Raport Badawczy
Research Report

RB/23/2016

# UCT in Capacitated Vehicle Routing Problem with traffic jams

J. Mańdziuk, M. Świechowski

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences

**POLSKA AKADEMIA NAUK**

**Instytut Badań Systemowych**

ul. Newelska 6

01-447 Warszawa

tel.: (+48) (22) 3810100

fax: (+48) (22) 3810105

Kierownik Zakładu zgłaszający pracę:
Prof. dr hab. inż. Janusz Kacprzyk

Warszawa 2016

# UCT in Capacitated Vehicle Routing Problem with Traffic Jams

Jacek Mańdziuk[a,b,*], Maciej Świechowski[c]

[a]*Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland*
[b]*School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[c]*Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland*

## Abstract

In this paper a dynamic version of the Capacitated Vehicle Routing Problem (CVRP) which takes into account traffic jams is considered. Traffic jams occur randomly according to pre-defined intensity and length distributions. In effect, static CVRP is transformed into a non-deterministic scheduling problem with high uncertainty factor and changing in time internal problem parameters. Our proposed solution to CVRP with traffic jams (CVRPwTJ) relies on application of the Upper Confidence Bounds applied to Trees (UCT) method, which is an extension of the Monte Carlo Tree Search algorithm. The most challenging issue here is finding a suitable mapping of the CVRPwTJ onto a tree-like problem representation required by the UCT. Furthermore, in order to prevent the size of the tree from explosive growth, an efficient mechanism for child nodes selection is proposed. UCT-based approach is compared with four other methods showing promising results and offering prospects for its wider applicability in the domain of stochastic optimization problems.

*Keywords:* stochastic optimization, Vehicle Routing Problem, UCT, optimization under uncertainty

*Corresponding author
*Email addresses:* j.mandziuk@mini.pw.edu.pl (Jacek Mańdziuk), m.swiechowski@ibspan.waw.pl (Maciej Świechowski)

## 1. Introduction

Vehicle Routing Problem (VRP) [9], along with its numerous variants, is a widely known combinatorial optimization task. The problem was formulated in 1959 [9] and subsequently proved to be NP-hard in 1981 [25]. In short, the problem consists in assigning a number of homogeneous vehicles to a number of clients, where each client has a certain 2D location and a certain demand of (homogeneous) goods. The optimization objective is to deliver the demanded goods to all clients while minimizing the sum of vehicles routes' costs (lengths). Additionally, each client must be served by exactly one vehicle and each vehicle's route must start and end in the depot (defined by its 2D coordinates). For practical reasons, the upper limit on vehicles' capacity is often imposed, leading to the Capacitated Vehicle Routing Problem (CVRP) formulation.

Since VRP/CVRP is NP-hard, no polynomial method of solving the problem is known and perfect solutions can only be obtained for relatively small-size problems. Among the exact algorithms one can distinguish the following three main approaches: full tree search (e.g. spanning tree and shortest path relaxations method [6]), dynamic programming (e.g. [13] in the case of problems with a priori known number of required vehicles) and integer programming (e.g. three-index vehicle flow formulation [16]).

There are also multiple approximation algorithms for VRP/CVRP, most of them designed to address specific problem formulations, e.g. multi-trip [4] or multi-compartment [1] versions of the problem, variants with certain delivery time-windows [18] or dynamically defined requests [28], combined pick up and delivery problem formulations [29], ecology-oriented Green VRP [42], and many others. Due to space limits, we are not able to provide a more in-depth characteristics of these and other related works. Please see the recent special issue [40] for an overview of the current developments and challenges in this domain and [14] for the VRP taxonomy.

A particular variant of CVRP considered in this paper, which we call CVRP with Traffic Jams (CVRPwTJ), introduces a high degree of uncertainty to the problem specification by means of traffic jams (TJ), which may dynamically occur on particular edges of the planned vehicles' routes. The existence of a traffic jam increases the cost of traversing a certain edge, usually to the extent that requires some re-modeling of the currently planned route. In the proposed solution, these dynamic changes are handled on-line by appropriate actions taken to alleviate their impact.

Our proposed solution to CVRPwTJ relies on the Upper Confidence Bounds Applied to Trees (UCT) method [22, 3], which is currently a state-of-the-art approach in game playing domain, particularly applicable to games for which compact and easily computable position assessment function is not known. Typical examples of such games/game frameworks are Go [17, 3], General Game Playing [35, 36, 41] or Havannah [39].

The main advantage of using UCT in games is its adaptability to the changing game situation and long-term reliability of position assessment. An additional asset of UCT is its "knowledge-free" nature [27] understood as the lack of the requirement for providing any domain-specific knowledge, except for the formal game definition, which is indispensable in the move generation process and for detection and evaluation of the final states of the game.

In the view of the above-listed UCT qualities, we conducted research on possible ways of applying the modified form of this algorithm to solving CVRPwTJ. In particular, presented research aims at verification of the UCT capability to flexibly address the *exploration vs. exploitation dilemma* in CVRPwTJ, i.e. the issue of balancing the usage of discovered best solutions vs. finding the new ones with respect to highly variable problem parametrization (stochastic changes of TJ intensities). Such a *plasticity* of the solution method seems to be indispensable for efficient solving stochastic optimization problems, in particular CVRPwTJ.

Since, to the best of our knowledge, this paper presents the first approach to solving the CVRPwTJ by means of the UCT method, we had to make several decisions related to particular implementation and usage of the method in a new application domain. The main issue was related to the CVRPwTJ problem representation in the form of a graph (which is a desirable representation for the UCT tree-search method), and definition of the set of UCT actions (possible "moves" in a given state of a partial solution) as well as their interpretation per analogy to game moves.

The efficacy of the proposed approach is compared with two versions of Genetic Algorithms (GA) implementations and with Tabu Search (TS) and Ant Colony Optimization (ACO) metaheuristics showing its upper-hand under the same time and resource availability. Several differences in which UCT and the above-mentioned metaheuristic methods tackle the problem have been pointed and discussed.

The rest of the paper is organized as follows: in the next section a formal definition of the CVRPwTJ is provided. Sections 3 and 4 summarize the UCT method and the way we propose to apply it to solving CVRPwTJ, respectively. In section 5, evolutionary methods used for comparison with the proposed UCT approach are introduced. Section 6 is devoted to presentation of an experimental setup, simulation results and their comparison with the above-mentioned metaheuristic approaches. A summary of the main contribution and directions for future research conclude the paper.

## 2. Capacitated Vehicle Routing Problem with Traffic Jams

In CVRP, a fleet $V = \{v_1, \ldots, v_n\}$ of $n$ vehicles is to deliver cargo to a set of $m$ clients $C = \{c_1, \ldots, c_m\}$ each of them having a demand of size $size_i, i = 1, \ldots, m$. Vehicles are homogeneous, i.e. have identical *capacity* $\in \mathcal{R}$. The cargo is loaded in a pre-defined *depot* (denoted by $c_0$ in our implementation)[1]. Each customer as well as the depot have certain location $loc_j \in \mathcal{R}^2, j = 0, \ldots, n$.

A travel distance $\rho_{i,j}$ is the Euclidean distance between $loc_i$ and $loc_j$ in $\mathcal{R}^2; i, j = 0, \ldots, m$. For each vehicle $v_i$ the $r_i = (i_0, i_1, \ldots, i_{p(i)})$ is a permutation of indexes of requests/customers assigned to $v_i$ to be visited by the vehicle, which

[1]In some CVRP formulations more than one depot is considered, however, the version with one depot is the most popular in the literature.

defines the route of the $i$th vehicle. The first and the last elements in $r_i$ always denote a depot, i.e. each route must start and end in a depot.

The goal is to serve all clients (requests) with minimal total cost (travel distance), with routes/vehicles starting from the depot, with fulfilling the trucks' capacity constraint, visiting each client only once (i.e., for each customer cargo must be delivered in one service) and returning to the depot afterwards. Formally, the goal is to find a set $R = \{r_1^*, r_2^*, \ldots, r_n^*\}$ of permutations of requests that minimizes the following cost function:

$$COST(r_1, r_2, \ldots, r_n) = \sum_{i=1}^{n} \sum_{j=1}^{p(i)} \rho_{i_j i_{j-1}} \tag{1}$$

under the following constraints:

$$\begin{cases} \forall_{i \in \{1,2,\ldots,n\}} & i_0 = i_{p(i)} = c_0 \\ \forall_{i \in \{1,2,\ldots,n\}} & \sum\limits_{l=1}^{p(i)-1} size_{i_l} \leq capacity \\ \forall_{l \in \{1,2,\ldots,m\}} \; \exists!_{i \in \{1,2,\ldots n\}} & l \in r_i \end{cases} \tag{2}$$

In practically all methods applied to CVRP, the solution process is divided into a number of discrete time steps. In each step, all active trucks (i.e. those which left the depot) move to their next clients and the trucks parking in a depot may (but do not have to) commence their routes. The process ends when all vehicles move back to the depot after having visited all the customers. The initial conditions, i.e. the number of available trucks, their capacity, clients requests' sizes, and the coordinates of the depot and the clients, are provided in the problem definition and therefore are known to the solving system *beforehand.*

What makes the CVRPwTJ distinct from the above-described (static) CVRP definition is considering road conditions by means of traffic jams. Specifically, at each time step, for each edge (a direct link between two clients or a client and a depot) the TJ is imposed with probability $P$. If TJ happens to appear on a given edge $a$ it is assigned a randomly selected intensity $I(a)$ and duration/length $L(a)$ measured in time steps.

Therefore, for each edge $a$ of cost $c(a)$, if a traffic jam $TJ(a)$ with intensity $I(a)$ and length $L(a)$ appears on that edge, its current cost is modified to $c(a) \cdot I(a)$ for the next $L(a)$ time steps and reduced to the previous value $c(a)$, afterwards. If TJ happens to be selected for an edge $a$ which is currently jammed (was jammed $k$ time steps before, for some $k$, with the TJ length $L(a) > k$ and intensity $I(a)$), then the length of the TJ on that edge is increased by the newly selected TJ's length, but the cost of that edge is not increased - it remains at the level of $c(a) \cdot I(a)$ (though, for a longer time period). This way we avoid TJ intensity multiplications which might otherwise have easily led to explosive growth. The above TJ assignment (with probability $P$) is applied at the beginning of each time step for all edges (see section 6.1 for traffic parametrization). It is worth to underline that although the times of occurrences, intensities and time spans of the traffic jams are generated according to some pre-selected probability distributions whose parameters are known *a priori*, the actual realizations of TJ are known only once they materialize.

## *2.1. Related work*

VRP formulations considered in the literature include the cases with specifically defined travel times. The most commonly used model of traffic conditions assumes that the congestion is a function of a working day time. For example, in [42], as well as in [5] and [10], time is arbitrarily divided into periods in which the traffic speed is assumed to be fixed (but may differ between time periods). The authors of [23] consider certain traffic-related factors that occur in particular cities at a given time of the day, which are then used to calculate the effective speed of traveling between those cities. Another approach is proposed in [38], where the traveling times are stochastic and the cost function depends not only on the total distance traveled but also the number of vehicles used and the total expected overtime of the drivers.

Generally speaking, each approach has its specificity manifested by either particular definition of the road congestion or by formulation of the ultimate goal and the respective cost function. In this respect it is not possible to compare directly the above-cited papers with our work, since except for a general notion of "changing in time traffic speed" they differ significantly in detailed formulation of the problem and the used method of solving it.

Perhaps the closest to ours CVRP formulation is presented in [21] where both stochastic travel times and online adaptation of the solution by means of Monte Carlo (MC) simulations is proposed. On a less general note, however, both papers differ in the following major aspects. Firstly, in [21] the problem is decomposed into independent single-vehicle problems contrary to global optimization performed in this paper. Secondly, the experiments in [21] are based on an in-house real-time traffic data (owned by logistic company from Singapore) and not on publicly available benchmarks. Lastly, MC simulations are used in [21] only to gather statistical data to be subsequently used by approximate dynamic programming, while our approach uses the simulations directly to construct and modify the routes.

## 3. Upper Confidence Bounds Applied to Trees

UCT is a simulation-based algorithm, which proved to be successful in multi-step decision-making under uncertainty, in particular in building playing agents for several demanding games.

In the case of game-playing framework, the method consists in performing multiple simulations of possible continuations of a game from the current state. Instead of performing fully random rollouts, as is the case of MC method, it proposes a more selective approach to choosing continuations worth analyzing, making the obtained results more meaningful. In each state, UCT advises to choose uniformly any of the actions not yet tried (if one exists) and to choose otherwise move/action a* according to the following formula:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\left[N(s)\right]}{N(s,a)}} \right\} \tag{3}$$

where $A(s)$ is a set of 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, $N(s)$ - a number of times state $s$ has been visited in previous simulations and $N(s,a)$ - a number of times action $a$ has been sampled in this state in previous simulations. Constant $C$ controls the balance between exploration and exploitation, since the formula postulates choosing actions with high expected rewards but, at the same time, avoiding repetitive sampling of the same actions while others might yet prove more beneficial.

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 game state.

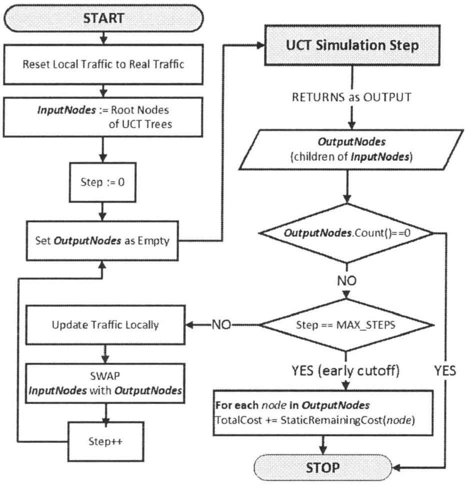
Recently, UCT gained attention also in the area of probabilistic planning implemented by the Partially Observable Markov Decision Process (POMDP) model [24, 19, 15]. Successful application of UCT in this area further motivated our work on CVRPwTJ.

## 4. UCT in CVRP with Traffic Jams

A general top-down view of our method is depicted in Figure 1a. The algorithm starts with finding an initial solution to the static version of the CVRP, which is explained in Section 4.1. The initial solution is used to create initial UCT trees, introduced in detail in Section 4.2. Next, the main loop of the algorithm is executed and performed until all customers are visited. In each step, once the traffic is updated, the algorithm runs a series of the UCT/MC simulations to expand the trees and gather statistics about various transformations of the planned routes. Section 4.3 is devoted to description of this underlying UCT/MC simulation procedure. The UCT trees are expanded by means of simulating various potential modifications of the routes as consequences of taking certain actions. The set of possible actions (local route modifications) is discussed in Section 4.4. Actions are chosen both in the internal UCT/MC simulations as well as in the main simulation used to solve the problem. When actions are chosen in the main simulation, the current state is updated and the UCT trees are trimmed so as to correspond to this state.

```
Benchmark = LoadBenchmark()
InitializeStructures()

StaticSolution = SolveStaticProblem(Benchmark)

// the initial state contains positions and planned routes
// from the static solution
InitialState = CreateFrom(StaticSolution)

uctSimulator.CreateTrees(InitialState)
CurrentState = InitialState

TotalCost = Step = 0
while(CurrentState is NOT terminal)
{
  //update jams according to the benchmark
  CurrentTraffic = UpdateTraffic(Benchmark, Step)

  uctSimulator.Synchronize(CurrentTraffic)
  uctSimulator.Simulate(SIM_COUNT) // the main MCTS routine
  actions[] = uctSimulator.SelectTopActions()

  // compute the next state by moving the vehicles to
  // their new positions (applying actions);
  NextState = Apply(actions[])

 // synchronize the UCT trees accordingly: the root
 // nodes will denote the next state
  uctSimulator.UpdateTrees(actions[]) //

  TotalCost += DynamicCost(CurrentState, NextState)
  CurrentState = NextState
  Step++
}
```

(a)


START
Reset Local Traffic to Real Traffic
InputNodes := Root Nodes of UCT Trees
Step := 0
Set OutputNodes as Empty
UCT Simulation Step
RETURNS as OUTPUT
OutputNodes (children of InputNodes)
OutputNodes.Count()==0
NO
Step == MAX_STEPS
NO
Update Traffic Locally
SWAP InputNodes with OutputNodes
Step++
YES (early cutoff)
YES
For each node in OutputNodes TotalCost += StaticRemainingCost(node)
STOP

(b)

Figure 1: Operational scheme of the proposed UCT-based approach. (a) A pseudocode of the proposed UCT algorithm. The UCT/MC simulations are executed until the allotted time / computational budget is used. (b) A scheme of a single UCT simulation.

### 4.1. Initial solution

The initial solution, in the form of a set of vehicles' routes is obtained **for the static problem instance** with the help of a modified Clark and Wright [7] (CW) *savings algorithm* [34]. The routes are pairwise separated, except for the initial and final position which is always the depot. This set of routes forms an input to the proposed UCT approach.

In general, any method suitable for solving the static version of the CVRP could be chosen to generate the initial solution. The main advantage of the algorithm of our choice is computational speed and deterministic nature by means of yielding the same set of routes when given the same problem instance. The solutions computed by the CW method are between 100% and 112% of the optimal solutions for the (static versions of) benchmark instances used in this paper, which is a reasonably good starting point. Please observe, however, that due to high level of dynamism in the problem definition (in terms of frequent traffic jams) the actual usage of the starting solution is limited to the very short initial period of the algorithm's performance.

### 4.2. UCT trees

Suppose that the initial solution is composed of $k$ routes, i.e. uses $k$ trucks. Then 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 planned to be visited by respective trucks in subsequent time steps (e.g., the third elements in all paths represent the set of clients visited in the second step of the solution). The interpretation of a tree node is valid only in the context of the respective route (called *route-states* representation hereafter). The complete UCT state is defined by a $k$-tuple of *route-states* - one *route-state* per each route.

Note that different sequences of visited customers in a route may lead to the same state, e.g. $[0, 12, 7, 3, \ldots, 0]$ and $[0, 7, 12, 3, \ldots, 0]$ after the first three time steps. In order to keep the size of the UCT trees within a reasonable limits, such isomorphic states (called transpositions) are detected and only one node per each transpositions-set is used. Two states are mapped to one state in the UCT tree iff they share the following data: (1) the current position of the vehicle in the route, (2) the remaining capacity of the vehicle which can be allocated and (3) the remaining set of customers scheduled for the route. The mechanism of *route-states* comparison was implemented with the help of hashing.

## *4.3. UCT simulations*

A single UCT simulation can be regarded as a sequential decision-making process (see Fig. 1b). The initial state of the simulation is composed of the roots of all $k$ trees. In each tree an individual UCT action-selection process is performed which, at each level of the tree, calculates an action to be simulated. Next, in each tree, the chosen action is applied, the current state is updated by the result of the simulated action, the simulation moves one level down the tree, and the simulation step counter is incremented. If any of the actions leads to a terminal state (the depot), then the respective route is completed, the tree is removed from the collection of active trees and will not be used in the current simulation anymore.

In general, simulations are performed in a typical UCT manner, going down the UCT tree at each step. There are, however, three main differences compared to the classical UCT usage presented in section 3. First of all, since the less costly the solution the better, the UCT formula (3) is modified to eq. (4), which favors lower average outcomes $Q(s, a)$:

$$a^* = arg \max_{a \in A(s)} \left\{ C\sqrt{\frac{ln\left[N(s)\right]}{N(s,a)}} - Q(s,a) \right\} \tag{4}$$

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 (4)) and executed in this order afterwards. If the execution of an action in the former tree disables the chosen-as-best action in any of the subsequent trees, then the next best action in the latter tree is selected instead.

The third difference compared to classical UCT implementation is that simulation ends when there are no more active trees or the step counter reaches the value of MAX_STEPS (the so-called *Early Termination*) whichever comes first. We set MAX_STEPS value to the maximum length of a traffic jam. There are few reasons behind using *Early Termination* over finishing at terminal states 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 drastically decreased. We tested both a regular 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 cost (affected by the traffic) 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 possible 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 roots.

After a certain number of the above-described simulations, the actual (real) decision regarding the movement of the $k$ trucks is made according to the smallest $Q(s, a)$ value among the child nodes in each of the $k$ trees, sorted in an ascending order. Hence, the action in the tree with the lowest $Q(s, a)$ is executed first, followed by the action in the tree with the second-lowest $Q(s, a)$, etc.

## *4.4. Possible actions in the UCT trees*

As stated above, at each step of both the UCT simulations and the real decision process 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. For each action there are some pre-conditions under which this action is legal in a given state. If the action is illegal in a given state it is not considered.

In the following description, legality conditions will be placed in the square braces, e.g. **[1-TJ]** / **[no 1-TJ]** will denote the fact that the edge planned to be traversed is jammed / not jammed, respectively. The legality conditions can be regarded as an expert knowledge built into the algorithm in order to avoid combinatorial explosion of possible action sequences if all actions would have been available in each state. A route will be called **fully jammed** if there is a traffic jam from the current vehicle's position to all currently scheduled clients in the route. A total demand of a route is the sum of demands of all clients in the route (both currently scheduled and already visited). Due to the existence of actions which produce a new route as their output the method assumes some number $k'$ of spare trucks at its disposal. Since in none of the tests a demand for additional trucks exceeded one (with the average across all tests equal to 0.3), the setting of $k' = 1$ was experimentally justified as a safe choice.

**Level-0 actions** (no changes are introduced to the routes)

**A0 - continue the planned non-jammed route.** [no 1-TJ].

**A1 - continue the planned jammed route.** [1-TJ].

Actions A0 and A1 are distinguished from each other, despite leading to the same result, due to different legality conditions. Both actions are effectively *noop* operations since no modifications to the current solution are applied. Note that in each non-terminal state there is exactly one level-0 action available, either A0 or A1.

**Level-1 actions** (changes are introduced to exactly one route)

**A2 - Move the current client to the end of a route.** [1-TJ and no 1-TJ after applying the action].

**A3 - Move the current client into locally optimal place in a route.** [1-TJ]. The current client $X$ is inserted in a greedy manner between the two neighboring clients $A$ and $B$, so as to minimize the value of $|AX|+|XB|-|AB|$. In effect, the route $\{X, Y, \ldots, A, B, \ldots, 0\}$ is changed to $\{Y, \ldots, A, X, B, \ldots, 0\}$.

**A4 - Find the first client to whom there is no TJ and insert him as the first in the planned route.** [1-TJ and there exists such a client to whom there is no TJ from the current location].

**A5 - Reverse the route.** [1-TJ and no 1-TJ after applying the action]. The order of the customers to visit is inverted, except for the depot, which remains at the end.

**A6 - Insert the nearest client as the first to visit.** [no 1-TJ].

**A7 - Insert the second nearest client as the first to visit.** [no 1-TJ].

The underlying idea both A6 and A7 is to allow to repair the route which was inefficiently modified by some of the previous actions. The average UCT score $Q$ of these actions (c.f. Equation (4)) is multiplied by 1.15 in order to prevent such greedy choices from happening too often.

**A8 - Complete the current route and start a new one.** [the route is fully jammed; the edge from the current vehicle's location to the depot is not jammed; the edge from the depot to the first customer is not jammed; the limit for available trucks $(k + k')$ is not exceeded].

The current route is completed by moving to the depot and a new route is commenced, initialized with all customers left from the finished route, in the same time step.

**Level-2 actions** (changes are introduced to exactly two routes)

Actions which modify two existing routes are created for each pair of routes if only all the legality conditions are met. Recall that $r_i$ denotes the route assigned to the $i$th vehicle.

**A9 - Move all customers from the current route ($r_i$) to another one ($r_j$).** [$r_i$ is fully jammed and $r_j$ has enough capacity left to accommodate all customers from $r_i$].

The action finishes $r_i$ and the customers are transferred to $r_j$ in the unchanged order. This action has a synergy with A8 whose newly started route has usually less customers than other routes and large part of its capacity remains available.

**A10 and A11 - MIN and MAX exchange between two routes ($r_i$ and $r_j$).** [$r_i$ is fully jammed; there will be no 1-TJ neither on $r_i$ nor on $r_j$ after the exchange; capacity constraints are fulfilled for both routes].

The algorithm in the MIN version is as follows:

1. Start from transferring the first customer from $r_i$ to $r_j$.
2. If the capacity constraints are fulfilled then *STOP*.
3. Continue the exchange of customers by transferring a subsequent one from the route with higher total demand to the other one.
4. Go back to step 2) or *STOP* with a *FAILURE* if there are no more customers for the exchange.

The algorithm in the MAX version is as follows:

1. Initialize *RESULT* as *FAILURE*
2. Start from transferring the first customer from $r_i$ to $r_j$.
3. If the capacity constraints are fulfilled then save the current result as *RESULT*
4. Continue the exchange by transferring a subsequent customer from the route with higher total demand to the other one.
5. Go back to step 2) or *STOP* returning the *RESULT* if there are no more customers for an exchange.

Both actions appear in four cases, each of which combines the above procedure with possible initial reversal of the routes: (1) use $r_i$ and $r_j$ as originally planned; (2) $Reverse(r_i)$; (3) $Reverse(r_j)$; (4) $Reverse(r_i)$ and $Reverse(r_j)$.

**A12 - Complete two routes ($r_i$ and $r_j$) and start a new one ($r_k$).** [$r_i$ is fully jammed; there will be no 1-TJ on $r_k$ after completing the action; the sum of customers' demands on $r_i$ and $r_j$ does not exceed the truck capacity; the limit of available trucks is not exceeded].

This is an extended version of A8, defined in four variants depending on how the new route $r_k$ is constructed: (1) $r_k := r_i + r_j$; (2) $r_k := r_j + r_i$; (3) $r_k := Reverse(r_i) + r_j$; (4) $r_k := Reverse(r_j) + r_i$. "$+''$ denotes simple concatenation of the routes excluding the depot from the first concatenated route.

### Summary

Majority of the proposed actions are based on the following 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 the clients to be visited. 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 unfavorable changes introduced in earlier steps.

In theory, one might proceed with defining even more complex and more sophisticated actions, e.g. the ones involving three or more routes (trucks), but such an approach would fast become infeasible due to its computational complexity. Based on performed experiments we found out that considering actions up to level 2 is a reasonable compromise between efficiency and feasibility of the method.

## 5. Metaheuristic Methods Used for Comparison

As we stated in section 2.1 majority of the papers published on VRP have peculiar assumptions or specific formulations which hinder their direct comparison with our approach. For these reasons we have decided to compare UCT-based approach with several well-established general-purpose metaheuristics, which **are commonly used across a variety of proposed VRP formulations.** After literature review three metaheuristics were selected (GA, TS and ACO) leading to four DVRPwTJ solutions. Due to space limits, another "natural" option, i.e. Particle Swarm Optimization (PSO) [20, 32, 33] was left for future work. Among four competitive solutions the two, denoted GA1 and TS, use exactly the same repertoire of possible actions as UCT does, which makes these methods directly comparable in terms of computational efficiency and quality of solutions. The other two, denoted GA2 and ACO, implement straightforward problem encodings with common-sense sets of actions.

All methods are tested under the same time conditions measured by the number of Fitness Function Evaluations (FFE) performed during the method's run, which is a widely-used measure of assessing method's complexity in Computational Intelligence (CI) and Operation Research (OR) communities.

## *5.1. Two genetic approaches*

Both GA implementations make use of the initial solution computed by the CW savings algorithm in a similar fashion as the UCT algorithm does. This initial solution is then dynamically adapted and updated in the process of genetic optimization in order to react to dynamically changing traffic conditions.

Each individual encodes a complete solution by means of a vector of routes $[r_1, ..., r_k, ... r_{k+k'}]$. Each of the first $k$ routes is a permutation of customers' identifiers with zeros (representing the depot) added as the first and the last element in the route. The first $k$ routes represent the initial solution and the next $k'$ routes denote the spare trucks' routes which are initially empty and ready to be used later during the algorithm's run, i.e.:

$$r_i = \begin{cases} (0, r_i^1, \ldots, r_i^{p(i)}, 0) & \text{for } \ 1 \leq i \leq k \\ (\#E_i) & \text{for } \ k < i \leq k + k' \end{cases} \tag{5}$$

where $\sum_{i=1}^{k} p(i) = m$ and $\#E_i$ denotes an empty route assigned to the $i$-th spare truck. For both methods, the settings from $k' = 0$ to $k' = 5$ were initially tested showing a gradual improvement of obtained solutions until $k' = 4$. For this reason the finally tested range of this parameter was $k' \in \{3, \ldots, 4\}$.

The initial population consists of $N = 200$ identical individuals representing the initial solution. Such a lack of diversity is not harmful due to the use of highly probable and extensive mutation operator which quickly leads to heterogeneity and high variability of individuals.

### GA operational scheme

Both GA implementations solve the problem step-by-step in a similar manner as UCT does, and in each step a certain number of GA generations is performed so as to provide solution for the current step of the main loop of the method. Once the solution is found, the vehicles move one step (to the currently assigned clients), the traffic conditions are updated and the next step of the main loop begins. Selection is performed by ranking all parent and child individuals in the current generation. The fittest $N$ individuals, either parents or children, advance to the next generation. The fitness function is defined as the inverted total length of the solution encoded by an individual. The length is computed by summing dynamic costs of the routes with respect to the current traffic distribution in the main simulation.

The best individual from the last generation in the current step of the main loop represents the solution to pursue. Trucks are moved by one step as scheduled by this solution and the main simulation is updated. Unless all customers have been visited, the evolutionary process is continued from the current state based on the updated traffic conditions.

## Mutation in Genetic Algorithm 1 (GA1)

In the first approach, the mutation algorithm iterates over all non-empty routes, which are randomly permutated, in the solution encoded by an individual. If there is a TJ on the route (to the currently planned customer), then the route is affected by the mutation operation with probability 0.95. If there is no 1-TJ situation, then the probability is equal to 0.01. Both above-mentioned mutation probabilities were selected based on a special tuning procedure discussed in Section 6.1.

The set of possible mutations includes exactly the same actions that are used in the UCT approach with the exception of the non-modifying Level-0 actions, which are applied by default if the mutation does not happen to occur for a particular route or there are no other legal actions available. All other action (with ID>1) which are legal by means of the above-defined legality conditions have pairwise equal probability of being chosen. Actions which do not fulfill legality conditions are not considered.

Actions which operate on two routes accept the second route only among those which are further in the iteration order (thus have not been mutated yet). Should such a case occur, the second route is marked as mutated and not considered for mutation again in its turn.

**Mutation in Genetic Algorithm 2 (GA2)**

In the second genetic approach (denoted GA2), mutation is implemented in a "classical" way as a random swap of two customers within an individual. Recall that each individual which encodes a $k$-route solution is represented as a vector of $m+2k+k'$ elements (c.f. Equation (5)), with $2k$ zeros (beginning and ending of each route), $k'$ elements of the form $\#E_i, i = 1, \ldots, k'$ representing empty routes of $k'$ spare vehicles and $m$ elements representing customers assigned to particular trucks (see Figure 2 as an example).

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 5 | 1 | 2 | 7 | 6 | 10 | 0 | 0 | 3 | 4 | 8 | 9 | 0 | $\#E_1$ | $\#E_2$ |
| slots: | | 1 | 2 | 3 | 4 | 5 | 6 | | | 7 | 8 | 9 | 10 | | 11 | 12 |

Figure 2: A sample encoding of a chromosome composed of two routes: $[0, 5, 1, 2, 7, 6, 10, 0]$ and $[0, 3, 4, 8, 9, 0]$ ($k = 2$ and $m = 10$) and two spare trucks: $\#E_1$ and $\#E_2$ which can be used if needed (with empty routes at the moment). The numbers below the boxes denote the numbering of these slots (gens) which are available for selection in the mutation operation.

In mutation operation, two different genes, corresponding to clients or spare vehicles, are drawn randomly, with uniform probability. Depending on the result, the following 4 cases are possible: **(1). Identifiers belong to the same route:** the respective customers are swapped. **(2). Identifiers belong to different non-empty routes:** the respective customers are swapped between the routes if the operation does not violate capacity constraints. Otherwise, the operation fails (is canceled). **(3). Identifiers belong to different empty routes:** this operation is always canceled. **(4.) Exactly one identifier belongs to a non-empty route:** a new route is commenced with the respective customer as the unique client. The customer is removed from its previous route.

If the operation is canceled, it is repeated with a new pair of randomly sampled identifiers. If it fails again, then the mutation is finished with no effect. The mutation operation is executed $R_{TJ}$ times with probability 0.95 and $R_F$ times with probability 0.05, where $R_{TJ}$ is the number of routes with 1-TJ, whereas $R_F$ denotes the number of routes without 1-TJ. Both probabilities were optimized based on preliminary tests.

In both GA implementations a population of $N = 200$ individuals was maintained for $GEN = 150$ generations, which is equivalent to 30 000 simulations made in each UCT run (see section 6.4 for further discussion on methods' complexity).

## 5.2. Tabu Search approach (TS)

The main loop of the TS approach was implemented in a classical way based on [8] and [37] with the main components (solution perturbation, local search, and intensification) being optimized towards the DVRPwTJ specification, which is a common approach to VRP (c.f. [26] for example). In each iteration of the main loop (i.e. after the TJ injection), for a given number of steps $M$, the current solution $S_0(M)$ is perturbed (leading to a state $S'(M)$) and a local search procedure is run in the neighborhood of $S'(M)$. In effect a new locally optimal solution state $S''(M)$ is obtained and the tabu lists for all routes are updated. Furthermore, at certain intervals ($IntIvs$) the intensification procedure is applied which attempts to significantly enhance the current solution.

The perturbation procedure uniformly selects one of the routes and one of the possible (legal) actions among those defined in section 4.4, which is then applied to that route with some probability $p_{tabu}$. If the action is of level-2 type a compound route is uniformly selected among all other suitable routes. In the local search procedure, for a randomly selected route $r_i$ all legal actions which are not present on the tabu list are first identified. These actions are then tried in random order and the first one which improves the current solution is accepted and executed as a result of the local search improvement procedure. The selected action is added to the tabu list $tabu_i$ associated with the route $r_i$.

The intensification procedure attempts to build a solution from scratch for all yet unserved customers using the *savings algorithm* [34], with the savings of the form $C(i) \to j$ and $j \to 0$, where $C(i)$ denotes the current customer visited in route $r_i$, 0 is the depot and $j$ denotes any customer other than $C(i)$ among those not yet served.

At the end of each iteration step the tabu lists (one per route) are updated by removing the actions which were added to the list more than $s_{tabu}$ steps before.

In order to make the comparison with UCT approach fair the number of iteration steps (M) was calculated as follows:

$$M = UCT_{Sim} * MAX\_STEPS/(UCT_L + 1) \tag{6}$$

where $UCT_{Sim}$ is the number of iterations of the UCT algorithm (each of them lasting for at most MAX_STEPS steps), and $UCT_L$ is the average number of actions available among all route-states. One is added to compensate the use of intensification procedure, which is computationally intensive, though not performed in each step ($IntIvl$ was set to 300). The remaining parameters $p_{tabu} = 0.3$ and $s_{tabu} = \frac{3\sqrt{K}}{2}$ (where $K$ denotes the number of active routes) were set based on grid-based preliminary tuning.

## *5.3. Ant Colony Optimization approach (ACO)*

Our implementation of the ACO approach is inspired by a standard algorithm used to solve the Traveling Salesman Problem [11, 12] enhanced to take into account the VRP specificity [2, 30] and furthermore the stochastic nature of the DVRPwTJ. In each main-loop iteration, each ant starts with the current state and 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. The initial solution computed by the CW algorithm [34] is used to set the starting number of routes ($k$) and deposit the initial pheromone.

At each time step, each ant, for each route starts its search in 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 the truck's capacity is not sufficient to serve any of the remaining customers or there are TJ to all customers from a particular position, the truck is sent to the depot and a new route is commenced.

The customer selection procedure chooses the next client to be visited using the pseudo-roulette. The closest (in terms of dynamic cost) unvisited customer is selected as next to be visited with probability 0.05 and with probability 0.95 the next customer is chosen according to the roulette-wheel selection with the following probabilities:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{ij}(\tau_{ij}^{\alpha} * \eta_{ij}^{\beta})} \qquad \eta_{ij} = (BASE/d_{ij})^2 \tag{7}$$

where $\tau_{ij}$ denotes the pheromone amount deposited on the edge $e_{ij}$ and $d_{ij}$ is the current dynamic (traffic-aware) cost of traversing this edge. $BASE$ 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.

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

$$\Delta\tau_{ij} = \sum_{a} \delta_{ij}(BASE/D_a)^2 \tag{8}$$

where $D_a$ denotes a dynamic cost of solution $s(a)$ found by ant $a$, and $\delta_{ij}$ can take one of the three values: **0**: if $e_{ij} \notin s(a)$; **10**: if $e_{ij} \in s(a)$ but $s(a)$ is not the globally best solution; **20**: if $e_{ij} \in s(a)$ and $s(a)$ is the overall best solution. After pheromone evaporation procedure we obtain the final amount of pheromone $\tau_{ij}$ deposited at the beginning of the next iteration:

$$\tau_{ij} := Conf(0.1 * \tau_{ij} + \Delta\tau_{ij}) \tag{9}$$

where $Conf$ confines pheromone values to pre-defined interval $[\tau_{min}, \tau_{max}]$.

Once the last iteration is completed by all ants, the best solution is found. It is used only for one step to move the trucks according to the schedule represented by this solution. Afterwards, the best solution is forgotten and the pheromone trails are reset in a similar way as in the beginning of the algorithm. New TJ are distributed and the system proceeds with solving the next step of the problem. Numerical tests clearly confirmed that resetting pheromone traits after each main simulation step, i.e. when the new TJ are imposed on the routes, is indispensable, since the problem is highly dynamic and the previous traces are misleading.

ANT algorithm was run with a population of $MAX\_A = max(100, 2n)$ ants, for $MAX\_I$ iterations. $MAX\_I$ was set to 200 for benchmarks with $n < 70$ and 75 for benchmarks with $n \geq 70$. 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.

## 6. Experimental Results

In this section, the experimental setup and traffic jams' parametrization are presented, followed by the results of applying the proposed approach to a set of popular static CVRP benchmarks modified by imposing the TJ. The results of the UCT method are compared with those of the two GA, ACO and TS systems described above, and then tested for statistical significance. In the last subsection the methods are compared on the ground of computational complexity.

For each benchmark set, a certain number of instances with different realizations of the traffic was prepared and saved. Each method was tested using the same set of saved instances during the main simulation[2] (the actual vehicles' movement decision process). Concrete realizations of traffic are not known to the methods beforehand. Therefore, there is a significant degree of uncertainty and dynamism introduced into the problem.

Each main simulation is performed in discrete time steps. The movement of vehicles is performed simultaneously by means of atomic state update oper-

[2]The word *simulation* is used in the paper in two meanings. Firstly, it denotes an internal UCT simulation, secondly it refers to the actual decision process regarding vehicles' movement. These two situations should be easily recognizable by the context, but in the case of potential ambiguity a word *main* or *actual* will be added to *simulation* so as to point the latter case.

ations, during which each active vehicle is moved from its current customer (or depot) to the next scheduled one. In each step, all vehicles which are not located in a depot must be moved. Consequently, an upper margin for the number of steps in a simulation is equal to the number of customers plus one (return to the depot).

### *6.1. Experimental setup*

A diversified set of static benchmark problems downloaded from the CVRP webpage [31] and transformed into CVRPwTJ instances by adding uniformly distributed TJ at each time step was used in the experiments. More precisely for a given edge $a$, the noise at time step $t$ was selected with probability distribution $P_t(a)$, intensity $I_t(a)$ and duration $L_t(a)$, within the following ranges:

$P_t(a) \in \{0.02; 0.05; 0.15\}$

$I_t(a) = U_{INT}[10, 20]$

$L_t(a) = U_{INT}[2, 5]$

where $U_{INT}[a, b]$ denotes random uniform selection of any integer $x$ such that $a \leq x \leq b$.

Please recall, however, that as stated in section 2, in order to prevent the noise intensity from explosive growth, if $a$ was already jammed at time $t$ (as a consequence of TJ distribution in one of the previous time steps) then when selected at time $t$, the noise would not change its intensity (i.e. $I_t(a) := I_{t-1}(a)$), and only the length of TJ on that edge would be increased by $L_t(a)$ (i.e. $L_t(a) := L_t(a) + L_{t-1}(a)$).

The set of tested benchmarks was chosen blindly, however, the final selection reflects quite significant variability in sizes and the estimated numbers of routes in the optimal solutions. Also the distributions of clients requests' sizes and their geographical (2D) locations are varying from benchmark to benchmark, including examples of large benchmarks with uniformly distributed customers (C-n150D-k12) or more clustered ones (Tai-n150b-k14).

The UCT approach is parameterized by one coefficient only, the so-called exploration constant, denoted by $C$ in (4). In games domain, this parameter is usually set to $Q_{MAX}$ or $\sqrt{2}Q_{MAX}$ where $Q_{MAX}$ is the maximum possible numerical outcome (score) of a single simulation. Since the best score is unknown in our problem, we propose to replace it with the cost of the initial solution (assuming no traffic) which is a good approximation of the best possible result. In order to trace the intrinsic properties of UCT application in this new domain, instead of using $\sqrt{2}$ as a pre-selected multiplier we tested values between 0.5 to 2.0 with step 0.1 as well as a few greater ones as candidate multipliers. Interestingly, it came out that the best results were observed within the whole range of $[0.7, 1.8]$ interval without the clear winner. The differences among the results for parameters belonging to this interval were statistically insignificant. This observation, together with promising numerical results, is a strong indication of flexibility and robustness of the proposed method, as it can perform effectively within a relatively wide range of its steering parameter selection. After a closer examination of this initial results we decided to use both boundary values, i.e. 0.7 and 1.8, for the results presentation, as they exhibit slightly distinct patterns of action distributions.

The GA1 approach operates with two parameters, which denote the probabilities of mutation for a currently jammed and non-jammed routes, respectively. A dedicated optimization experiment was conducted to find the best settings of these parameters. To this end we have performed grid search checking the Cartesian product of the set $\{1, 0.95, 0.85, 0.75, 0.65\}$ for the first parameter and the set $\{0, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.15\}$ for the second one. The best results were found for the pair $\{0.95, 0.01\}$.

### *6.2. Numerical results*

For each of 57 pairs (benchmark, $P$) 50 experiments were performed for all five approaches (UCT, GA1, GA2, ACO and TS). In a given experimental trial (one out of 50) the same TJ realizations were used for all tested methods, and certainly different realizations were used in different trials. It should be underlined that these TJ realizations were not known to the tested algorithms beforehand (i.e. unless the traffic jams actually materialized in a given time step). Hence, solving the CVRPwTJ required truly *on-line* and self-adaptive probabilistic planning capabilities. The main results are presented in Table 1.

Due to the lack of space in Table 1, we did not show results of the "static solution", which consists in applying TJ realizations to the initial solution without any routes' modifications. Inferiority of such approach is a foregone conclusion with a few-times longer tours, in average, compared to the winning scenarios. For the same reasons (space savings), for each GA implementation only the results obtained with a more efficient selection of $k\prime$ (which happened to be $k\prime = 4$ for both GA1 and GA2) are presented.

On a general note, it can be seen that UCT and GA1 are undisputable winners of this comparison. Out of 57 combinations of a benchmark and probability $P$, when counting multiple winners when two or more methods tied the best result, GA1 performed most effectively in 18 cases, followed by UCT-0.7 and UCT-1.8 (15 and 13 wins, respectively), GA2 (12 wins) and ACO (6 wins). TS was only tied for 2 wins (but, at the same time, was also very rarely the worst method). Since the above-mentioned general statistics may be biased the selection of methods used for presentation we also compared the 3 best performing methods directly. In this pairwise comparison the winner is UCT-1.8 scoring 30 : 25 : 2 (win:loss:draw) against GA1, 32 : 23 : 2 against UCT-0.7. The outcome of GA1 vs. UCT-0.7 equaled 24 : 31 : 2 (the advantage of UCT-0.7). All 3 results are statistically significant (section 6.3 discusses this topic in more detail). While the above statistics clearly confirm a dominance of UCT and GA1 over the competitive methods in terms of quality, what seems to be **a truly salient feature of the UCT approach is the stability of results.** When standard deviations (for 50 tests in each of 57 cases) are compared UCT-1.8 leads with 25 wins, followed by UCT-0.7 (24 wins), GA1 (7 wins) and TS (1 win). GA2 and ACO did not win in any of the test cases. The standard deviation of both UCT variants is, on average, as low as 7% only while the same measure for other methods yields 9% (GA1 and TS), 15% (GA2) and 18% (ACO).

Stability of the UCT method seems to be a real advantage in the real-world

DVRP scenarios, and beyond dynamic transportation problems. We believe that repeatability of results is a crucial asset of the proposed method as it makes the outcomes more reliable and more feasible from the real-world deployment viewpoint. The advantage of UCT-0.7 becomes more visible in the case of high TJ probability ($P = 0.15$) when the ability of on-line adaptation to the new traffic conditions is a critical facet. It stems from the experimental results (9 wins out of 19) that adaptability of UCT with a lower exploration factor to the dynamic changes in the problem parametrization is visibly greater than in the competitive methods. UCT-1.8 and GA1 came second in this classification, tied for 4 wins only.

Generally speaking, the action-based methods led to visibly better results than the remaining two approaches. In particular, UCT and GA1, which are both action-driven and involve testing multiple variants in a given situation, i.e. by building a game-tree (in case of UCT) or managing a population of solutions (in case of GA1) significantly outperform the other approaches. TS, which is also based on actions, proved to be not as strong method probably because it does not maintain a set of candidate solutions, but instead traverses the solution space with a single path.

The above-mentioned differences in methods' suitability to particular amount of uncertainty in the problem formulation, imposed by the occurrence of TJ, stem from various expansiveness of local optimization operators implementation in the respective methods. Observe that in the case of $P = 0.02$, the quality of initial solution (which is high when using CW method [34]) plays a crucial role in the final result, due to a low degree of problem's dynamism. In the case of higher amount of uncertainty GA1 and UCT are undoubtedly the best suited among all tested methods. On the other hand, despite potential incentives of having a well-suited repertoire of actions, GA1 cannot compete on equal terms with the UCT in the highest $P$ regime. Along with the increase of the amount of randomness in the problem, the advantages of the UCT approach (related to its natural adaptability due to simulation-based operational scheme and internal way of maintaining a balance between exploration and exploitation) start to gradually manifest themselves.

### *6.3. Statistical significance*

The results presented in previous subsection showed an advantage of UCT-0.7, UCT1.8 and GA1 over the remaining tested approaches and additionally some upper-hand of both UCT methods over GA1 (in further one-to-one direct pairwise comparison).

In order to verify whether the differences in results are in fact significant *double-sided paired t-Student tests with significance level $\alpha$=0.95 (p-value=0.05)* were performed with $H0$ hypothesis stating that the differences between averages are statistically insignificant. Ten test was performed for all possible pairs of methods, including GA2 approach (15 cases).

When considering the average values over all 57 test cases the differences between UCT-0.7, UCT-1.8, GA1 (TOP3) are not significant. All the remaining pairwise differences are statistically significant, except for the case of GA2 vs. ACO. For $P = 0.15$ all pairwise differences are statistically significant except for the case of GA1 vs. TS. In the case of $P = 0.05$, the differences within TOP3 are significant, and the remaining differences are not significant except for the cases GA2 vs. TS and GA2 vs. ACO. Finally, for $P = 0.02$ all differences within the set of 4 methods (TOP3 and GA2) are insignificant and the rest of the cases presents significant differences, except for GA1 vs. TS.

In summary, it could be concluded from the above that the TOP3 methods are generally significantly better than the remaining 3 approaches and in the case of $P = 0.15$ the UCT approach additionally significantly outperforms GA1. In the case of $P = 0.02$ the TOP3 set should be extended by the GA2 implementation which is not significantly different the leading approaches.

### *6.4. Computational complexity*

In this section insights into computational efficiency of the examined approaches are presented. First of all, for a given benchmark set, the main simulation lasts for about the same number of steps for all considered methods.

Likewise applying the result of each step is computationally comparable across all the methods. Consequently, what effectively matters in the comparative analysis is the cost of a single core iteration, which in each case is implemented as a kind of a local search procedure.

The most common ground for estimating the complexity of local search for both population-based and simulation-based methods is the number of the FFE (Fitness Function Evaluations). For the UCT approach, the number of FFE is equal to the number of simulations in each step (which equals 30 000) multiplied by the number of steps. In both GA implementations, in each step the number of FFE is equal to the number of generations ($GEN = 150$) multiplied by the number of individuals ($N = 200$), which is equal to 30 000 to match the complexity of the UCT approach. The number of iterations in TS was also set accordingly, to make the numbers of FFE in TS and UCT equal, as defined in eq. (6). In the case of ACO, the number of FFE in each step is equal to the number of ants ($MAX_A$) multiplied by the number of iterations ($MAX_I$). Although $MAX_A * MAX_I$ is lower than 30 000 ($N \cdot GEN$ in GA) these two parameters were really pushed to the limits, since ACO is a significantly slower method (due to the way it constructs a solution) and it would be impractical to level up these parameters to the evolutionary ones or TS. The number of steps which is derived by the longest route created for a particular problem may vary between benchmarks, but for a given instance these values are close to each other across all tested approaches. In effect, the estimated complexities of the methods, measured as the number of FFE evaluations, are fairly comparable.

In terms of real running times, it can be observed that ACO is significantly slower than the remaining approaches. Among GA1, UCT and TS, which all share a common set of route modification operators, the UCT is the fastest, at the expense however of higher memory requirements (since the method stores each modification in memory, many repeated calculations are avoided), but only by a small margin. Finally, since GA2 uses the simplest form of mutation it is slightly faster than GA1, UCT and TS. Overall, in practice the differences among these four approaches are negligible.

## 6.5. *Statistics of chosen actions*

Among the two genetic methods, GA2 applies a classical form of mutation, commonly used in the CVRP literature. The mutation operator in GA1, however, is a direct implementation of the set of actions utilized by the UCT method and, to the best of our knowledge, was never proposed before. Similarly, the neighborhood structure used by TS is induced from the same set of actions that is utilized by both UCT and GA1. This common selection of possible actions/movements in the search space provides a unique opportunity for a direct comparison of the UCT, TS and GA methods *in terms of their internal underlying activity patterns*. To this end, a thorough comparison of the frequency of selection of particular actions in the three methods was performed. For each of the three values of $P$ the numbers of particular actions' selections (from $A0$ to $A12$) were summed across all benchmarks for GA1, TS, UCT-0.7 and UCT-1.8 (in both cases of UCT, separately for the simulation phase (denoted by *UCT-sim*) and decision (moving) phase (denoted by *UCT-play*). Several conclusions can be drawn from analysis of this action-selection data.

**(1):** while differences between both UCT versions in the play (decision) phase are negligible, when it comes to the simulation phase the action selection patterns differ in a few aspects. First of all, with higher value of $C$ the method prefers more explorative behavior and hence the usage of action $A0$ ("move along the non-jammed route, as planned") is visibly lower for $UCT-1.8$. The usage of $A0$ by UCT-0.7-sim is equal to 63.4%, 80.6% and 87.3% for $P = 0.15, 0.05, 0.02$, respectively, whereas the respective values for UCT-1.8-sim are equal to 55.9%, 67% and 72%. Leaving $A0$ aside, the next most frequently used actions by UCT-sim were $A4$, $A6$, $A10$ ($P = 0.15$), $A5$, $A6$, $A7$ ($P = 0.05$) and $A6$, $A7$, $A4$ ($P = 0.02$).

**(2):** the UCT action-selection profiles clearly differ between UCT-sim and UCT-play. This is especially apparent when comparing the usage of $A0$. This action is played by both variants of UCT, on average, in 73%, 88% and 94% of the cases for $P = 0.15, 0.05, 0.02$, respectively. There are two main reasons for this discrepancy. First of all, statistics of UCT-sim actions are collected at several levels of the UCT trees while UCT-play represents the statistics of the exactly those actions that were performed. Second of all, the legality of actions frequently tested during simulations might have changed at the decision-taking time, making them impossible to be applied to the real route modification. And conversely, some less frequently tested actions might have apparently become the most favourable ones due to the changes of legality conditions. Apart from $A0$, the most frequently played actions by UCT were $A4$, $A3$, $A5$ ($P = 0.15$), $A4$, $A3$, $A2$ ($P = 0.05$) and $A4$, $A3$, $A6$ ($P = 0.02$).

**(3:)** vast preference of GA1 for applying action $A0$ (if available) can be observed, especially for lower values of $P$. This action is selected by GA1 on average in 91.3%, 95.68% and 96.9% of cases, for the respective values of $P$. While such a preference for exploitative behavior (follow the planned non-jammed edge if possible) proves sufficient for relatively stable situations ($P = 0.02, 0.05$), it apparently becomes not adequate for the cases of larger amount of noise ($P = 0.15$), where more exploratory attitude is desired. In such cases, UCT seems to maintain the exploration-exploitation balance more efficiently. In contrast to GA1, TS chooses $A0$ only in 57.3%, 71.3% and 76.5% of the cases, for $P = 0.15, 0.05, 0.02$, respectively, but instead it prefers actions $A6$ and $A7$ visibly more than the other two methods. The combined usage of $A6$ and $A7$ by TS equals 17.2%, 19.3% and 19.4%. For comparison, UCT chooses them in 9.4%, 9.2% and 8.4% of the cases, and GA1 in 2.2% only (regardless of the setting of $P$).

**(4):** while generally the usage of complex actions involving two routes ($A9$-$A12$) is not impressive in the absolute figures it is, nevertheless, visibly much higher in the case of UCT. For $P = 0.15$, the combined usage of actions $A9$-$A12$ equals 9% (UCT-1.8-sim), 8.6% (UCT-0.7-sim), 6.6% (TS), 5.2% (UCT-0.7-play and UCT-1.8-play), 0.5% (GA1). This distinction most probably constitutes one of the decisive factors of the UCT dominance in bigger and/or more noisy benchmarks. Even though complex actions are not used frequently, they often serve as a last resort instance when the current solution process is trapped in a local minimum with all relevant edges being jammed. Due to substantial routes' modification actions $A9$-$A12$ allow for making a "long jump" in a solution space where new options may possibly emerge.

**(5):** besides actions $A0, A6, A7$, all three methods assign higher than average preference to applying action $A4$ ("searching for the first customer with non-jammed route") - especially in the regime of $P = 0.15$.

In summary, the above analysis confirms high flexibility of the UCT approach to DVRPwTJ and its generally higher adaptability to the temporal changes in the problem definition compared to GA1 and TS. This better overall suitability of UCT should, most probably, be attributed to its frequent exploratory behavior manifested by skipping the obvious choice of $A0$ option and searching for other possibilities, as well as, by applying complex actions involving two routes which consequently introduce large changes in the current partial solution. Despite efforts that had been devoted to accomplishing a similar operational scheme in GA1 and TS implementations we were unable to find a suitable parametrization that would lead to the qualitatively similar activity pattern in the case of these two algorithms.

## 7. Conclusions and Directions for Future Research

This paper proposes a new approach to solving the Capacitated Vehicle Routing Problem with Traffic Jams. Presented solution relies on the UCT algorithm, which was hitherto applied mainly in game domain (in particular to games for which a compact and meaningful evaluation function is not known) and to Partially Observable Markov Decision Processes in the area of probabilistic planning. Application of UCT to solving CVRPwTJ required suitable problem representation (in the form of a forest of UCT trees) and specific definition of legal actions to be performed in these trees in order to prevent the method from explosive growth of memory requirements.

The UCT-based results were successfully compared with Ant Colony Optimization method, Tabu Search and two approaches employing Evolutionary Algorithms. All five methods were tested on a selection of diversified widely-used benchmark problems. The advantage of proposed UCT approach was manifested in the strongest way in the case of the largest benchmark problems and/or higher levels of uncertainty (imposed into the problem definition by means of stochastic occurrence of traffic jams).

Further investigation into the issue of action-selection schemes of UCT, TS (which uniformly selects one of the legal actions in its perturbation procedure) and GA1 (with the mutation range equivalent to the set of UCT actions) revealed interesting differences between operational preferences of these approaches. In turned out that UCT is more leaned to explore possible changes in the assigned schedules even though the current schedule does not suffer from the existence of the TJ on the edge to be traversed (action $A0$ is available). Furthermore, the relative usage of complex actions (involving two routes) is visibly higher in the UCT performance scheme compared to the cases of GA1 and TS. Most probably, these complex actions ($A9$-$A12$) serve as the last resort in the highly jammed situations in which simple, local improvement schemes are ineffective. It seems reasonable to say that the two above-mentioned operational differences are largely responsible for the UCT upper-hand in the case of high TJ probability and/or large benchmark instances.

We believe that UCT, when applied to CVRPwTJ, has two important merits which were manifested during the experiments. First of all, the method is flexible in terms of parameterization. It is easy to apply the educated guess approach and effectively set parameter $C$ in eq. (4), which is actually *the only* method's parameter. What can be even more important is that within a wide range of reasonable selections of $C$ the results are qualitatively comparable. While the internal simulation pattern differs depending on particular setting of $C$, the final decisions (applied actions) are repeatable within the whole range of tested $C$ values.

The other asset of the proposed method is the repeatability of results. When compared to GA1, TS and ACO the average standard deviation of the UCT across all tested benchmarks is lower by at least 20%.

Our current focus is on applying the UCT approach to solving other types of stochastic transportation problems, in particular the so-called Vehicle Routing problem with Dynamic Requests, where only part of the customers' destinations and demands is known beforehand and the remaining information is gradually revealed to the dispatcher during the working time hours. This problem formulation seems to be well-suited for the UCT approach as it requires prediction (or *simulation*) of unknown demands' distributions in both time and space dimensions.

## Acknowledgment

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

## References

[1] M. M. Abdulkader, Y. Gajpal, T. Y. ElMekkawy, Hybridized ant colony algorithm for the multi compartment vehicle routing problem, Applied Soft Computing 37 (2015) 196 – 203.

[2] J. E. Bell, P. R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Advanced Engineering Informatics 18 (1) (2004) 41–48.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games 4 (1) (2012) 1–43.

[4] D. Cattaruzza, N. Absi, D. Feillet, T. Vidal, A memetic algorithm for the multi trip vehicle routing problem, European Journal of Operational Research 236 (3) (2014) 833 – 848.

[5] H.-K. Chen, C.-F. Hsueh, M.-S. Chang, The real-time time-dependent vehicle routing problem, Transportation Research Part E: Logistics and Transportation Review 42 (5) (2006) 383–408.

[6] N. Christofides, A. Mingozz, P. Toth, Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, Mathematical Programming 20 (1) (1981) 255–282.

[7] G. Clarke, J. Wright, Scheduling of vehicles from a central depot to a number of delivery points., Operations Research 12 (4) (1964) 568–581.

[8] J.-F. Cordeau, M. Maischberger, A Parallel Iterated Tabu Search Heuristic for Vehicle Routing Problems, Computers & Operations Research 39 (9) (2012) 2033 – 2050.

[9] G. B. Dantzig, J. Ramser, The truck dispatching problem, Management Science 6 (1) (1959) 80–91.

[10] A. V. Donati, R. Montemanni, N. Casagrande, A. E. Rizzoli, L. M. Gambardella, Time dependent vehicle routing problem with a multi ant colony system, European journal of operational research 185 (3) (2008) 1174–1191.

[11] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. thesis, Politecnico di Milano (1992).

[12] M. Dorigo, L. M. Gambardella, Ant colonies for the travelling salesman problem, BioSystems 43 (2) (1997) 73–81.

[13] S. Eilon, C. Watson-Gandy, N. Christofides, Distribution Management: Mathematical Modelling and Practical Analysis, 1st ed., Griffin, 1976.

[14] B. Eksioglu, A. V. Vural, A. Reisman, The vehicle routing problem: a taxonomic review, Computers & Industrial Engineering 57 (4) (2009) 1472 – 1483.

[15] Z. Feldman, C. Domshlak, On Monte-Carlo tree search: To MC or to DP?, in: Proceedings of ECAI-14. 21st European Conference on Artificial Intelligence, 2014.

[16] M. Fisher, R. Jaikumar, A Decomposition Algorithm for Large-scale Vehicle Routing, Dep. of Decision Sciences, Wharton School, Univ. of Pennsylvania, Philadelphia, 1978.

[17] S. Gelly, D. Silver, Monte-carlo tree search and rapid action value estimation in computer go, Artificial Intelligence 175 (11) (2011) 1856–1875.

[18] K. Ghoseiri, S. F. Ghannadpour, Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm, Appl. Soft Comput. 10 (4) (2010) 1096–1107.

[19] T. Keller, P. Eyerich, PROST: Probabilistic planning based on UCT, in: Proceedings of International Conference on Automated Planning and Scheduling, 2012, pp. 119–127.

[20] M. Khouadjia, E. Alba, L. Jourdan, E.-G. Talbii, Multi-Swarm Optimization for Dynamic Combinatorial Problems: A Case Study on Dynamic Vehicle Routing Problem, in: Swarm Intelligence, vol. 6234 of LNCS, Springer, Berlin / Heidelberg, 2010, pp. 227–238.

[21] G. Kim, Y. S. Ong, T. Cheong, P. S. Tan, Solving the dynamic vehicle routing problem under traffic congestion, IEEE Transactions on Intelligent Transportation Systems 17 (8) (2016) 2367–2380.

[22] L. Kocsis, C. Szepesvri, J. Willemson, Improved monte-carlo search, Working Paper (2006).

[23] A. Kok, E. Hans, J. Schutten, Vehicle routing under time-dependent travel times: the impact of congestion avoidance, Computers & Operations Research 39 (5) (2012) 910–918.

[24] A. Kolobov, Mausam, D. S. Weld, LRTDP versus UCT for online probabilistic planning, in: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012, pp. 1786–1792.

[25] J. K. Lenstra, A. R. Kan, Complexity of vehicle routing and scheduling problems, Networks 11 (1981) 221–227.

[26] S. C. Leung, X. Zhou, D. Zhang, J. Zheng, Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem, Computers & Operations Research 38 (1) (2011) 205 – 215.

[27] J. Mańdziuk, Knowledge-Free and Learning-Based Methods in Intelligenet Game Playing, vol. 276 of Studies in Computational Intelligence, Springer-Verlag, Berlin, Heidelberg, 2010.

[28] J. Mańdziuk, A. Żychowski, A memetic approach to vehicle routing problem with dynamic requests, Applied Soft Computing 48 (2016) 522 – 534.

[29] R. Masson, S. Ropke, F. Lehud, O. Pton, A branch-and-cut-and-price approach for the pickup and delivery problem with shuttle routes, European Journal of Operational Research 236 (3) (2014) 849 – 862.

[30] S. Mazzeo, I. Loiseau, An ant colony algorithm for the capacitated vehicle routing, Electronic Notes in Discrete Mathematics 18 (2004) 181–186.

[31] NEO. Networking and Emerging Optmization, http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/.

[32] M. Okulewicz, J. Mańdziuk, Application of Particle Swarm Optimization Algorithm to Dynamic Vehicle Routing Problem, in: Artificial Intelligence and Soft Computing, vol. 7895 of LNCS, Springer Berlin Heidelberg, 2013, pp. 547–558.

[33] M. Okulewicz, J. Mańdziuk, Two-Phase Multi-Swarm PSO and the Dynamic Vehicle Rouring Problem, in: IEEE Symposium on Computational Intelligence for Human-Like Intelligence, IEEE Press, 2014, pp. 86–93.

[34] T. Pichpibul, R. Kawtummachai, An improved clarke and wright savings algorithm for the capacitated vehicle routing problem, Science Asia (2012) 307–318.

[35] Stanford University, Stanford gamemaster online repository, http://gamemaster.stanford.edu/showgames (2012).

[36] M. Świechowski, J. Mańdziuk, Self-adaptation of playing strategies in general game playing, IEEE Transactions on Computational Intelligence and AI in Games 6 (4) (2014) 367–381.

[37] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, A Tabu Search Heuristic for the Vehicle Routing Problem With Soft Time Windows, Transportation Science 31 (2) (1997) 170–186.

[38] D. Tas, M. Gendreau, N. Dellaert, T. van Woensel, A. de Kok, Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach, European Journal of Operational Research 236 (3) (2014) 789 – 799.

[39] F. Teytaud, O. Teytaud, Creating an upper-confidence-tree program for havannah, Advances in Computer Games (2010) 65–74.

[40] P. Toth, D. Vigo, Special issue on vehicle routing and distribution logistics, European Journal of Operational Research 236 (3) (2014) IFC –.

[41] K. Walędzik, J. Mańdziuk, An Automatically-Generated Evaluation Function in General Game Playing, IEEE Transactions on Computational Intelligence and AI in Games 6 (3) (2014) 258–270.

[42] Y. Xiao, A. Konak, Green vehicle routing problem with time-varying traffic congestion, in: Proceedings of the 14th INFORMS Computing Society Conference, 2015, pp. 134–148.

Table 1: The average values across 50 trials. For GA1 and GA2 methods the results for better performing variant are presented, which in both cases was that of $k' = 4$. For each benchmark, there are 3 lines of results: for $P = 0.02$ (top), $P = 0.05$ (middle) and $P = 0.15$ (bottom). Benchmarks are shown in two groups: the left one (columns 1-7) and the right one (columns 8-14). The best results are bolded. The last row presents the number of winning cases for each method across all 57 benchmark cases and for 19 of them with the highest degree of uncertainty ($P = 0.15$).

| Set | GA1 | GA2 | UCT | UCT | Ant | TS | Set | GA1 | GA2 | UCT | UCT | Ant | TS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $k' = 4$ | $k' = 4$ | 0.7 | 1.8 | | | | $k' = 4$ | $k' = 4$ | 0.7 | 1.8 | | |
| P-n19 | 244.6 | **230.9** | 246.2 | 244.9 | 281.2 | 262.4 | F-n45 | 740.0 | **733.6** | 775.6 | 774.5 | 761.5 | 775.2 |
| k=2 | 263.4 | **242.4** | 269.6 | 269.3 | 311.8 | 351.2 | k=4 | **774.7** | 801.5 | 828.9 | 830.7 | 831.1 | 847.8 |
| | **324.9** | 410.7 | 338.2 | 340.9 | 391.2 | 373.6 | | 1040.7 | 1504.4 | **1037.2** | 1048.6 | 1138.8 | 1128.9 |
| P-n45 | 652.7 | **575.9** | 604.0 | 601.0 | 607.6 | 622.3 | f-n135 | **1224.9** | 1345.9 | 1285.2 | 1286.3 | 2976.6 | 1295.0 |
| k=5 | 694.9 | 659.0 | 648.6 | **646.8** | 682.0 | 664.3 | k=7 | 1412.6 | 1741.9 | **1403.8** | 1406.2 | 3211.8 | 1434.2 |
| | 875.2 | 1148.4 | 787.7 | **781.8** | 949.7 | 851.0 | | 1886.1 | 3838.5 | **1819.3** | 1872.5 | 3963.7 | 1936.6 |
| E-n51 | 669.7 | **572.8** | 615.4 | 615.6 | 614.1 | 643.0 | F-n72 | 276.3 | 308.0 | **268.7** | 268.8 | 292.3 | 269.2 |
| k=5 | 713.5 | 686.6 | 665.7 | 667.1 | **650.1** | 675.6 | k=5 | 312.6 | 431.1 | 292.0 | **290.6** | 424.2 | 293.9 |
| | 928.7 | 1303.7 | 847.1 | 845.4 | **789.9** | 896.6 | | 427.2 | 917.5 | **372.7** | 376.2 | 537.6 | 396.0 |
| A-n54 | 1253.7 | **1216.1** | 1258.3 | 1254.9 | 1338.7 | 1310.2 | tai-n75a | **1778.5** | **1778.5** | **1778.5** | **1778.5** | 2257.1 | **1778.5** |
| k=7 | **1326.2** | 1333.3 | 1355.0 | 1347.5 | 1456.4 | 1395.5 | k=10 | **1808.7** | 1873.6 | 1843.5 | 1840.5 | 2447.8 | 1899.0 |
| | 1641.9 | 2155.6 | **1634.5** | 1647.6 | 1829.0 | 1790.8 | | 2545.3 | 3337.8 | 2396.6 | **2385.8** | 3236.5 | 2530.0 |
| A-n69 | 1298.6 | **1185.9** | 1263.5 | 1265.2 | 1395.9 | 1332.6 | tai-n75b | 1315.9 | **1293.4** | 1414.1 | 1410.8 | 2025.1 | 1437.0 |
| k=9 | 1394.9 | 1457.5 | 1385.8 | **1377.3** | 1538.1 | 1413.0 | k=10 | **1436.1** | 1499.4 | 1540.6 | 1540.5 | 2209.2 | 1604.5 |
| | 1793.0 | 2672.0 | 1735.5 | **1731.8** | 2096.4 | 1815.5 | | 2152.9 | 2769.2 | 2025.4 | **2023.6** | 2769.6 | 2225.7 |
| E-n76 | 827.7 | 750.3 | 779.3 | 779.0 | **746.0** | 878.3 | vrp_n75 | **527.9** | 543.5 | 577.3 | 575.6 | 627.6 | 580.4 |
| k=7 | 924.6 | 1001.0 | 834.8 | 834.7 | **826.7** | 1059.0 | k=10 | **548.9** | 579.2 | 618.6 | 617.6 | 635.4 | 637.8 |
| | 1186.7 | 2187.4 | 1083.5 | 1085.6 | **1037.2** | 1132.7 | | **687.1** | 936.9 | 788.9 | 783.9 | 898.2 | 842.2 |
| A-n80 | 1953.2 | **1766.1** | 1938.6 | 1929.3 | 1907.1 | 2029.5 | tai-n100a | 2226.8 | **2178.9** | 2267.4 | 2262.9 | 3236.5 | 2282.6 |
| k=10 | 2052.3 | 2038.0 | 2070.0 | 2063.8 | **2003.3** | 2194.4 | k=12 | **2416.4** | 2667.9 | 2423.9 | 2422.4 | 3737.4 | 2501.0 |
| | 2615.1 | 3690.2 | **2578.6** | 2588.4 | 3161.8 | 2641.4 | | 3162.9 | 4904.6 | **3081.5** | 3118.0 | 3390.6 | 3262.8 |
| P-n101 | 899.8 | 1053.5 | **813.7** | 815.0 | 846.8 | 850.9 | tai-n100b | **2140.6** | **2140.6** | **2140.6** | **2140.6** | 2973.0 | **2140.6** |
| k=4 | 1058.3 | 1728.2 | 893.3 | **891.6** | 893.8 | 927.4 | k=12 | 2322.4 | 2590.3 | **2315.3** | 2316.3 | 3155.9 | 2351.7 |
| | 1426.6 | 3711.8 | **1194.7** | 1204.0 | 1375.0 | 1235.5 | | 3124.4 | 4670.4 | **2904.2** | 2931.8 | 3739.6 | 3051.9 |
| C-n150D | 1287.9 | 1281.4 | 1203.1 | **1202.0** | 1297.0 | 1246.7 | vrp_n100 | **441.3** | 465.4 | 499.3 | 496.8 | 469.3 | 502.6 |
| k=12 | 1420.7 | 1937.4 | 1318.8 | **1318.7** | 1392.5 | 1379.9 | k=8 | **486.7** | 576.5 | 536.8 | 532.8 | 538.3 | 563.0 |
| | 1866.2 | 4425.2 | **1730.2** | 1810.2 | 1987.5 | 1866.8 | | **614.5** | 1075.1 | 721.5 | 718.5 | 872.9 | 784.0 |
| Tai-n150b | **2761.3** | 2905.7 | 3016.1 | 3021.5 | 4367.5 | 3172.2 | | | | | | | |
| k=14 | **3150.4** | 4271.7 | 3259.9 | 3270.1 | 4834.3 | 3438.6 | | | | | | | |
| | **4352.4** | 9798.9 | 4388.0 | 4442.5 | 7081.4 | 4594.2 | | | | | | | |
| #Best Overall | 18 | 12 | 15 | 13 | 6 | 2 | #Best P=15 | 4 | 0 | 9 | 4 | 2 | 0 |