Swarm Intelligence in Solving Stochastic Capacitated Vehicle Routing Problem

Jacek Mańdziuk[1,2] and Maciej Świechowski[3]

[1] Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland, mandziuk@mini.pw.edu.pl
[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore, j.mandziuk@ntu.edu.sg
[3] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland, m.swiechowski@ibspan.waw.pl

Abstract. In this paper, the two most popular Swarm Intelligence approaches (Particle Swarm Optimization and Ant Colony Optimization) are compared in the task of solving the Capacitated Vehicle Routing Problem with Traffic Jams (CVRPwTJ). The CVRPwTJ is a highly challenging optimization problem for the following reasons: while the CVRP is already a problem of NP complexity, adding another stochastic layer to its definition (related to stochastic occurrence of traffic jams while traversing the planned vehicle routes) further increases the problem's difficulty by requiring that potential solution methods be capable of on-line adaptation of the routes, in response to changing traffic conditions. The results presented in the paper shed light on the underlying differences between ACO and PSO in terms of their suitability to solving particular instances of CVRPwTJ.

Keywords: Vehicle Routing Problem, Traffic Jams, Particle Swarm Optimization. Ant Colony Optimization, Imperfect Information

1 Introduction

Capacitated Vehicle Routing problem (CVRP) is a popular NP-complete optimization problem which consists in finding the set of routes of a minimum cumulative length (cost) for a given number of trucks that serve a given set of clients. All trucks start from and ultimately return to a pre-defined depot (with a certain 2D location). Each client is defined by its location on a plane and an amount of goods (a demand) to be delivered to them in one shot (a client cannot be served by multiple trucks). Each truck has some pre-defined capacity and all trucks (as well as goods to be delivered) are homogenous. In short, the problem combines the multiple-tour Traveling Salesman Problem with the Bin Packing Problem. For its formal definition please refer, for example, to [10].

There are many approaches rooted in Operation Research or Computational Intelligence which can be applied to solve the CVRP (see, for instance, [10] for their overview).

In this paper, similarly to [9], the baseline problem formulation is extended by adding traffic jams which may occur on the edges (atomic parts of the routes) and therefore increase the cost of their traversal. This extension leads to the Capacitated Vehicle Routing problem with Traffic Jams (CVRPwTJ) specification. Due to highly dynamic nature of CVRPwTJ, the methods used to solving it must be able to swiftly adapt to the on-line changes in the cost function (the cost of currently planned routes) due to frequently changing traffic conditions.

The proposed idea of solving the CVRPwTJ is based on the concept of Swarm Intelligence (SI) which consists in having a population of simple objects that encode solutions to the problem in the search space. These objects iteratively communicate and influence each other, which enables them to modify the encoded solution. Each object has relatively simple rules and goals and the complexity of the system is an emergent feature resulting from maintaining a swarm as a whole. Two such metaheuristic SI methods are employed: Ant Colony Optimization [2] (which uses the notion of *an ant* as the baseline element of the swarm) and Particle Swarm Optimization (PSO) [4] (which refers to *a particle* as an atomic object).

2 ACO in CVRPwTJ

Our implementation of the ACO approach is inspired by a classical algorithm used to solve the Traveling Salesman Problem [2, 3] which was adjusted to take into account the CVRP specificity [1, 11] and furthermore the stochastic nature of the CVRPwTJ stemming from the existence of traffic jams in the problem definition. The approach was initially proposed and described in detail in our previous work [9] devoted to comparison of ACO and the Upper Confidence Bounds Applied to Trees (UCT) method [7]. In this section the ACO-based algorithm presentation is limited to introduction of its main components. For the full coverage and in-depth description of the method please refer to [9].

Assuming that the number of available trucks is equal to k, the initial solution is computed by the modified Clark and Wright (CW) *Savings algorithm* [13] and used to deposit the initial pheromone traits on the initial k routes. Then, in each time step of the algorithm, each ant seeks the solution for the remaining part of the problem (i.e., the complete routes for k vehicles) using the current solution as the starting point (state). Once the solution is found, its quality is evaluated based on the cumulative length of all k routes and the pheromone is deposited on the route's segments.

For each of the k routes, the ant starts its search in the current vehicle's position and looks for the next most suitable customer to be added to the route according to the current pheromone traits and considering the current (stochastic, due to the existence of TJ) cost of traversing particular edges. If the space left on the truck is not sufficient to serve any of the remaining customers or all edges from the current position to the remaining customers are jammed, the truck returns to the depot to accommodate the left customers within a new route.

A pseudo-roulette is used to select the next customer to be visited by an ant. The greedy selection (i.e., of the closest, in terms of dynamic cost, yet not visited client) takes place with probability 0.05. Otherwise, the roulette-wheel method is applied, which selects customer j while being currently at customer i with the following probability:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{ij}(\tau_{ij}^{\alpha} * \eta_{ij}^{\beta})} \qquad \eta_{ij} = (BASE/d_{ij})^2 \tag{1}$$

where τ_{ij} is pheromone amount deposited on edge e_{ij} and d_{ij} is the dynamic (traffic-aware) cost of traversing this edge at the moment. $BASE$ is a normalization factor equal to the length of the initial (static) solution. Coefficients α and β were set to 2 and 3, respectively, based a limited number of preliminary tests.

Once solutions are found by all ants in the current iteration, the pheromone deposit on the edge e_{ij} is incremented in the following way:

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

where D_a denotes a cost of solution $s(a)$ found by ant a, and δ_{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 best overall solution; **20**, if $e_{ij} \in s(a)$ and $s(a)$ is the generally best current solution (among all solutions found by the ants in the current iteration).

The final step in the pheromone update procedure is the evaporation, which is defined at the level of 90% of the previous amount (due to high degree of system's dynamism) and then confined to the predefined interval $[\tau_{min}, \tau_{max}]$ by $Conf_{\tau_{min}}^{\tau_{max}}$:

$$\tau_{ij} := Conf_{\tau_{min}}^{\tau_{max}}(0.1 * \tau_{ij} + \Delta\tau_{ij}) \tag{3}$$

Once the last iteration is completed by all ants in a given time step, the best overall solution is found and used to move the trucks one step ahead (to the next client) according to the schedule represented by this solution. At the beginning of the next step, the best solution is left out, the pheromone traits are reset, new TJ are distributed, and the system proceeds with solving the next step of the problem. Please note, 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 traces from the previous step are misleading. This necessity was fully confirmed in the preliminary simulations.

3 PSO in CVRPwTJ

In this section, the proposed approach to CVRPwTJ with the use of PSO metaheuristic is presented and discussed in detail.

3.1 Problem Encoding

We use one of the standard CVRP encodings presented in the literature [6, 5], in which a population of M particles is maintained, and each particle is encoded as a vector of length N, where N is the number of yet unvisited customers. Each position in the vector is associated with a particular customer's ID. This association, i.e., $positionIndex \leftrightarrow customerID$ is maintained by means of a dictionary (see section 3.2 for the details).

3.2 Initial Population

The modified CW algorithm [13] is used to obtain the initial solution to the static problem, i.e., a set of initial routes. The dictionary, which maps indices of the particle encoding vector to customers' IDs is populated in the following way:

```
int index = 0
for each route R in the initial solution
   for each customer C in R
      dictionary.Add(index, C)
      index = index+1
```

Whenever a customer is visited by a vehicle, the dictionary is updated in a way that it maintains the original order, but uses only indices that are smaller than the number of unvisited customers (i.e., $\{0, \ldots, N-1\}$). More precisely, if a given customer is visited, and consequently should be dropped from the remaining schedule, all subsequent customers are shifted to the left (assuming the schedule is sorted from left to right). The idea is depicted in Figure 1.

In order to induce diversity within the swarm, only 20% of particles are dedicated to encoding the initial solution. The remaining 80% are initialized randomly and afterwards undergo the corrective procedure (c.f. section 3.5) if needed, followed by a local optimization phase by means of 2-OPT algorithm [8].

3.3 Operational scheme of the proposed method

The algorithm solves the problem iteratively, in discrete time steps. A pseudocode of one time step of the method is listed in Algorithm 1.

In each step (which corresponds to atomic movement of vehicles to their next assigned customers) the best solution from the previous step is reset and a series of MAX_{PSO} iterations is executed. For each particle, its velocity and new position are calculated (the details are presented in the next subsection) and then the vehicles' routes are decoded to a vehicle-centric representation that allows immediate analysis of the routes. Particles containing at least one route (one vehicle), which does not obey the maximum capacity constraint, are marked as invalid and undergo a corrective procedure (discussed in details in section 3.5).

In the next step, for each valid particle (either initially or after correction), a 2-OPT local optimization procedure is executed. The best particle, i.e., the one

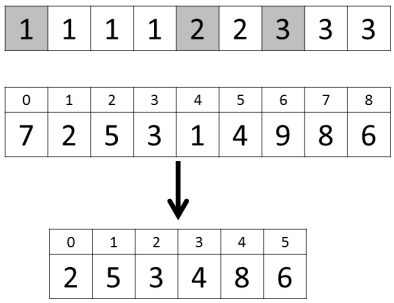

Fig. 1. The top row presents an encoding of a sample solution (3 vehicles and 9 unserved customers) stored in a particle. In the middle part a corresponding sample dictionary state is shown with the customers $(7, 2, \ldots, 8, 6)$ indexed from 0 to 8. The lowest encoding corresponds to the situation after the first customers assigned to vehicles 1, 2 and 3 (i.e., number $7, 1$ and 9) have been visited and removed from the schedule. All the remaining customers are shifted to the left and their corresponding indices are renumbered to $(0, \ldots, 5)$.

having the lowest total cost of routes, is stored. Once the MAX_{PSO} iterations are over, the vehicles move to the next customers based on the encoding kept by the best particle.

3.4 Position and Velocity Update

The position of particle x at step $t + 1$ is updated according to the following equation:

$$x_{t+1}^i = (x_t^i + v_t^i) \bmod K \tag{4}$$

where x_t^i and v_t^i are the ith components of a particle's position at time step t and particle's velocity at time step t, respectively, and K is the number of available vehicles.

Particle's velocity is updated according to the following equation:

$$v_{t+1}^i = inertia * v_t^i + u_{[0;g]}^{(1)}(x_{gBest} - x_t^i) + u_{[0;l]}^{(2)}(x_{lBest} - x_t^i) \tag{5}$$

where *inertia* is a factor that specifies how much of the previous velocity is retained; x_gBest is the global best solution found so far; x_lBest is the local best solution (i.e., the one found by the current particle), g and l are global and local attractors, respectively, and $u_{[0;g]}^{(1)}$ and $u_{[0;l]}^{(2)}$ denote random variables drawn from the uniform distribution bounded by g and l, respectively.

Algorithm 1: A pseudocode of the PSO method. Procedures for *calculating velocity* and *updating position* are described in section 3.4. *Decode routes* and *repair* procedures are discussed in section 3.5.

```
1  set iteration := 0;
2  Reset BestParticle;
3  while iteration < MAX_PSO do
4  |  forall particle in Particles do
5  |  |  calculate velocity of particle;
6  |  |  update position of particle;
7  |  |  decode routes in particle;
8  |  |  if particle is not valid then
9  |  |  |  repair particle;
10 |  |  end
11 |  |  if particle is valid then
12 |  |  |  run 2-OPT for particle;
13 |  |  |  if cost of particle < cost of BestParticle then
14 |  |  |  |  BestParticle := particle;
15 |  |  |  end
16 |  |  end
17 |  end
18 end
```

3.5 Corrective Procedure

The *DecodeRoutes* procedure starts with an empty set of routes and iterates over the *vehicleIDs* stored in the particle encoding vector. For each *vehicleID* it consults the *dictionary* to identify the client associated with the current index and appends them to the route of vehicle *vehicleID*.

The above-described decoding process may create routes for which the total sum of customers' requests exceeds available capacity, in which case the corrective *Repair* procedure is executed. Please note, that the available capacity depends on two factors: the sum of all requests of customers that have already been visited and those who are planned to be visited – the corrective procedure can only influence the not yet visited customers. In the following description of the corrective procedure each vehicle with exceeded capacity will be referred to as ILV (*illegal vehicle*) and the remaining ones as LV (*legal vehicles*).

1) First, ILVs are identified.
2) For each ILV, a minimal number of customers, removal of whom would result in a highest capacity within the allowed limit, is identified. First, the algorithm tries to remove one customer starting from the one with the lowest requested amount. If it finds a legal situation, it stops. If it does not, it will investigate all pairs (again starting from the pair with the lowest cumulative requested amount), etc.

3) The identified sets of customers for each ILV are candidates for transfer to other vehicles. Let us denote such sets of customers as $TSET_i$, where i is a vehicle's identifier.
4) An *outerList* is created, which contains all vehicles, sorted in descending order, by cumulative amount of requests in $TSET_i$. The initially LV have the $TSET_i$ empty, therefore will be placed at the end of the list.
5) An *innerList* is created, which contains all vehicles, sorted in descending order, with respect to their available space. In this step, the customers in $TSET_i$ are ignored.
6) In a double loop possible transfers between two vehicles are investigated. The outer loop iterates over *outerList*, whereas the inner loop iterates over *innerList*. The transfer algorithm is described below. If a transfer leads to a legal situation (both vehicles' capacities are within the limit) it is applied.
7) If there are still vehicles with exceeded capacity, the algorithm will allocate requests from the respective $TSET_i$ to new vehicles. If there are not enough available vehicles to allocate all requests, then the particle is marked as *invalid* in the current iteration of the PSO procedure. Such a particle may potentially be repaired in subsequent iterations.

The transfer algorithm in point 6) above investigates, in a fixed order, three possibilities of exchanging customers between two given vehicles (say k and l). It returns *success* as soon as it finds the first legal situation, i.e., capacities of both vehicles are below the limit.

6a) Cross-pairing: customers from $TSET_k$ and $TSET_l$ are appended to the customers of vehicles l and k, respectively.
6b) After performing the cross-pairing, all customers from a more loaded vehicle (by means of a sum of requests) are one-by-one tried to be transferred to the other vehicle starting from the biggest request.
6c) After performing the cross-pairing, all pairs of customers, one per vehicle, are tried to be exchanged starting from the biggest requests.

4 Experimental Setup

Both methods are directly compared based on a set of widely-known benchmarks taken from the literature (see 4.2 for their exact selection). While the common benchmark instances are static (their definition does not include dynamic elements, such as traffic jams), they are extended to dynamic versions by adding, stochastically distributed, traffic jams. More precisely for each benchmark set, at each time step t a traffic jam of intensity I_t can be imposed on each edge with probability P, independently of other edges.

The following ranges of TJ intensity were tested: $P \in \{0.02; 0.05; 0.15\}$, $I_t = U_{INT}[10, 20]$, $L_t = U_{INT}[2, 5]$, where $U_{INT}[a, b]$ denotes random uniform selection of any integer x such that $a \leq x \leq b$ and $L_t(e)$ denotes a duration of a TJ.

For each of the three values of P and each benchmark set 50 pairwise independent distributions of TJ were samples and used in the experiments. Consequently, for both ACO and PSO we obtained 50 independent results (for pairwise the same sets of TJ distributions) which were subsequently averaged to yield the final score.

4.1 Steering parameters

Both methods are used with the best parameterizations we were able to find. This methods' calibration was performed based on initial tests on 7 benchmarks and 30 trials per benchmark (please recall that the final experimental setup included 19 benchmarks, each tested 50 times).

ACO algorithm was run with a population of $max(100, 2n)$ ants (where n is the size of a benchmark set), for MAX_{ACO} iterations. MAX_{ACO} was set to 200 for benchmarks of size $n < 70$ and to 75 for benchmarks with $n \geq 70$.

PSO method was run for $MAX_{PSO} = 200$ iterations with the number of particles equal to 150 (for all benchmark sizes). The remaining steering parameters in equation (5) were set as follows: $inertia = 0.3$, $l = 0.3$, $g = 0.6$.

The above parameters, for both methods, were selected based the assumed reasonable time allotted for reaching the solution. Clearly, there is still possibility of improvement of results with bigger populations (either of ants or particles), but we believe that the current setup provides a good estimation of the general quality of both approaches, and what is more important, based on the execution times comparison it can be concluded that the selection is fair, i.e., not biased towards any of the two proposed and investigated approaches.

4.2 Benchmark problems

A set of 19 benchmark instances for the static CVRP problem was downloaded from the webpage [12]. Dynamic traffic jams were added to these benchmarks according to the procedure described above in this section. In order to maintain diversity, those instances were chosen from five sets proposed by: Augerat et al. (3 instances of "type A" and 3 of "type P"); Christofides and Eilon (2 instances); Fisher (3 instances); Christofides, Mingozzi and Toth (2); Christofides (1), and Taillard (5). The number of customers in the selected benchmarks varies from 19 to 150 and the number of vehicles (routes) required to construct the initial solution is between 2 and 14. Moreover, the distributions of clients requests' sizes and their locations vary significantly from benchmark to benchmark.

5 Results

The results are presented in Table 1. First of all, a clear advantage of both proposed approaches over the static solution based approach can be observed. While this is an expected result, it is, nevertheless, worth noting that the improvement stemming from application of noise-adaptive methods (ACO, PSO), is quite significant, around 3-4 times, in most of the cases.

Table 1. The average values and standard deviations (in parentheses) across 50 trials. The **Static** column presents application of the initial solution (found at step 0, without any TJ imposed yet) applied to (a dynamic version of) a benchmark set. The best result for each pair (instance, P) are bolded.

P	Instance	Static (σ)	ACO (σ)	PSO (σ)	Instance	Static (σ)	ACO (σ)	PSO (σ)
0.02	P19	388.9 (214.9)	281.2 (46.9)	**251.8 (10.9)**	E76	1318.7 (295.5)	**746.0** (54.4)	764.0 **(32.6)**
0.05	P19	612.0 (213.3)	**311.8** (93.1)	326.1 **(23.1)**	E76	2130.0 (460.4)	**826.7** (159.9)	887.0 **(39.8)**
0.15	P19	1278.0 (358.9)	**391.2** (155.9)	541.3 **(61.3)**	E76	4536.7 (838.0)	**1037.2** (264.3)	1444.9 **(68.9)**
0.02	P45	1007.7 (326.2)	607.6 (53.9)	**590.9 (20.7)**	A80	2774.1 (625.2)	**1907.1** (146.7)	1937.9 **(58.4)**
0.05	P45	1759.6 (411.9)	**682.0** (74.4)	740.9 **(35.7)**	A80	4100.6 (830.1)	**2003.3** (442.6)	2383.4 **(121.9)**
0.15	P45	3299.5 (733.3)	**949.7** (281.7)	998.0 **(54.3)**	A80	9066.5 (1437.4)	**3161.8** (829.6)	3723.5 **(141.7)**
0.02	F45	1515.9 (703.3)	**761.5** (75.8)	771.1 **(33.7)**	Tai100a	3615.6 (849.4)	**2316.3 (227.3)**	3236.5(296.1)
0.05	F45	2078.7 (949.0)	**831.1** (159.3.)	836.4 **(77.5)**	Tai100a	5238.7 (1028.2)	**2875.4 (420.3)**	3390.6 (511.1)
0.15	F45	5060.3 (1519.1)	1138.8 (416.4.)	**1103.6 (186.4)**	Tai100a	11029.6 (1717.2)	4788.9 **(903.9)**	**3737.4** (953.4)
0.02	E51	989.2 (240.6)	**614.1** (40.0)	637.0 **(22.2)**	Tai100b	3425.1 (938.7)	**2339.2 (337.3)**	2973.0 (388.8)
0.05	E51	1571.6 (386.3)	**650.1** (50.4)	778.9 **(40.5)**	Tai100b	5246.1 (1084.6)	3201.0 **(506.4)**	**3155.9** (586.9.)
0.15	E51	3509.7 (824.7)	**789.9** (174.8)	1215.3 **(70.2)**	Tai100b	10660.1 (1713.1)	5141.9 **(940.0)**	**3739.6** (953.4)
0.02	A54	1939.2 (542.7)	1338.7 (84.0)	**1260.5 (41.3)**	chmt100	792.3 (291.5)	469.3 (91.3)	**467.3 (27.2)**
0.05	A54	3072.4 (887.7)	1456.4 (286.0)	**1418.0 (77.9)**	chmt100	1178.7 (409.5)	**538.3** (103.9)	543.4 **(58.4)**
0.15	A54	6275.0 (1441.9)	**1829.0** (519.4)	1901.0 **(139.2)**	chmt100	2471.9 (549.1)	872.9 (319.8)	**730.0 (116.9)**
0.02	A69	2005.7 (531.8)	1395.9 (96.7)	**1297.4 (43.3)**	P101	1436.5 (262.6)	846.8 (69.2)	**809.1 (25.4)**
0.05	A69	3235.4 (644.1)	**1538.1** (294.5)	1668.3 **(103.0)**	P101 5	2552.0 (547.3)	**893.8** (127.3)	931.0 **(38.2)**
0.15	A69	6631.7 (1437.4)	**2096.4** (588.4)	2634.0 **(173.1)**	P101	5419.6 (801.5)	1375.0 (322.2)	**1327.7 (81.9)**
0.02	F72	445.3 (133.1)	292.3 (27.0)	**273.0 (17.2)**	F135	2062.4 (484.0)	2976.6 **(125.3)**	**1301.4** (261.2)
0.05	F72	712.9 (211.4)	424.2 (64.9)	**313.5 (43.3)**	F135	3390.7 (962.8)	3211.8 **(240.0)**	**1532.4** (256.6)
0.15	F72	1502.3 (279.6)	537.6 (119.3)	**455.5 (60.7)**	F135	6945.8 (1410.6)	**1936.6** (654.7)	2141.4 **(423.4)**
0.02	Tai75a	2781.6 (1015.3)	2257.1 **(276.1)**	**1819.0** (280.4)	C150D	1883.0 (368.8)	1297.0 (75.5)	**1202.2 (39.1)**
0.05	Tai75a	4036.8 (1095.0)	2447.8 (415.3)	**2213.0 (403.5)**	C150D	3099.1 (587.5)	**1392.5** (193.3)	1504.0 **(53.5)**
0.15	Tai75a	9281.3 (2008.2)	**3236.5** (873.8)	3918.8 **(830.7)**	C150D	6766.7 (840.1)	**1987.5** (492.4)	2226.6 **(110.6)**
0.02	Tai75b	2494.7 (1145.5)	2025.1 **(146.4)**	**1496.3** (293.9)	Tai150b	4994.7 (1165.5)	4367.5 (515.0)	**2790.4 (100.7)**
0.05	Tai75b	4578.9 (1283.6)	2209.2 (364.0)	**1930.1 (353.0)**	Tai150b	8751.9 (1936.7)	4834.3 (836.4)	**3201.0 (176.3)**
0.15	Tai75b	9108.3 (1752.7)	**2769.6 (777.7)**	3411.8 (804.7)	Tai150b	18104.0 (2581.2)	**7081.4** (1715.2)	7815.7 **(285.5)**
0.02	vrpnc75	817.1 (239.4)	627.6 (154.2)	**584.7 (58.6)**	-			
0.05	vrpnc75	1167.3 (349.9)	**635.4** (156.6)	741.2 **(141.2)**	-			
0.15	vrpnc75	2394.8 (653.4)	**898.2** (438.3)	1166.0 **(250.6)**	-			
Best result count		0 (0)	31 (11)	26 (46)	-			
Best P=2 count		0 (0)	6 (5)	13 (14)	-			
Best P=5 count		0 (0)	12 (3)	7 (16)	-			
Best P=15 count		0 (0)	13 (3)	6 (16)	-			

In a head-to-head comparison of both SI methods there is no clear winner, although ACO seems to be slightly more effective, in general, than PSO. In the summary of best results across all (instance, P) pairs ACO wins 31 cases compared to 26 wins of PSO (and none of Static). When it comes to stability (repeatability) of results the order is reversed: clearly the more stable method (with lower standard deviation) is PSO (46 wins out of 57 cases).

Closer examination reveals that PSO is better suited for the cases with lower amount of noise imposed by traffic jams ($P = 0.02$) with 13/19 of won cases, while ACO is superior for more noisy instances ($P = 0.15$) with exactly the same balance. For the mid-range traffic jams intensity ($P = 0.05$) the advantage is with the ACO approach, albeit, as stated above, in none of the cases is ACO stronger than PSO in terms of results' stability.

6 Conclusions

The paper compares the efficacy of two popular swarm-based methods (Particle Swarm Optimization and Ant Colony Optimization) in solving the Capacitated Vehicle Routing Problem with Traffic Jams. To this end a new approach to CVRPwTJ relying on the PSO algorithm has been proposed and experimentally compared with the ACO-based method proposed by the authors in their previous paper [9]. Experimental results presented in this study lead to the three following conclusions: firstly, the use of swarm-based methods (either ACO or PSO) significantly improves the results compared to static (non-adaptive) approaches; secondly, ACO seem to be slightly superior than PSO (at least in the context of the particular benchmark selection), but at the same time the results yielded by PSO have much lower variance; thirdly, for the cases of relatively low amount of noise (by means of stochastic traffic jams) in the CVRPwTJ instance the preferable method is PSO, while with more dynamic situations (higher amount of noise) the ACO system manifests its upper-hand.

References

1. Bell, J.E., McMullen, P.R.: Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 18(1), 41–48 (2004)
2. Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Politecnico di Milano (1992)
3. Dorigo, M., Gambardella, L.M.: Ant colonies for the travelling salesman problem. BioSystems 43(2), 73–81 (1997)
4. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, Proceedings., IEEE International Conference on. vol. 4, pp. 1942–1948 (1995)
5. Khouadjia, M.R., Talbi, E.G., Jourdan, L., Sarasola, B., Alba, E.: Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. The Journal of Supercomputing 63(3), 836–853 (2013)
6. Khouadjia, M.R., Alba, E., Jourdan, L., Talbi, E.G.: Multi-swarm optimization for dynamic combinatorial problems: a case study on dynamic vehicle routing problem. In: International Conference on Swarm Intelligence. pp. 227–238. Springer (2010)
7. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Proceedings of the 17th European conference on Machine Learning. pp. 282–293. ECML'06, Springer-Verlag, Berlin, Heidelberg (2006)
8. Lin, S.: Computer solutions of the traveling salesman problem. The Bell System Technical Journal 44(10), 2245–2269 (1965)
9. Mańdziuk, J., Świechowski, M.: Simulation-based approach to Vehicle Routing Problem with Traffic Jams. In: 4th IEEE Symposium on Computational Intelligence for Human-like Intelligence. pp. 1–8. IEEE, Athens, Greece (2016)
10. Mańdziuk, J., Żychowski, A.: A memetic approach to vehicle routing problem with dynamic requests. Applied Soft Computing 48, 522–534 (2016)
11. Mazzeo, S., Loiseau, I.: An ant colony algorithm for the capacitated vehicle routing. Electronic Notes in Discrete Mathematics 18, 181–186 (2004)
12. NEO. Networking and Emerging Optmization: (2013), http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/
13. Pichpibul, T., Kawtummachai, R.: An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem. Science Asia pp. 307–318 (2012)