [11], [12].

In this paper, we propose a new approach based on the Upper Confidence Bounds applied to Trees (UCT) algorithm [13], [14]. UCT is a machine-learning scheme that incorporates a tree of possible problem states (problem configurations), which are searched by means of simulations. Our approach has been inspired by wide UCT utilization in game domain, in particular in the General Game Playing framework [15], [16], [17], [18], [19], or in games with high degree of dynamism (like Havannah [20], [21]), or the ones for which a compact and reliable evaluation function in not known (e.g., Go [22], [14]).

In all the above-mentioned game domains, the UCT method defines de facto a state-of-the-art approach. Although the basic MCTS/UCT approach can be classified as a “knowledge-free” method [23], [24], this work investigates various possibilities of injecting domain knowledge into the system so as to make the obtained solutions more robust, while preserving the “learning by simulation” nature of the method.

The approach presented in this study shares common traits with the human way of solving tasks. Using the set of sensible actions that modify routes and the idea of checking possible scenarios coming out from taking these actions are very close to what humans actually do when solving dynamic versions of VRP in practice. Certainly, humans do not require examining such a high number of possible scenarios (i.e. do so many simulations) as UCT does, as they can efficiently rely on their cognitive skills, in particular experience-based knowledge transfer between solved problem instances. Apparently, this human-centric cognitive ability can be, in some sense, compensated by large enough number of tested scenarios, leading to high quality results.

The remainder of the paper is organized as follows: in the next section, the particular variant of VRP (called CVRPwTJ) considered in this piece of research is introduced. Section III provides a brief description of the baseline MCTS/UCT method, whereas Section IV is devoted to application of UCT to solving CVRPwTJ, which is the main contribution of this paper. Section V presents the experimental setup, empirical results and their comparison with the Ant Colony Optimization (ACO) algorithm. The final section summarizes the main outcomes and concludes the paper.

II. PROBLEM FORMULATION

VRP is a generalization of the Traveling Salesman Problem. The problem can be modeled using an undirected graph
The graph $G = (V,E)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the vertex set and $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$ is the edge set. In VRP, a set of $n$ customers represented by vertices $\{v_1, v_2, \ldots, v_n\}$, a specified depot $v_0$, and a fleet of $m$ vehicles are considered. Each edge $e_{ij} = (v_i, v_j)$, $i, j = 0, \ldots, n$ has an associated weight $c_{ij}$ which represents the cost (a distance) between $v_i$ and $v_j$ (being either two customers or a customer and a depot). Furthermore, for each customer $v_i$ the demand $d_i$, which needs to be serviced by exactly one vehicle, is defined. The time in the problem is discrete and the speed of each vehicle is defined as one distance unit per one time unit.

The goal is to minimize the total routes’ length of all vehicles according to the following constraints: (*) each vehicle has to start from a depot and end its route in a depot, and (***) every customer has to be served exactly once and by one vehicle.

If vehicles are homogenous, each with identical capacity $c$ and it is additionally required that (***) the sum of customers’ demands assigned to each vehicle must not exceed vehicle’s capacity $c$, then VRP is extended to Capacitated VRP (CVRP).

Capacitated Vehicle Routing problem with Traffic Jams (CVRPwTJ) further extends the CVRP by introducing the Traffic Jams (TJ). TJ may occur at any edge $e_{ij}$ with certain probability $P$ at the beginning of each (discrete) time step. If it does, the regular cost $c_{ij}$ of traversing the given edge is multiplied by the intensity factor $I(e_{ij})$ (sampled from a certain probability distribution) for a randomly selected number of steps $L(e_{ij})$. If a TJ happens to occur on an already jammed edge, then only TJ length is increased by a newly sampled length $L(e_{ij})$, whereas the intensity $I(e_{ij})$ remains unchanged (i.e. the one assigned previously is used). This way an exponential growth of TJ intensity is avoided which might have otherwise appeared in the case of repetitive TJs occurrence on a certain edge. The details of experimental setup are discussed in Section V-A. It is worth underlying that the solution methods tested in this paper are aware beforehand only about the TJ probability distributions and do not have access to the actual realizations of TJ, unless their materialization in a certain time step.

Introduction of the varying traversal costs carry a few consequences. Firstly, the triangle inequality does no longer hold with regards to the customers’ positions which may render geometry-based methods useless. Secondly, highly dynamic changes may require immediate reactions by means of remodeling the current solution constructed during the main simulation. To address this issue, we propose the MCTS/UCT algorithm equipped with a set of actions aimed at reactive local optimizations (see Section IV-C).

More precisely, the solution, i.e. a set of routes, is constructed step by step by means of simulations. We will reserve the term main simulation for the dynamic construction of the solution, in order to avoid confusion with UCT internal, off-line simulations aimed at testing various actions/modifications without applying them to the actual solution. Apart from the impact (the actual solution vs. various tested scenarios), the main simulation and off-line simulations have the same structure. They are divided into discrete steps. In each step, the current traffic situation is calculated based on assumed TJ distributions and the resulting TJ are imposed. Next, at least one vehicle must be assigned a non-empty visiting schedule. We will call such vehicles active. Active vehicles are moving simultaneously one step ahead according to their plans. The total cost of the simulation is incremented by the sum of edges costs traversed by the vehicles. Unless the problem is solved, i.e. all vehicles have returned to the depot after serving all clients and fulfilling the constraints (*)-(***), the simulation proceeds to the next step.

### III. Monte Carlo Tree Search and Upper Confidence Bounds Applied to Trees

UCT is an action-selection algorithm in Monte Carlo Tree Search aimed at maintaining a balance between exploration and exploitation. As we have already stated in the introduction, UCT is the main routine of many world-top game-playing agents. Recently, the algorithm also gained attention in the area of probabilistic planning implemented by the Markov Decision Process (MDP) model [25], [26], [27].

The underlying UCT idea is to perform multiple simulations in order to verify various lines of actions (scenarios) and estimate their potential payoffs. The problem is represented in the form of a tree whose nodes and edges represent problem states and possible actions in that states, respectively. The root node represents the current state, its children the possible subsequent states, etc. The tree is iteratively searched (by simulating sequences of performed actions) and gradually expanded, usually one node per simulation due to memory limitations. Each iteration consists of four phases depicted in Figure 1. The maximum number of iterations per decision step is either defined a priori or limited by the time allotted for performing a single action. In the first phase – Selection –
reached again in a search process, choose move $a^*$ according
to the following formula (1):

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln (N(s))}{N(s, a)}} \right\} \quad (1)$$

where $A(s)$ is a set of all actions available in state $s$, $Q(s, a)$
denotes the average result of playing action $a$ in state $s$ in
the simulations performed so far, $N(s)$ is a number of times
state $s$ has been visited and $N(s, a)$ - a number of times
action $a$ has been sampled in this state. Constant $C$ controls
the balance between exploration and exploitation, since the
formula postulates choosing actions with the highest expected
rewards and, at the same time, avoiding repetitive sampling of
the same actions while others might yet prove more beneficial.

In the second phase – Expansion – the first state expanded
outside the in-memory part of the UCT tree is added to the
tree. Next, in the Simulation phase a uniform random playout
(also called rollout) is performed until the terminal state is
reached. Finally, in the last phase, called Backpropagation,
the statistics related to the obtained result of the performed
simulation (problem outcome read out in the terminal state)
are used to update the internal statistics of all in-memory tree
nodes visited in the current iteration up to the root node. This
way, the algorithm performs partly guided random sampling of
the state-space. In our approach, in the Simulation phase the
UCT algorithm may choose among a certain set of specifically-
designed available actions (see Section IV-C).

With the UCT simulations finished in a given time step,
choosing the next move is simply a matter of finding the action
with the highest $Q(s, a)$ value in the current (root) state.

IV. APPLICATION OF UCT IN CVRPwTJ

Our main task consists in adaptation of the UCT method
to solving CVRPwTJ. To this end we need to address the
following issues:

- Decide whether the algorithm should construct the solution
  from scratch or start from the initial solution. We
  went for the latter option. Since at time zero there are
  no TJ imposed yet, we start our algorithm by finding
  an initial solution to the static CVRP problem instance
  with the help of a modified Clark and Wright [3] savings
  algorithm [28]. In this initial solution, each route commences
  and ends in a depot and the routes are pairwise separated
  (except for the initial and final position which is always a
  depot). This set of optimized paths, with depot being the
  first and the last element, forms the input to our UCT-
based approach.

- Define the way in which partial problem solutions represen-
ted by individual trucks’ routes will be combined into
  one, coherent and effective global solution. This issue is
  discussed in section IV-A.

- Define what constitutes the problem state in CVRPwTJ
  and what kind of information needs to be stored in the
  UCT tree nodes (c.f. section IV-B).

- Design the set of possible actions in the way that will
  allow to avoid combinatorial explosion of possible next
  states in the case where actions performed for each
  vehicle during simulations are taken jointly as a single
  state update (see section IV-C).

A. Application of UCT to the problem

Suppose the initial solution is composed of $k$ routes, i.e.
uses $k$ trucks. Then, as stated above, the initial UCT tree is
actually a forest composed of $k$ trees - each in the form of a
path with the first and the last elements being a depot. The
consecutive elements on each path denote the clients visited
by respective trucks in subsequent time steps. For example,
the fourth elements in all paths represent the set of clients
visited in the third step of the solution.

The internal UCT simulations are performed at each time
step from the root nodes simultaneously in all $k$ trees. As
explained in Section III the trees are gradually extended in
a typical UCT fashion (one leaf node at each simulation),
however, there are three main differences compared to the
baseline UCT implementation.

First of all, since the shorter the solution the better, the UCT
formula (1) is modified to the following version (2), which
favors the shorter average outcomes $Q(s, a)$:

$$a^* = \arg \max_{a \in A(s)} \left\{ C \sqrt{\frac{\ln (N(s))}{N(s, a)}} - Q(s, a) \right\} \quad (2)$$

Second of all, the next compound step (movement of all $k$
trucks) is a result of a combined knowledge obtained from all $k$
trees. More precisely, in each tree the most promising action is
selected, then these $k$ selected actions are sorted in descending
order based on their UCT values (i.e. values $C \sqrt{\frac{\ln (N(s))}{N(s, a)}} - Q(s, a)$ in (2)) and afterwards executed in this order.

The third difference compared to classical UCT implement-
ization is that simulation ends when there are no more
active trees (all trucks have completed their routes) or the step
counter reaches a pre-defined threshold value of MAX_STEPS
(the so-called Early Termination), whichever comes first. We
set MAX_STEPS to be equal to the maximum length of
a traffic jam. There are few reasons behind using Early
Termination over the “natural” finishing at terminal state only.
Firstly, simulations are much faster and, therefore, more of
them can be performed. Secondly, simulation is focused more
on the current traffic situation. Thirdly, memory usage is
dramatically decreased. We tested both a natural termination
and Early Termination and the latter leads to the comparable
or slightly improved results in approximately one order of
magnitude shorter computational time.

A dynamic (affected by the traffic) cost of each action
chosen in a simulation increases the cumulative score of a
simulation. When the Early Termination condition is applied,
the sum of static costs (i.e. without TJ consideration) computed
for the remaining fragments of the routes is added to the score.
Once the simulation is completed, such a compound result
from all $k$ trees is back-propagated from the last visited nodes
in each tree to the root nodes.
After a certain number of internal simulations, the actual (real) decision regarding the movement of $k$ trucks is made according to the smallest $Q(s, a)$ value among the child nodes in each of the $k$ trees. These $Q(s, a)$ values are sorted in an ascending order (i.e. first the action in the tree with the lowest $Q(s, a)$) is executed, then the action in the tree with the second-lowest $Q(s, a)$, etc.). Please note, that execution of an action in one tree may disable some further actions (in subsequent trees). In that case the next best action in the latter tree is selected instead.

B. State Representation

Note that different sequences of visited customers in a route may lead to the same state, e.g., $[0, 11, 9, 4, \ldots, 0]$ and $[0, 9, 11, 4, \ldots, 0]$ after the first two time steps. In our implementation, we detect isomorphic states (called transpositions) and use only one node in the tree per each transposition. In that case the tree is transformed to a Directed Acyclic Graph (DAG)\(^1\). Implementation of the concept of state transpositions requires precise conditions for the state equality to be defined. Two states are mapped to one state (in the UCT tree) iff they share the following data: (1) the current position of a vehicle in a route, (2) the remaining capacity of a vehicle which can be allocated and (3) the remaining set of customers scheduled for a route.

If two states differ by at least one of the above-listed properties, they are represented by different nodes. Hence a node can be properly interpreted only in the context of the respective route (route-states representation). The complete UCT state is defined by a $k$-tuple of route-states - one per each route.

C. Possible Actions in the UCT Trees

At each step of the UCT simulations as well as real decisions regarding the vehicles’ tours, all possible actions are considered in each of the $k$ root nodes. There are three types of actions differing by their complexity: level-0, level-1 and level-2, which modify 0, 1 and 2 existing routes, respectively. In each case, there are some pre-conditions which define legality of an action. Otherwise (if the action is illegal) it is not considered in a given state. Level-0 and level-1 actions used in our approach are listed in Table I.

All these actions are simple and self-explained. Apart from specific legality conditions, there are also “natural” legality conditions related to the number of customers left in a route. For instance, each action apart from $A0$ and $A1$ requires existence of at least two non-depot customers - the current planned one to visit and at least one candidate for the replacement of the current one in the tour order. Action $A7$ requires a route to have at least three customers.

Except for the above-mentioned 9 actions there are also 4 more complex, level-2 ones, numbered from $A9$ to $A12$. These actions operate on two routes and, in principle, all pairs of routes are considered. Consequently, there can be many realizations of a particular action in one time step, depending on the number of route pairs that fulfill legality conditions. In the following description the two currently iterated routes are denoted by $r_i$ and $r_j$, where $i \neq j$.

In the case of action $A9$, the current route ($r_j$) is finished and all its customers are appended to $r_i$. This action is similar to $A8$, except that in this case the customers are inserted into the existing route instead of forming a new one. The following legality conditions must be fulfilled: (1) from the current vehicle location there are TJ to all remaining customers planned in $r_i$; (2) the edge from the vehicle location in $r_i$ to the depot is not jammed; (3) $r_j$ has enough capacity left to accommodate demands of all customers in $r_i$ (planned to be transferred to $r_j$).

Not going into details, actions $A10$ and $A11$ exchange customers between two routes based on a common exchange scheme, differing by the range of customers replacement. In $A10$, the replacement is the smallest possible (the exchange is finished with the first non-jammed situation). In $A11$, the exchange is more complex and aims at exchanging the biggest possible parts of the routes (to some extent similarly to the crossover operation in Genetic Algorithms with the crossover point fixed on the current locations in the respective routes). In both cases the following prerequisites must be fulfilled: (1) one of the routes begins with a TJ; (2) after exchange none of the routes begins with TJ; (3) the exchange does not violate capacity constraint in any of the vehicles.

The last action - $A12$ - is a two-route extension of $A8$. It finishes two routes $r_i$ and $r_j$ and starts a new one $r_k$. The action is implemented in four variants depending on the order in which customers are inserted into $r_k$. Therefore, even for the same pair of routes, there can be up to four instances of the $A12$ action: (1) $r_k := r_i + r_j$; (2) $r_k := r_j + r_i$; (3) $r_k := Reverse(r_i) + r_j$; (4) $r_k := Reverse(r_j) + r_i$. The + sign denotes the concatenation of routes, which omits the depot from the first one, e.g., $[4, 5, 1, 0] + [3, 2, 0] = [4, 5, 1, 3, 2, 0]$.

All proposed actions are based on the following underlying rationale: if the currently selected candidate edge is not jammed then traverse it, otherwise try to enhance the planned route (by avoiding the traffic jam) by means of local changes in the planned orders of visited clients.

In the case of actions $A0$-$A7$ the in-route optimization takes place. Actions $A6$ and $A7$ are the only ones which allow to optimize a route which is not jammed. They may be particularly suitable when the route is far from optimal due to some changes forced in earlier steps. Since these actions implement greedy local improvement scheme, in order to prevent their too frequent usage, there is a penalty coefficient assigned to them which increases the estimated score value $Q$ in the UCT equation (2). Actions $A8$ and $A12$ immediately complete the current routes. Since there is neither TJ on the edge leading to the depot nor on the one from the depot to the next planned customer, the next turn will potentially be made on non-jammed edge starting from the depot. The remaining three actions: $A9$, $A10$ and $A11$ exchange customers between the two routes so as to reach locally (temporarily) non-jammed

\(^1\)For the sake of simplicity and due to a commonly used collocation the above-described DAG will be referred to as UCT tree.
solution.

In theory, one might proceed with defining even more complex actions, e.g., the ones involving three or more routes, but such approach immediately becomes infeasible due to computational complexity explosion.

V. EXPERIMENTAL RESULTS

In this section, the experimental setup and TJ parametrization are presented together with the results of applying the proposed approach to a set of popular static CVRP benchmarks modified by imposing the TJ. The results are compared with the realization of static solution (not reacting to TJ occurrence) and ACO algorithm designed for solving CVRPwTJ.

A. Experimental Setup

The experiments were performed for a set of static benchmark problems specified in Table II enhanced by adding dynamic TJ. The initial conditions (i.e. the number of available trucks, their capacity, clients requests’ sizes and the coordinates of the depot and the customers) are part of the benchmark sets. At each time step, for each edge (a direct link between two clients or a client and a depot) the TJ was imposed with probability $P$. If TJ happened to appear on a given edge $a$ it was assigned a randomly selected intensity $I(a)$ and length $L(a)$ (measured in time steps). The above TJ assignment was applied at the beginning of each time step, for all edges.

The benchmarks were downloaded from the CVRP webpage [29] and modified into CVRPwTJ according to the following TJ uniform probability distributions:

\[ P \in \{0.02; 0.05; 0.15\}, \]
\[ I = U_{INT}[10; 20], \]
\[ L = U_{INT}[2; 5], \]

where $U_{INT}[a, b]$ denotes random uniform selection of any integer $x$, such that $a \leq x \leq b$. Based on the initial calibration tests, the value of $C$ in (2) was set to 1.8 multiplied by the length of the initial solution found for the static instance.

B. ACO Approach to CVRPwTJ

Our implementation of the ACO approach is based on a standard algorithm used to solve the Traveling Salesman Problem (TSP). The two main differences (compared to TSP) are that in CVRPwTJ a set of $k$ cycles (one per each vehicle) needs to be found instead of one Hamiltonian cycle and that the problem is dynamic. Between every two consecutive updates of the main simulation state, the ACO system is executed to find the current best solution and based on that (and the new traffic situation), a new state is defined. The state is defined as a list of current trucks’ positions, a collection of current traffic jams and a list of unvisited customers.

Main algorithm loop

A pseudocode of the main loop is presented in Algorithm 1. During the decision time, the ACO system runs $MAX_I$ iterations with a population of $MAX_A$ ants. The number of ants was set to $max(100, 2n)$, $MAX_I$ was set to 200 for benchmarks with $n < 69$, to 100 for $n = 69$, and to 75 for benchmarks with $n > 69$. The above parameters were limited by the assumed time allotted for reaching the solution, which was, anyway, greater than that required by the UCT-based approach. When comparing real-time usage of the methods, the ACO system required nearly twice as much time than the UCT approach to complete all benchmarks.

In each iteration, each ant starts with the current state and

<table>
<thead>
<tr>
<th>Instance</th>
<th>$n$</th>
<th>$m$</th>
<th>$c$</th>
<th>$BEST$</th>
<th>#Sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-n19-k2</td>
<td>19</td>
<td>2</td>
<td>164</td>
<td>212</td>
<td>30 000</td>
</tr>
<tr>
<td>P-n43-k5</td>
<td>45</td>
<td>5</td>
<td>150</td>
<td>510</td>
<td>30 000</td>
</tr>
<tr>
<td>E-n51-k5</td>
<td>51</td>
<td>5</td>
<td>160</td>
<td>521</td>
<td>30 000</td>
</tr>
<tr>
<td>A-n54-k7</td>
<td>54</td>
<td>7</td>
<td>100</td>
<td>1167</td>
<td>30 000</td>
</tr>
<tr>
<td>A-n69-k9</td>
<td>69</td>
<td>9</td>
<td>100</td>
<td>1168</td>
<td>30 000</td>
</tr>
<tr>
<td>E-n76-k5</td>
<td>76</td>
<td>7</td>
<td>220</td>
<td>682</td>
<td>30 000</td>
</tr>
<tr>
<td>A-n80-k10</td>
<td>80</td>
<td>10</td>
<td>100</td>
<td>1764</td>
<td>30 000</td>
</tr>
<tr>
<td>P-n101-k4</td>
<td>101</td>
<td>4</td>
<td>400</td>
<td>681</td>
<td>30 000</td>
</tr>
<tr>
<td>C-n150D-k5</td>
<td>150</td>
<td>12</td>
<td>200</td>
<td>1097</td>
<td>30 000</td>
</tr>
<tr>
<td>Tai-n150B-k5</td>
<td>150</td>
<td>14</td>
<td>1918</td>
<td>2862</td>
<td>30 000</td>
</tr>
</tbody>
</table>
Algorithm 1 The external loop of the ACO approach run at the beginning of each time step. A pseudocode of the AntIteration function is presented in Algorithm 2.

```plaintext
bestSolution = NULL
for iteration = 1 to MAX_I do
    for ant = 1 to MAX_A do
        SetTruckPositions() // to real positions
        SetTraffic() // to real traffic
        AntIteration(ant)
        solution = ant.GetSolution()
        if solution better than bestSolution then
            bestSolution = solution
        end if
        UpdatePheromone()
    end for
    return: bestSolution
```

Algorithm 2 The procedure of choosing the most feasible client for a route continuation, executed for each ant during a single AntIteration. The selection of a customer is based on eq. (3) presented below.

```plaintext
while UnvisitedCities.Count > 0 do
    for each active route: R do
        viableCustomers[] = Customers WHERE Demand <= R.CapacityLeft
        if ((there exists a non-jammed transit from R.LastCustomer to any customer in viableCustomers) OR (new truck is not available)) then
            PseudoRoulette(viableCustomers[])
        else
            FinishRoute(R)
            StartNewActiveRoute()
        end if
        AdvanceMovement() // move trucks
        RemoveFinishedRoutes()
    end for
end while
```

finds a complete remaining solution to the problem, i.e. a set of $k$ routes for the trucks. When the solution is found, its quality is evaluated and the pheromone is deposited in a way explained in detail in the following subsections. In addition, the overall best solution found so-far, by any ant, is stored in memory. When there is time to make a real decision (in the main simulation) the next positions of the trucks are read out from this best solution.

Each of the $MAX_I$ external iterations depicted in Alg. 1 consists of repetitive calls of AntIteration procedure, presented in Algorithm 2. For each route, the procedure starts from the current vehicle’s position and finds the next customer to be added to the route based on pheromone trails and the current dynamic cost of traversing particular edges. If there are TJ to all customers from a particular position or the left truck’s capacity is not sufficient to serve any of the remaining customers then the truck is set to return to the depot and a new route is started.

Pseudo-roulette selection

The minimum and the maximum possible pheromone deposits, $f_{min}$ and $f_{max}$, are set to 2 and 50, respectively. The initial solution computed by the CW algorithm [28] is used to set the starting number of routes ($k$) and deposit the initial pheromone. Edges, which belong to the initial solution are initialized with the pheromone value equal to $5 * f_{min}$, whereas those which are not part of the initial solution receive the minimum amount of pheromone $f_{min}$. The pheromone value is confined to the interval $[f_{min}, f_{max}]$ by assigning it the respective boundary value whenever it falls outside this interval. This restriction scheme has proven to be beneficial during preliminary experiments.

The customer selection procedure chooses the next client to be visited using the pseudo-roulette. Let $v_i$ denotes the current customer, $v_j$ - the potential unvisited customer, and $d_{ij}$ - the current dynamic (traffic-aware) cost of moving between $v_i$ and $v_j$. With probability 0.05 the closest (by means of dynamic cost) unvisited customer is selected as the next one to be visited and with probability 0.95 the next customer is chosen according to the roulette-wheel selection mechanism with probabilities calculated as follows:

$$p_{ij} = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{i,j}(\tau_{ij}^\alpha \cdot \eta_{ij}^\beta)} \quad \eta_{ij} = \left(\frac{BASE}{d_{ij}}\right)^2$$

where $\tau_{ij}$ denotes the pheromone amount deposited on the edge $e_{ij}$ and $BASE$ is the length of the initial (static) solution. The remaining two parameters: $\alpha = 2$ and $\beta = 3$ were set based on some number of preliminary tests.

Pheromone update

Once ant $a$ finds a solution (completes one iteration of the algorithm), its quality is computed according to the following equation:

$$Q_a = (BASE/D)^2$$

where $D$ denotes the total dynamic cost of the found solution.

After all ants complete the current iteration, the pheromone increment is computed for each edge $e_{ij}$ in the following way:

$$\Delta \tau_{ij} = \sum_{ants} (\delta_{ij} Q_a)$$

where $\delta_{ij}$ can take one of the three values: 0: if $e_{ij}$ was not part of the solution found by ant $a$; 10: if $e_{ij}$ was part of the solution found by ant $a$, but the solution was not the globally best one; 20: if $e_{ij}$ was part of the solution found by ant $a$ which was the overall best solution (elitist approach).

The pheromone update procedure also takes into account pheromone evaporation, namely 90% of the pheromone deposited so-far is evaporated. Equation (6) presents the final formula defining the amount of pheromone deposited at the beginning of the next iteration.

$$\tau_{ij} := Conf(0.1 \cdot \tau_{ij} + \Delta \tau_{ij})$$

where $Conf$ function confines pheromone values to $[f_{min}, f_{max}]$ interval in the way described above.

When the last iteration is completed by all ants, the best solution is found. It is used only locally to move the trucks according to the schedule contained in the solution.
natural text
seems to be a truly interesting observation is the fact that all 8 winning cases of ACO happened for the 3 data sets only ($E51$ and $E71$ - three wins and $A80$ - two wins). Further investigation of the peculiarities of these three sets in an attempt to discover the reasons for their particular unsuitability for the UCT-based approach is one of our current research goals.

VI. CONCLUSIONS

In this paper, a new method of solving the Capacitated Vehicle Routing Problem with Traffic Jams is presented. The method has been inspired by the MCTS/UCT algorithm successfully applied in General Game Playing and several other games. We proposed a way of adapting this approach to CVRPwTJ - an example of NP-hard dynamic combinatorial optimization problem. To this end we introduced new problem representation in the form of a forest of UCT trees, in which each tree is associated with one route. The number and range of actions which are performed in these trees are restricted in order to avoid combinatorial explosion of possible simulated scenarios.

The results of UCT-based approach were evaluated against results of a carefully tuned ACO algorithm and the static solution (not reacting to traffic jams). Our approach produces convincingly better solutions to the benchmark problems in 22 out of 30 cases. Moreover, it is the most stable method, i.e., having the lowest standard deviation, among the tested algorithms.

Promising results and the inherent generality of proposed approach, which stems from the MCTS/UCT universality, give hope for further successful development of the method and its application to solving other types of stochastic transportation problems and beyond that area.

ACKNOWLEDGMENTS

The research was financed by the National Science Centre in Poland grant number DEC-2012/07/B/ST6/01527.

REFERENCES