 doi:10.20944/preprints201905.0187.v1

Article

A Self-Adaptive Discrete PSO Algorithm with Heterogeneous Parameter Values for Dynamic TSP

Łukasz Strąk, Rafał Skinderowicz, Urszula Boryczka and Arkadiusz Nowakowski

University of Silesia in Katowice, Institute of Computer Science, Będzińska 39, 41-205 Sosnowiec, Poland
* Correspondence: lukasz.strak@us.edu.pl

This paper presents a discrete particle swarm optimization (DPSO) algorithm with heterogeneous (non-uniform) parameter values for solving the dynamic travelling salesman problem (DTSP). The DTSP can be modelled as a sequence of static sub-problems, each of which is an instance of the TSP. We present a method for automatically setting the values of the DPSO parameters without three parameters, which can be defined based on the size of the problem, the size of the particle swarm, the number of iterations, and the particle neighbourhood size. We show that the diversity of parameter values has a positive effect on the quality of the generated results. We compare the performance of the proposed heterogeneous DPSO with two ant colony optimization (ACO) algorithms. The proposed algorithm outperforms the base DPSO and is competitive with the ACO.

1. Introduction

In recent years, considerable attention has been paid to optimization in a dynamically changing environment in which the problem being solved is modified periodically or even continuously [1]. This interest is related to the growing practical demand for such solutions. For example, in the problem of task scheduling in a factory, a change to the production schedule might be required if there is a malfunction of part of the production line. The optimization algorithms should be able to adapt rapidly to changes so that the quality of the generated solutions remains acceptable. A problem in which the input data are conditional upon time is called a dynamic optimization problem (DOP). The aim of optimization in a DOP is a constant trace and an adaptation to changes, in order to allow high-quality solutions to be found efficiently [2].

A simple example of a DOP is the dynamic travelling salesman problem (DTSP). This consists of a sequence of static travelling salesman problem (TSP) instances (sub-problems). Each successive sub-problem is created on the basis of the previous one. A portion of the sub-problem's data is transferred unchanged from the predecessor, while the remaining portion is modified. Figure 1 summarizes this concept.

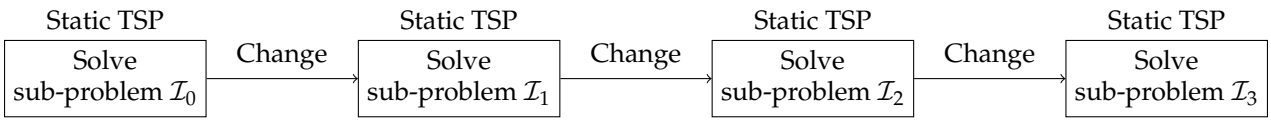

Figure 1. An example of a DOP—the DTSP. A change in the problem instance may affect both the distances between the cities and the number of cities (vertices). This figure presents an example of a DTSP that includes a primary sub-problem ($\mathcal{I}_2$) and three successive sub-problems.

The aim is to find an optimal solution to every sub-problem. If all the sub-problems are solved to optimality, then the resulting sum of the solution (route) lengths is also minimal. Our DTSP benchmark generator includes the optimal value for every sub-problem; hence, the aim can be expressed in terms of finding the minimal sum of the differences between the generated solutions and the optima for the sub-problems. If the value of the sum is zero, then the optimum value is found for every sub-problem.

The particle swarm optimization (PSO) algorithm is an optimization technique created by Kennedy and Eberhart [3] in 1995. This technique is inspired by the natural behaviour of a group of

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

animals, e.g. a shoal of fish or a flock of birds. Every particle represents one of the possible solutions to a problem. In the continuous optimization case, the solution is a point in a real-valued space. The movement of the swarm can be interpreted as searching in the solution space. At the beginning of execution of the algorithm, the position of each particle is chosen randomly. Then, in each iteration of the algorithm, the velocities of the particles are calculated (direction of searching) and their positions are updated. This results in a new solution to the problem. The velocity of the particle is influenced by the best-so-far solution (position) of the swarm and the best-so-far (previous) position of the particle. This process allows the swarm of particles to *learn* and move towards the areas of the problem solution space that contain higher-quality solutions. The movement of the swarm in the solution space is described by the following equations:

$$\begin{aligned} \vec{v}_i^{\,k+1} \leftarrow\ & \omega \cdot \vec{v}_i^{\,k} \\ & + \vec{U}(0,\phi_1) \otimes (\overrightarrow{pBest}_i - \vec{x}_i^{\,k}) \\ & + \vec{U}(0,\phi_2) \otimes (\overrightarrow{gBest} - \vec{x}_i^{\,k}) \,, \end{aligned} \tag{1}$$

$$\vec{x}_i^{\,k+1} \leftarrow \vec{x}_i^{\,k} + \vec{v}_i^{\,k+1} \,, \tag{2}$$

$$\vec{v},\ \vec{x},\ \overrightarrow{gBest},\ \overrightarrow{pBest}_i \in \mathbb{R}^n \,,$$

where i indexes the particles, k is the current iteration, $\vec{v}_i^{\,k}$ is the velocity of the ith particle in the kth iteration, $\vec{x}$ is the position of the particle equal to one of the solutions of the problem, the function $U(0,\phi)$ takes a uniform random value in the range $[0,\phi]$, and ω is an *inertia* parameter. The variables $\overrightarrow{pBest}_i$ and $\overrightarrow{gBest}$ denote the best-so-far solutions found by the particle and by the swarm, respectively, and ϕ_1 and ϕ_2 are cognitive and social parameters, respectively, that scale the influence of $\overrightarrow{pBest}_i$ and $\overrightarrow{gBest}$ on the position of the next particle.

Initially, the algorithm was created for optimization in a continuous space, but was later adapted to discrete optimization. In 1997, Kennedy and Eberhart [4] presented the first discrete PSO (DPSO) algorithm, in which the particle position was a binary vector and the velocity (direction of movement through the solution space) was the probability of a binary negation of the bits of the particle position. In 2004, Clerc [5] proposed a new DPSO algorithm and applied it to solve the TSP. In this algorithm, the particle position was a vector of vertices, while the velocity comprised a list of pairs of vertices, which changed in the next solution. In 2007, Shi et al. [6] presented an improved DPSO algorithm, which they used to solve the following TSP instances: *eil51*, *berlin52*, *st70*, *eil76*, and *pr70*. In their algorithm, the particle position is a permutation, and its modification resembles the well-known crossover found in genetic algorithms. Shi et al. showed that the proposed algorithm is capable of solving the generalized TSP, in which the edge lengths do not satisfy the triangle inequality. The homogeneous and heterogeneous versions of the DPSO algorithm described in the present paper are based on work by Zhong et al. [7] and on our previous DTSP variant [8]. In the implementation presented here, the particle position comprises a set of edges connecting TSP cities (nodes) and the corresponding probabilities of selecting the edges to the next solution (the next position of the particle). As far as we know, our previous work on applying the DPSO to solving the DTSP was the first publication on this topic in the literature.

The original PSO algorithm and its discrete versions are homogeneous, i.e. all particles have the same values of the parameters and hence share the same pattern of moving through a solution space [4]. However, heterogeneous populations are common in the natural environment [9]. One of the most important problems with the PSO concerns the balance between exploration and exploitation. A heterogeneous population allows particles to have various patterns of moving through the solution space and thus to exhibit different levels of emphasis on the exploitation and exploration of the solution space. It is possible that some of the parameter values might turn out to be useful at the beginning of

 doi:10.20944/preprints201905.0187.v1

the algorithm runtime, and others in the later stages. In this way, the balance between exploration and exploitation can be influenced [10,11].

1.1. Contributions

Our previous work has focused mainly on DPSO with *homogeneous* (uniform) parameter values [8]. In this paper, we extend our initial work on DPSO in which individual particles may have non-uniform (varying) values of the parameters [12]. Specifically, our contributions are as follows:

- We propose a method for automatically setting the values of four crucial DPSO parameters. This method is based on discrete probability distributions defined to diversify the behaviours of the particles in the heterogeneous DPSO. The aim of this diversification is to improve the convergence of the algorithm.
- We perform an analysis of the convergence of the proposed algorithm based on computational experiments conducted on a set of DTSP instances of varying sizes. We discuss the relationships between the values of the DPSO parameters and their effect on particle movement through the problem's solution search space.
- We compare the efficiency of the proposed heterogeneous DPSO with that of the base DPSO and two algorithms based on ant colony optimization (ACO). The results show that the proposed algorithm outperforms the base DPSO and is competitive with the ACO-based algorithms.

The structure of this paper is as follows. Section 2 presents a review of the literature concerning the DTSP. Section 3 describes the heterogeneous version of PSO. Section 4 gives a brief description of DPSO with pheromone. Section 5 describes the heterogeneous swarm. Section 6 presents our experimental results. Finally, Section 7 presents a summary and conclusions.

2. Dynamic Travelling Salesman Problem

The dynamic nature of the DTSP can entail changes in the distances between cities (nodes) and in the number of cities to be visited [13,14]. Every data transformation can trigger changes in local and global optima. The distance matrix can be defined as

$$D(t) = \{d_{ij}(t)\}_{n(t)\times n(t)}, \tag{3}$$

where t is time, i and j denote vertices, and n is the number of vertices. Most often, it is assumed that the time is discrete, and hence the DTSP can be viewed as a series of *static* TSP instances (Figure 1). Each sub-problem can be more or less similar to the previous one, depending on the number of changes and their magnitude. In this paper, we assume that only the distances between the cities are subject to change, while the number of nodes (vertices) remains constant.

Obviously, each of the DTSP sub-problems can be solved separately using one of the methods developed for the TSP [15]. Nevertheless, if the differences between consecutive DTSP sub-problems are small, it is possible that the optimal solutions differ only slightly. In such a case, it is possible to use the knowledge gathered while solving the previous sub-problem to speed up solving the current one. A summary of recent research on solving the DTSP that has been presented in the literature is given in Table 1.

Table 1. Summary of recent papers on solving the DTSP with computational intelligence methods.

Year	Authors	Algorithm	DTSP variant
2001	Guntsch and Middendorf [16]	ACO with local and global reset of the pheromone	Addition/removal of vertices
2002	Eyckelhof and Snoek [17]	ACO with various variants of pheromone matrix update to maintain diversity	Changes in edge lengths with time (simulated traffic jam on the road)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

Table 1. Summary of recent papers on solving the DTSP with computational intelligence methods.

Year	Authors	Algorithm	DTSP variant
2006	Li et al. [14]	GSInver-Over and Gene Pool with α-measure [18]	CHN145+1: 145 cities and one satellite
2010	Mavrovouniotis and Yang [19]	ACO with immigrants scheme to increase population diversity	Coefficients: frequency and size of changes
2011	Simões and Costa [20]	CHC algorithm	A test involving addition of changes and their subsequent withdrawal [21]. In that way, the optima at the beginning and the end are the same
2014	Tinós et al. [22]	EA algorithm	Random changes in the problem
2014	Zhang and Zhao [23]	Hopfield neural network	Simulation of various types of real random events in the street
2016	Eaton et al.[24]	ACO with immigrants scheme	Changes in edge lengths. Simulated delays of trains
2016	Mavrovouniotis and Yang [25]	MMAS	Encoding of the problem is changed, but the optimal solution remains the same
2017	Mavrovouniotis et al. [26]	ACO	Distances between cities are changed. The problem can be transformed to an asymmetric one

3. Heterogeneity

Heterogeneity can be defined as the absence of uniformity (diversity). In computational intelligence algorithms, it can appear in many ways. A taxonomy of the various levels of heterogeneity that are possible in the PSO algorithm was given by Montes de Oca et al. [11], who divided heterogeneity into the following four categories:

1. *Neighbourhood heterogeneity*: this concerns cases in which the size of the neighbourhood is different for every particle, and hence the virtual topology of connections between particles is not regular. Some particles can have a wider influence than others on the movement of the swarm.
2. *Best-particle heterogeneity*: here there can be variations in the method of selecting the best particle, i.e. the particle whose position is used when updating the current velocity and position. For instance, one particle might update its position following the best particle in its (small) neighbourhood, while the second particle might be fully informed and follow the global best particle.
3. *Heterogeneity of the position update strategy*: here the particles differ in their patterns of movement (searching) through the solution space. For example, one group of particles might *explore* the solution space, while the other group might conduct a local search by restricting their velocities or even positions to a certain range. This type of heterogeneity diversifies the population to the greatest extent, since it provides the greatest flexibility in diversifying particle movement.
4. *Heterogeneity of parameter values*: here each particle or group of particles in the swarm can have different values of the parameters. For example, some particles might have a large inertia ω and explore the solution space, whereas other particles might have a small value of ω and perform the search locally (around the best position found). Although this type of heterogeneity is not as flexible as heterogeneity of the position update strategy, it requires relatively few changes to the PSO, since only the values of the particle parameters need be set individually. It is this strategy that we apply in the proposed heterogeneous DPSO algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

Although there is a lack of information in the literature with regard to heterogeneity in the case of the DPSO algorithm, it is possible to adapt the solutions proposed for standard PSO.

4. DPSO with Pheromone

A (homogeneous) DPSO algorithm with pheromone was proposed in our previous work [8], and this section contains only a brief description. Adaptation to a discrete space forces some changes to the original PSO algorithm designed for solving continuous optimization problems. All variables (i.e. X and V) become *sets* of edges instead of real-valued vectors. An edge is represented by a tuple: $\langle p, \{a, b\}\rangle$, where a and b are endpoints and p is the probability of selecting the edge (a, b) to become part of the constructed solution. The equations governing the movement of the particles become

$$\begin{aligned} V_i^{k+1} = {} & c_2 \cdot U(0,1) \cdot (gBest \setminus X_i^k) \\ & \cup\, c_1 \cdot U(0,1) \cdot (pBest_i \setminus X_i^k) \\ & \cup\, \omega \cdot V_i^k \,, \end{aligned} \tag{4}$$

$$X_i^{k+1} = \Delta\tau^k(V_i^{k+1}) \oplus c_3 \cdot U(0,1) \cdot X_i^k \,, \tag{5}$$

where i is the particle index, k is the iteration, and $U(0,1)$ is a uniform random number from the range $[0,1]$. The operators $\cup$ and $\setminus$ denote the classical operations on sets, while the multiplication of a set by a scalar (i.e. $c_2 \cdot U(0,1) \cdot (gBest \setminus X_i^k)$) represents multiplication of the p value of each edge by the scalar. The $\oplus$ operator does not exist in classical PSO—its purpose in DPSO is to complete the solution with missing edges so that it forms a Hamiltonian cycle. The $\Delta\tau$ function changes the probability p of the edge using the pheromone matrix familiar from ACO. The pheromone has two main functions in the algorithm:

1. It alters the probability of edge selection during the solution construction process; i.e. the higher the value of the pheromone, the greater is the probability of selecting the corresponding edge. In other words, the pheromone serves as an additional memory of the swarm, allowing it to learn the structure of high-quality solutions and, potentially, improve the convergence of the algorithm.
2. The pheromone matrix created while solving the current DTSP sub-problem is retained and used when solving the next sub-problem. This allows knowledge about the previous solution search space to be transferred with the aim of helping the construction of high-quality solutions to the current sub-problem. This implicitly assumes that the changes between consecutive sub-problems are not very great, so that the high-quality solutions to the current sub-problem share most of their structure with the high-quality solutions to the previous one.

For example, let $\mathcal{G}_u$ be the undirected graph defined as follows:

$$\mathcal{V}_u = \{1,2,3,4,5,6\}\,, \qquad |\mathcal{E}_u| = \binom{n}{2}$$

(all two-pairs combinations of the set $\mathcal{G}_u$). Let the first particle in the first iteration represent the solution

$$\begin{aligned} X_0^1 &= \{\langle 1,\{1,4\}\rangle, \langle 1,\{4,2\}\rangle, \langle 1,\{2,5\}\rangle, \langle 1,\{5,3\}\rangle, \langle 1,\{3,6\}\rangle, \langle 1,\{6,1\}\rangle\}\,, \\ V_0^1 &= \{\langle 1,\{2,5\}\rangle\}, \\ gBest &= \{\langle 1,\{1,2\}\rangle, \langle 1,\{2,3\}\rangle, \langle 1,\{3,4\}\rangle, \langle 1,\{4,5\}\rangle, \langle 1,\{5,6\}\rangle, \langle 1,\{6,1\}\rangle\}\,, \\ pBest_0 &= \{\langle 1,\{1,2\}\rangle, \langle 1,\{2,3\}\rangle, \langle 1,\{3,5\}\rangle, \langle 1,\{5,4\}\rangle, \langle 1,\{4,6\}\rangle, \langle 1,\{6,1\}\rangle\}\,. \end{aligned}$$

The result of applying Eq. (4) is

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

$$gBest \setminus X_0^1 = \{\langle 1,\{1,2\}\rangle, \langle 1,\{2,3\}\rangle, \langle 1,\{3,4\}\rangle, \langle 1,\{4,5\}\rangle, \langle 1,\{5,6\}\rangle\},$$
$$pBest_0 \setminus X_0^1 = \{\langle 1,\{1,2\}\rangle, \langle 1,\{2,3\}\rangle, \langle 1,\{5,4\}\rangle, \langle 1,\{4,6\}\rangle\}.$$

The next velocity of the particle V_0^2 after the operation of multiplication by $c_1 \cdot rand()$, $c_2 \cdot rand()$, or $\omega \cdot rand()$ is

$$(gBest \setminus X_0^1) \cup (pBest_i \setminus X_0^1) \cup V_0^1 = \{\langle 0.3,\{1,2\}\rangle, \langle 0.1,\{2,3\}\rangle, \langle 0.5,\{3,4\}\rangle, \langle 0.6,\{4,5\}\rangle, \langle 0.1,\{5,6\}\rangle, \langle 0.2,\{1,2\}\rangle, \langle 0.9,\{2,3\}\rangle, \langle 0.7,\{5,4\}\rangle, \langle 0.4,\{4,6\}\rangle\}.$$

The edge from the previous velocity is not added to the sum, because of the rule forbidding any vertex (node) to occur more than four times ($\deg(2) = 5$) [7]. Let us assume that pheromone reinforcement is equal to zero (no influence) and that the random function returns the values 0.1, 0.7, 0.49, 0.5, 0.9, 0.3, 0.6, 0.55, 0.39. Then, after the filtration stage, the (incomplete) particle position set is

$$X_0^2 = \{\langle 0.3,\{1,2\}\rangle, \langle 0.5,\{3,4\}\rangle, \langle 0.6,\{4,5\}\rangle, \langle 0.9,\{2,3\}\rangle, \langle 0.7,\{5,4\}\rangle, \langle 0.4,\{4,6\}\rangle\}.$$

The next stage is more restrictive. Any edge that creates an incorrect tour is removed from the set. The edge $\langle 0.4,\{4,6\}\rangle$ is rejected, because $\deg(4) = 3$. The edge $\langle 0.7,\{5,4\}\rangle$ is also rejected, because the edge with $\{4,5\}$ endpoints already exists in the next position. The next incomplete particle position is

$$X_0^2 = \{\langle 0.3,\{1,2\}\rangle, \langle 0.5,\{3,4\}\rangle, \langle 0.6,\{4,5\}\rangle, \langle 0.9,\{2,3\}\rangle\}.$$

At this stage, the first part of Eq. (5) is completed. The operation $\oplus$ adds to the result the edge (X_0^2) $\langle 1,\{6,1\}\rangle$ chosen from the previous particle position set. To complete the set to form the Hamiltonian cycle, the nearest-neighbour heuristic is used and the edge $\langle 1,\{5,6\}\rangle$ is selected. The final particle position is

$$X_0^2 = \{\langle 1,\{1,2\}\rangle, \langle 1,\{2,3\}\rangle, \langle 1,\{3,4\}\rangle, \langle 1,\{4,5\}\rangle, \langle 1,\{5,6\}\rangle, \langle 1,\{6,1\}\rangle\}.$$

Figure 2 presents a visualization of all the primary operations, i.e. the edges from the particle's previous position X_i^{k-1} before the filtration (a), after the filtration (b), and the final particle position (c). The dashed line marks the edge from Eq. (5) and the dotted line the edge from the completion process (c).

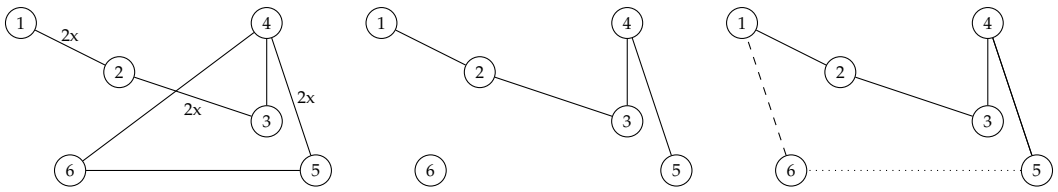

Figure 2. Example of calculation of a new particle position in the DPSO.

5. Heterogeneous Swarm

The DPSO has four main parameters that influence particle movement through the solution search space: c_1, c_2, c_3, and ω. To better understand how the parameter values are set in the proposed *heterogeneous* DPSO, it is helpful to focus on how the parameters govern the swarm behaviour. Zhong et al. [7] suggested the following ranges of values for the parameters: $c_1 \in [0, 1.5]$, $c_2, c_3 \in [0, 2]$,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

$\omega \in [0, 0.6]$. Setting the parameters to small values, i.e. close to the start of the range, forces the particles to change their positions (edges) frequently, since the probability of selecting the edges from the current best positions (local and global) is relatively small. Also, in the initial stage of execution of the algorithm, the pheromone values cannot guide the construction process, since they are also small. On the other hand, setting the parameters to higher values forces the solution construction process to become more exploitative, since the constructed solutions resemble the previously obtained high-quality solutions. Based on our earlier studies of the DPSO algorithm, we have selected the *characteristic sets* of the parameter values, which are shown in Table 2. For each set, we provide a brief description of the corresponding DPSO particle behaviour. Below, we present a more detailed description of the sets, supported by some experimental data analysis.

Table 2. Characteristic sets of particle parameter values for the DPSO algorithm along with their influence on particle movement.

No	c_1	c_2	c_3	ω	Description
1	0.1	0.1	0.1	0.1	Favours quick changes of position
2	2.0	0.1	0.1	0.1	Emphasis on the information from *pBest*
3	0.1	2.0	0.1	0.1	Emphasis on the information from *gBest*
4	0.1	0.1	2.0	0.5	Very slow changes of position
5	0.75	1.0	1.0	0.25	Weak *pBest*, *gBest* influence
6	1.25	1.5	1.5	0.5	Stronger *pBest*, *gBest* influence
7	1.5	2.0	2.0	0.5	Strong *pBest*, *gBest* influence
8	1.75	2.0	2.0	0.75	Very strong *pBest*, *gBest* influence

Figure 3 presents the numbers of new edges for X^{k-1} and X^k (the previous and current positions). The blue line indicates the particle parameter values, which often change edges (setting 1), and the red line is for more stable particles, with less frequent changes (setting 4).

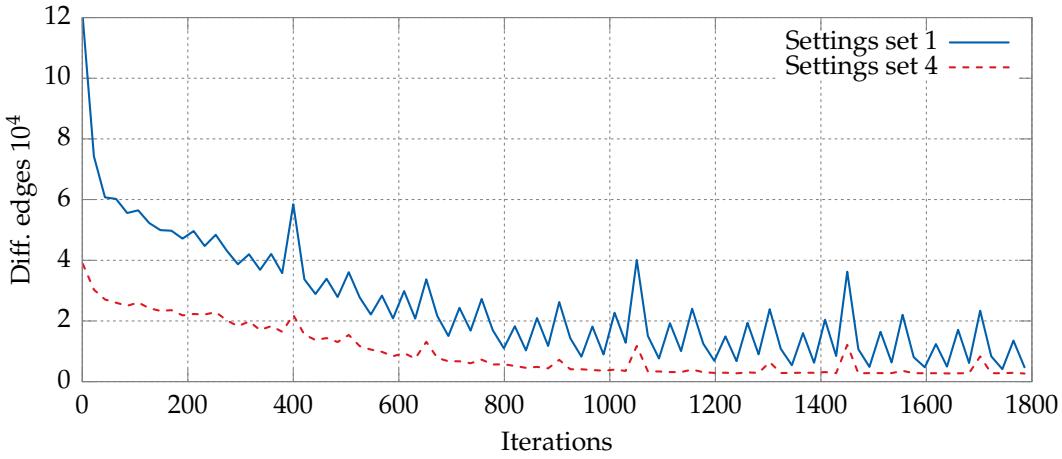

Figure 3. Numbers of new (different) edges between X^{k-1} and X^k (the previous and current positions) in the DPSO solving the static *kroA200* TSP instance. The blue line indicates the particles with the first set of values from Table 2 (setting 1) and the red line is for the more "stable" particles for which the fourth set (setting 4) of parameter values were used. The remaining parameters were taken from Table 5. The values were averaged over 30 runs of the algorithm.

The first and fourth sets of parameter values from Table 2 differ in terms of the dynamics of changes in the number of common edges between the current and previous positions of the particle. For the small parameter values taken from the first set, the probability of edge selection to the next position (p) is very small and can only be increased if the corresponding pheromone has a high value. On the other hand, in the fourth set of parameters, c_3 has the highest value from the range. As a

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

result, the edges from the previous position will be added to the next position of the particle with high probability. Both characteristics can be clearly seen in Figure 3. The blue line is below the red one, which means that the position of the particle from the first set has more changed edges.

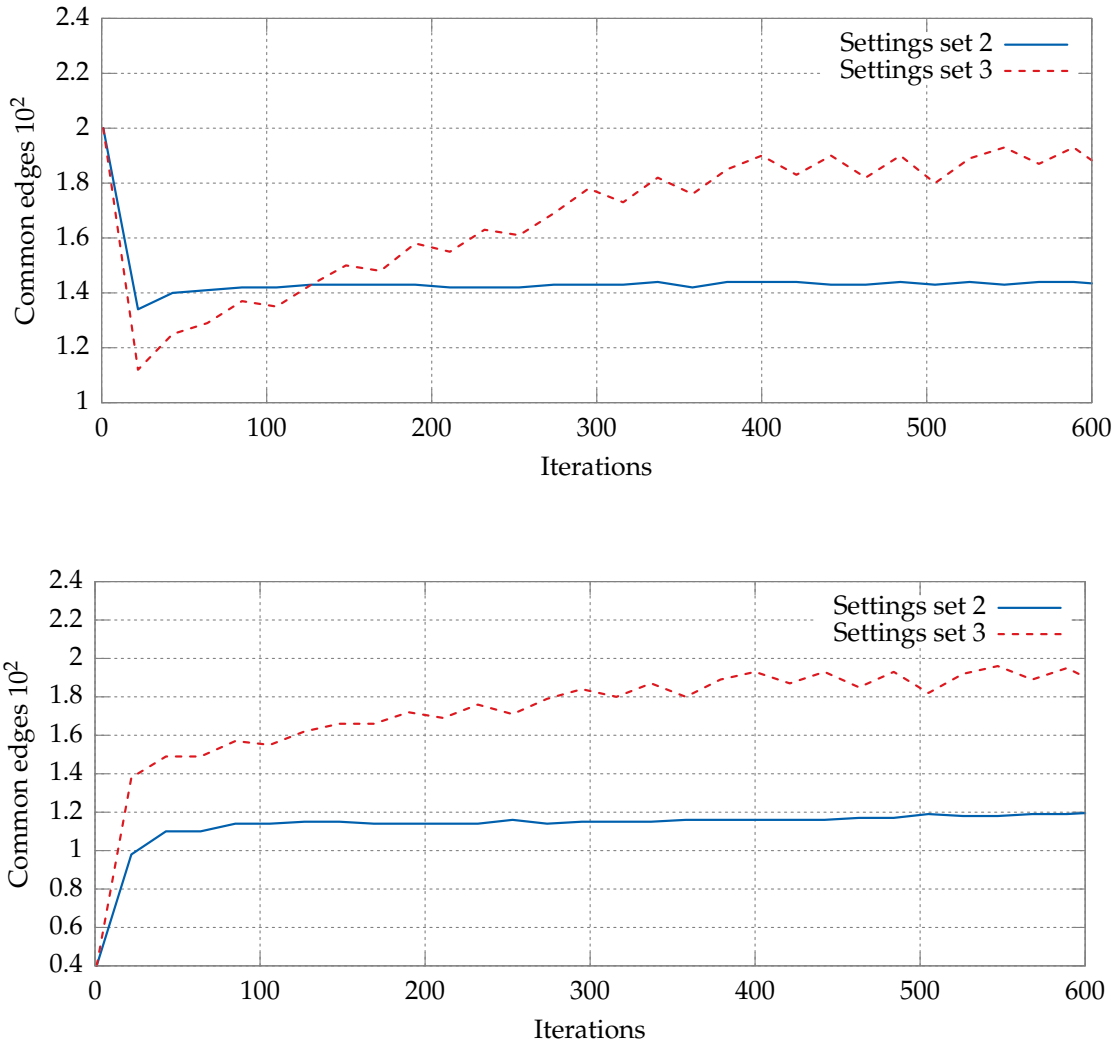

Figure 4. Numbers of common edges between X^k and $pBest$ (top) and X^k and $gBest$ (bottom) for the second and third sets of characteristic parameter values (Table 2). (The DPSO algorithm was run for the static *kroA200* TSP instance.) The remaining parameters were taken from Table 5. The values were averaged over 30 runs of the algorithm.

An analogous comparison can be made for the second and third sets of values shown in Table 2. Figure 4 shows the average numbers of common edges between the current position of a particle, X^k, and the best positions, i.e. the particle's local best $pBest$ and the swarm best $gBest$. For the second set of parameter values, the number of edges shared with $pBest$ was higher than for the third set. This was caused by the high c_1 value, equal to 2, which affected in particular the initial iterations of the algorithm. After the first 100 iterations, the number begins to change as $pBest$ and $gBest$ become more similar. This is an effect of the high value of the c_2 parameter in the third set of parameter values. The bottom plot in Figure 4 shows the average number of common edges for the sets X^k and $gBest$. We can see a growing similarity of the current position X_k to the current best position $pBest$. This effect can be observed for both sets of parameter values. The number of common edges is higher for the third set, since it has the highest possible value of c_1.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

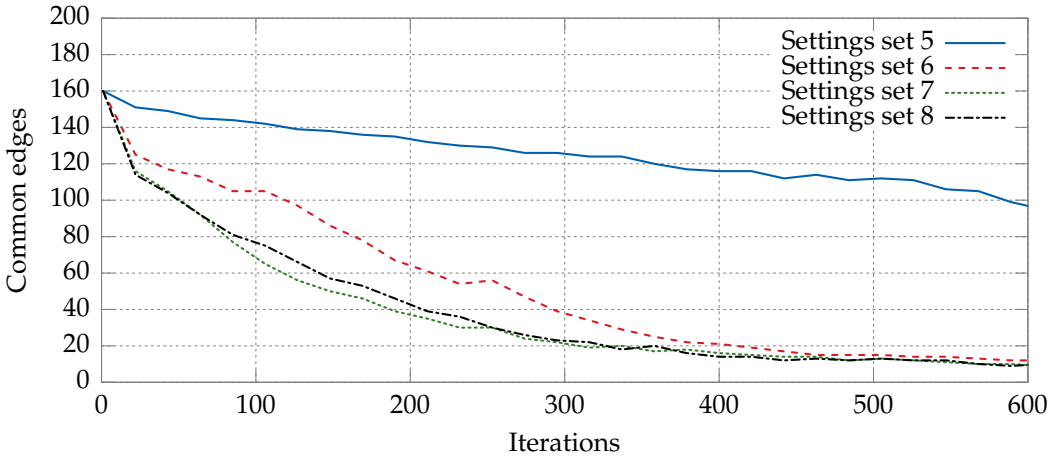

Figure 5. Total numbers of common edges between the current position of the particle, X^k, and $pBest$, and between X^k and $gBest$ for the DPSO solving the *kroA200* TSP instance with the parameters given by sets 5–8 in Table 2. The remaining parameters were taken from Table 5. The values were averaged over 30 runs of the algorithm.

An analogous comparison, this time for sets 5–8 from Table 2, is presented in Figure 5. The largest differences can be observed for the fifth and the sixth sets, and the smallest for the seventh and eighth. This is due mainly to the small differences between the parameter values, namely $\Delta c_1 = 0.25$ and $\Delta\omega = 0.25$ (the remaining parameters c_2 and c_3 have the same value).

Based on the number of times each value of a parameter appears in Table 2, a discrete probability distribution for the parameters can be defined:

1. c_1: $P(0.1) = 0.4$, $P(0.75)$=0.15, $P(1.5)$=0.3, $P(1.75)$=0.15;
2. c_2 and c_3: $P(0.1)$=0.4, $P(1)$=0.15, $P(1.5)$=0.15, $P(2)$=0.3;
3. ω: $P(0.1)$=0.4, $P(0.25)$=0.2, $P(0.5)$=0.4.

This allows the values of the DPSO parameters to be controlled, while also allowing them to be mixed together; i.e. any combination of the listed values is possible. As a result, we can expect that both the exploration- and exploitation-oriented behaviours of the particles will be present in a swarm, hence increasing the chances of finding high-quality solutions regardless of the "landscape" of the solution space. This also has the advantage of being more computationally efficient compared with a completely random setting (e.g. with uniform probability), since, in the latter case, one would need a larger number of particles to observe a similar mix of characteristic particle behaviours.

6. Experimental Results

This section is divided into two parts. In the first, we focus on the effect of the parameter values on the performance of individual particles in the heterogeneous DPSO algorithm. In the second, we conduct a comparison between the homogeneous DPSO, the proposed heterogeneous DPSO, and two well-known ACO algorithms, namely the ant colony system (ACS) and population-based ACO (PACO).

6.1. Convergence Analysis for Various Sets of Parameters

To assess the performance of individual particles in a swarm of the heterogeneous DPSO, we counted the number of times the particle improved the current global best solution $gBest$. The parameter values were set randomly according to the discrete probability distribution described in Sec. 5. The *gr666* TSP instance was used as a test bed.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

Table 3 shows the sets of parameter values for which the particles were able to improve the global best solution most frequently. As can be seen, the top two are the sets in which the parameters c_1, c_2, c_3, and ω are relatively small. These values favour exploratory behaviour of the DPSO particles, and hence the particles are more likely to find an improved solution, especially in the initial phases of algorithm execution. The set for which the behaviour should be more stable and less exploratory, i.e. with $c_2 = 2$, turned up as third in the ranking. The relatively large difference of 53 between the second and third positions is also noteworthy. The lower rankings of the particles exhibiting more exploitative behaviour confirm that they could be more important in the later stages of algorithm execution, in which smaller changes to the solution structure are preferred.

Table 3. Ranking of parameter values for which the particles in the heterogeneous DPSO were able to improve the global best solution the greatest number of times. The results are accumulated over 30 executions for the *gr666* TSP instance.

Rank	Parameters				Number of *gBest* improvements
	c_1	c_2	c_3	ω	
1	0.1	0.1	0.1	0.5	113
2	0.1	0.1	0.1	0.1	102
3	0.1	2	0.1	0.1	49
4	0.1	2	2	0.5	46
5	0.1	2	2	0.1	42
6	0.1	1.5	0.1	0.5	39
7	0.1	1	0.1	0.1	38
8	0.1	1	0.1	0.25	34
9	0.75	2	2	0.25	27
10	0.1	1	2	0.1	26
11	0.1	2	0.1	0.25	24
12	0.75	2	2	0.1	22
13	1.5	1.5	2	0.5	21
14	1.5	2	0.1	0.1	21
15	1.5	2	0.1	0.25	20

To clarify this distinction, we have analysed which values of the parameters proved to be working best during subsequent phases of algorithm execution. The phases were defined by dividing the total number of iterations into equal parts (intervals). For each interval, we ranked the sets of parameter values based on the number of times they led to a new global best solution within the respective interval. Table 4 presents the results, while Figure 6 shows the speed of convergence towards an optimum in each phase. As can be seen, different sets of parameter values dominate subsequent phases (intervals) of the computations. In the first interval (0–1250), the sets with small parameter values are predominant—which indicates that rapid changes in the particle solutions are beneficial. In the third interval (2500–3750), the sets of parameter values are mixed, i.e. they contain both small and high values. This can be interpreted as a sign that the exploration of the solution space slows down and, more importantly, becomes exploitation. In the last interval (5000–6144), the best particles have relatively high parameter values, which, combined with stronger pheromone reinforcement, causes mainly small changes to the particle positions.

6.2. Comparative Study

To evaluate the performance of the proposed DPSO algorithm, we have compared it with the homogeneous version of the DPSO and with ACS and PACO, which are among the best-performing metaheuristics for the TSP and DTSP problems. The DTSP test instances were generated based on the static TSP instances from the well-known TSPLIB repository. The test data can be found in a public

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

Table 4. Ranking of parameter values for which the particles in the heterogeneous DPSO were able to improve the global best solution the greatest number of times within four designed subsequent phases of the computations. The results are accumulated over 30 executions for the *gr666* TSP instance.

Iterations	Parameters				Number of *gBest* improvements
	c_1	c_2	c_3	ω	
0–1250	0.1	0.1	0.1	0.1	94
	0.1	0.1	0.1	0.5	93
	0.1	2	2	0.5	38
	0.1	2	0.1	0.1	38
	0.1	2	2	0.1	32
1250–2500	0.75	2	2	0.25	12
	1.5	2	0.1	0.1	10
	0.1	2	0.1	0.1	10
	0.1	1.5	0.1	0.5	9
	0.1	1	2	0.1	8
2500–3750	0 .1	0.1	0.1	0.5	10
	1.5	1.5	2	0.5	4
	1.5	2	0.1	0.25	4
	0.75	2	2	0.25	3
	0.1	1	2	0.1	3
3750–5000	0.1	1	0.1	0.1	2
	0.1	1	2	0.1	2
	0.75	0.1	2	0.5	2
	1.5	0.1	2	0.1	2
	1.5	2	1.5	0.1	2
5000–6144	1.75	2	1	0.5	3
	1.5	2	1.5	0.1	2
	1.75	0.1	2	0.5	2
	0.1	1.5	0.1	0.5	2
	1.5	1	1	0.1	1

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

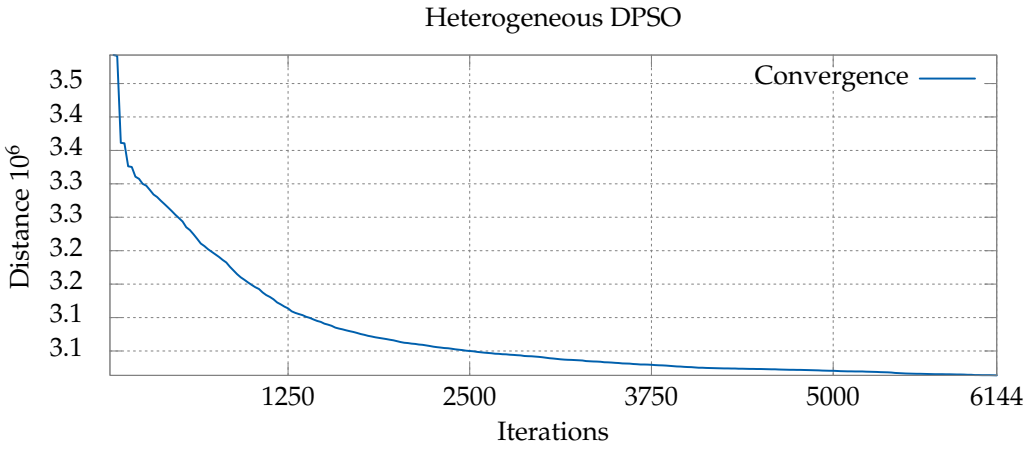

Figure 6. Chart showing convergence with the optimum of the heterogeneous version of the algorithm for the *gr666* problem.

repository.[1] Algorithm 1 presents an outline of the general test procedure used to solve the DPSO with the algorithms mentioned.

Algorithm 1 Outline of the procedure for solving the DTSP.

```
Load the static TSP instance        ▷ The original TSP instance becomes the first DTSP sub-problem
Initialize the algorithm-related data
while Stop criterion is not met do
    sub-problem-related initialization        ▷ Create swarm and neighbourhood etc.
    Solve the current sub-problem             ▷ Solve with DPSO, ACO, etc.
    Modify the current sub-problem to obtain the next one
end while
```

To make the comparison fair, all algorithms were solving the same DTSP instances, i.e. starting from the same static TSP and including the same DTSP-related changes to the positions of the cities. Each DTSP instance comprised 11 static TSP sub-problems, namely the original problem from TSPLIB and ten sub-problems resulting from random changes to the position of the cities. The *gr666* problem was an exception, since it included only one sub-problem (the original TSPLIB problem). Figure 7 shows an example of a DTSP instance consisting of two static TSP sub-problems.

Table 5 shows the parameter values of the two DPSO variants. The numbers of iterations used are shown alongside the results in Table 6. The size of the swarm and the size of the particle neighbourhood were determined from preliminary computations, keeping in mind that both parameters strongly influence the computation time and the quality of the solutions. A smaller neighbourhood limits the solution space and speeds up computation. However, too low a value could hamper finding the optimum. The parameters (c_1,c_2, c_3, ω, *SwarmSize*, and *neighbourhood*) for the *homogeneous* version of the DPSO were chosen based on preliminary computations and our previous work on DPSO.

The ACS and PACO parameters were set as follows: number of ants = 10; number of iterations = $\lfloor 0.1 \cdot p_{ev} \rfloor$; $\beta = 3$; local and global pheromone evaporation coefficients $\alpha = 0.1$ and $\rho = 0.1$, respectively; and $q_0 = (n - 10)/n$, where n is the size of the problem. For the PACO algorithm, $q_0 = 0.8$ was used and the *age-based* strategy for updating the solution archive (of size 5) was used. The values of the parameters were set based on preliminary computations and the suggestions by Cáceres el al. [27], in which the ACO was tested with a small computation budget.

[1] https://github.com/lukaszstrak/DTSP-repository

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

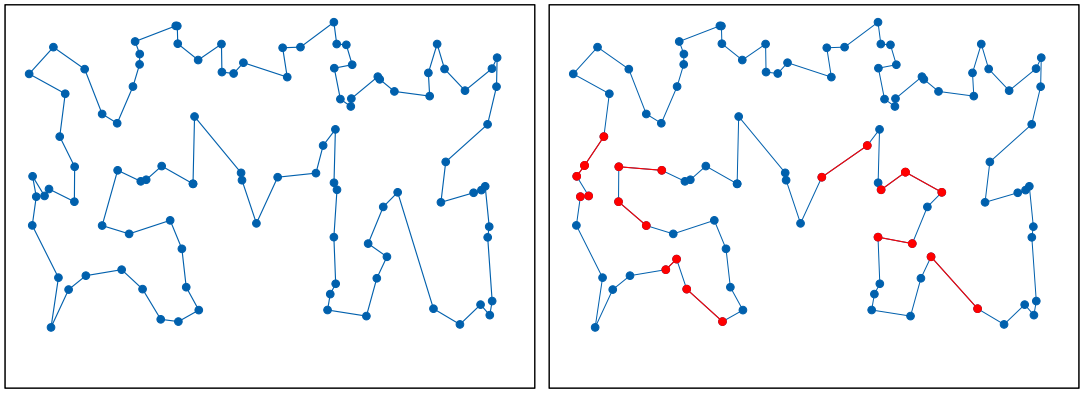

Figure 7. Visualization of the optimum routes for the static *kroA100* TSP instance (left side) and the DTSP instance after a random relocation of some cities (right side). The edges differentiating the new optimum from the previous are marked in red.

All the considered algorithms, including DPSO and ACO, were allowed to construct and evaluate exactly the same number of solutions (p_{ev}) to a problem. For example, the DPSO algorithm with 104 iterations and a swarm of size 32 constructed a total of $p_{ev} = 104 \cdot 32 = 3328$ solutions. All algorithms were implemented in the C# language and run on a computer with an Intel i7 3.2 GHz CPU. All computations were repeated 30 times and the results were averaged.

Table 5. Values of DPSO-related parameters.

Homogeneous DPSO					Heterogeneous DPSO					Common parameters	
Problem	c_1	c_2	c_3	ω	Problem	c_1	c_2	c_3	ω	*SwarmSize*	Neighbourhood
berlin52	0.5	0.5	0.5	0.2	*berlin52*, *kroA100*, *kroA200*, *gr202*, *gr666*	Chosen randomly as described in Sec. 5				32	7
kroA100	0.5	0.5	0.5	0.5						64	7
kroA200	0.5	0.5	0.5	0.5						80	7
gr202	0.5	0.5	0.5	0.5						101	10
gr666	0.5	1.0	1.5	0.6						112	30

For the smallest DTSP instance (*berlin52*), both DPSO versions generated results that were of similar quality and, at the same time, better than those of the ACO algorithms. For the larger instances, the heterogeneous DPSO shows a clear advantage over the homogeneous DPSO. The biggest differences were observed for the *pcb442* and *gr202* instances, for which the heterogeneous version generated higher-quality solutions, especially if the number of iterations was low. This confirms that the heterogeneity of the parameter values results in a broader exploration of the solution search space. At the same time, the heterogeneous DPSO is also more consistent in finding high-quality solutions, which is manifested in the smaller average standard deviation compared with the homogeneous version. When the number of iterations grows, the advantage of the heterogeneous DPSO becomes less—confirming that, in the later stages of the computations, the exploitative nature of the algorithm becomes more important. Generally, both DPSO versions benefit from a larger number of iterations. Compared with the ACO algorithms, the DPSO variants converge more rapidly and the increasing number of iterations allows them to outperform ACS and PACO in almost all cases.

7. Conclusions

We have proposed a heterogeneous DPSO algorithm for solving the DTSP. In this algorithm, each particle can have different values of the crucial DPSO parameters c_1, c_2, c_3, and ω. These values are chosen randomly according to the discrete probability distribution defined so that different behaviours of the DPSO particles can be obtained. Computational experiments conducted on a set of DTSP instances have shown that it is beneficial if some particles explore the solution space while others are more exploitative, i.e. narrow their search by constructing solutions similar to the high-quality solutions

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 May 2019 doi:10.20944/preprints201905.0187.v1

Table 6. Comparison of results for the homo- and heterogeneous DPSO variants and the ACO algorithms obtained for four DTSP (*berlin52*, . . ., *pcb442*) and one TSP (*gr666*) instances. "G" denotes the distance to the optimum and "D" the average standard deviation of this distance. The numbers of iterations are given per sub-problem. The best solutions found by the DPSO algorithms are marked in boldface.

Problem	Iterations	DPSO algorithms						Counterparts	
		Homogeneous			Heterogeneous			ACS	PACO
		T [s]	G [%]	D [%]	T [s]	G [%]	D [%]	G [%]	G [%]
berlin52	104	0.13	0.15	0.32	0.13	**0.13**	0.15	0.96	0.96
berlin52	416	0.3	0.01	0.04	0.28	0.01	0.05	0.5	0.5
berlin52	1664	0.98	**0**	0	0.89	0.01	0.05	0.46	0.46
kroA100	100	1.03	5.44	2.47	0.86	**2.68**	1.4	1.8	2.97
kroA100	400	1.63	1.28	1.02	1.27	**1.05**	0.81	1.31	2.13
kroA100	1600	4.11	**0.64**	0.69	3.38	0.78	0.77	0.82	1.36
kroA200	160	2.49	15.63	2.77	2.18	**5.14**	1.84	2.41	3.33
kroA200	640	5.13	4.45	1.62	4.46	**2.89**	1.09	1.62	2.71
kroA200	2560	15.6	**1.62**	0.81	13.18	2.02	0.8	1.47	2.28
gr202	128	8.82	13.75	2.06	8.17	**4.19**	1.2	6.26	4.91
gr202	512	11.54	6.81	2.11	10.88	**1.97**	0.66	4.88	3.9
gr202	2048	23.01	**1.52**	0.6	21.98	1.53	0.55	3.93	3.34
pcb442	272	11.22	29.31	5.33	11.16	**6.73**	1.68	6.18	4.44
pcb442	1088	28.52	13.41	5	30.69	**2.87**	0.89	4.87	3.56
pcb442	4352	102.78	3.13	1.52	108.25	**1.92**	0.79	3.91	3.3
gr666	384	85.19	10.84	1.52	91.83	**9.58**	0.86	9.18	5.89
gr666	768	98.36	7.37	1.0	115.19	**6.88**	0.78	7.46	4.77
gr666	1536	124.84	5.62	0.84	163.48	**5.33**	0.57	6.09	4.51
gr666	3072	180.66	4.88	0.63	259	**4.52**	0.88	5.67	4.14
gr666	6144	296.83	3.99	0.77	453.83	**3.8**	0.78	4.92	4.21

found so far. The heterogeneous DPSO algorithm improves the quality of the results obtained compared with the homogeneous version. Moreover, the algorithm is easier to use, since fewer parameters have to be set manually, which is important because choosing the right values of the parameters can be especially difficult for the DTSP. It is also worth emphasizing that both versions of the DPSO algorithm are comparable to the proven ACS and PACO metaheuristics in terms of solution quality. In fact, heterogeneous DPSO is able to generate solutions of better quality than both of ACO-based algorithms in most cases, while also exhibiting more rapid convergence if the computation time is extended.

In the future, we plan to test different types of heterogeneity in addition to the parameter diversity considered here.

Author Contributions: Conceptualization, Łukasz Strąk, Rafał Skinderowicz, Urszula Boryczka and Arkadiusz Nowakowski; Formal analysis, Łukasz Strąk, Rafał Skinderowicz and Urszula Boryczka; Investigation, Łukasz Strąk, Rafał Skinderowicz, Urszula Boryczka and Arkadiusz Nowakowski; Methodology, Łukasz Strąk, Rafał Skinderowicz, Urszula Boryczka and Arkadiusz Nowakowski; Project administration, Łukasz Strąk and Rafał Skinderowicz; Resources, Łukasz Strąk and Rafał Skinderowicz; Software, Łukasz Strąk and Rafał Skinderowicz; Supervision, Łukasz Strąk and Rafał Skinderowicz; Validation, Łukasz Strąk and Rafał Skinderowicz; Visualization, Łukasz Strąk and Rafał Skinderowicz; Writing – original draft, Łukasz Strąk, Rafał Skinderowicz and Urszula Boryczka; Writing – review & editing, Łukasz Strąk, Rafał Skinderowicz and Urszula Boryczka.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this paper:

ACO	ant colony optimization
DPSO	discrete particle swarm optimization
DTSP	dynamic travelling salesman problem
PACO	population ant colony optimization
PSO	particle swarm optimization
TSP	travelling salesman problem

References

1. Branke, J. Evolutionary approaches to dynamic environments. *In GECCO Workshop on Evolutionary Algorithms for Dynamics Optimization Problems* **2001**.
2. Li, W. A parallel multi-start search algorithm for dynamic traveling salesman problem. International Symposium on Experimental Algorithms. Springer, 2011, pp. 65–75.
3. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. *In Proceedings of the IEEE International Conference on Neural Networks* **1995**, pp. 1942–1948.
4. Kennedy, J.; Eberhart, R.C. A discrete binary version of the particle swarm algorithm. Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, 1997, Vol. 5, pp. 4104–4108 vol.5.
5. Clerc, M. Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem. In *New Optimization Techniques in Engineering*; Springer Berlin Heidelberg, 2004; Vol. 141, *Studies in Fuzziness and Soft Computing*, pp. 219–239.
6. Shi, X.H.; Liang, Y.C.; Lee, H.P.; Lu, C.; Wang, Q. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters* **2007**, *103*, 169–176.
7. liang Zhong, W.; Zhang, J.; neng Chen, W. A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. In *Evolutionary Computation, 2007. CEC 2007*; IEEE, 1997; Vol. 14, pp. 3283–3287.
8. Strąk, Ł.; Skinderowicz, R.; Boryczka, U. Adjustability of a discrete particle swarm optimization for the dynamic TSP. *Soft Computing* **2018**, *22*, 7633–7648.
9. Hansell, M. *Built by Animals*; Oxford University Press: New York, 2007.
10. Nepomuceno, F.V.; Engelbrecht, A.P. A Self-adaptive Heterogeneous PSO Inspired by Ants. In *Swarm Intelligence*; Springer Berlin Heidelberg, 2012; Vol. 7461, *Lecture Notes in Computer Science*, pp. 188–195. doi:10.1007/978-3-642-32650-9_17.
11. Montes de Oca, M.A.; Peña, J.; Stützle, T.; Pinciroli, C.; Dorigo, M. Heterogeneous Particle Swarm Optimizers. In *IEEE Congress on Evolutionary Computation (CEC 2009)*; Haddow, P.; others., Eds.; IEEE Press: Piscatway, NJ, 2009; pp. 698–705.
12. Boryczka, U.; Strąk, L. Heterogeneous DPSO Algorithm for DTSP. In *Computational Collective Intelligence*; Núñez, M.; Nguyen, N.T.; Camacho, D.; Trawiński, B., Eds.; 2015; Vol. 9330, *Lecture Notes in Computer Science*, pp. 119–128.
13. Psaraftis, H. Dynamic vehicle routing problems. *Vehicle routing: Methods and studies* **1988**, *16*, 223–248.
14. Li, C.; Yang, M.; Kang, L. A new approach to solving dynamic traveling salesman problems. Proceedings of the 6th international conference on Simulated Evolution And Learning; Springer-Verlag: Berlin, Heidelberg, 2006; SEAL'06, pp. 236–243.
15. Cook, W.J. *In pursuit of the traveling salesman: mathematics at the limits of computation*; Princeton University Press, 2011.
16. Guntsch, M.; Middendorf, M. Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP. In *Applications of Evolutionary Computing*; Springer Berlin Heidelberg, 2001; Vol. 2037, *Lecture Notes in Computer Science*, pp. 213–222.
17. Eyckelhof, C.J.; Snoek, M. Ant Systems for a Dynamic TSP. In *Ant Algorithms*; Springer Berlin Heidelberg, 2002; Vol. 2463, *Lecture Notes in Computer Science*, pp. 88–99.
18. Helsgaun, K. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research* **2000**, *126*, 106–130.

19. Mavrovouniotis, M.; Yang, S. Ant Colony Optimization with Immigrants Schemes in Dynamic Environments. In *Parallel Problem Solving from Nature, PPSN XI*; Springer Berlin Heidelberg, 2010; Vol. 6239, *Lecture Notes in Computer Science*, pp. 371–380.
20. Simões, A.; Costa, E. CHC-Based Algorithms for the Dynamic Traveling Salesman Problem. In *Applications of Evolutionary Computation*; Springer Berlin Heidelberg, 2011; Vol. 6624, *Lecture Notes in Computer Science*, pp. 354–363.
21. Younes, A.; Basir, O.; Calamai, P. A Benchmark Generator for Dynamic Optimization. Proceedings of the 3rd WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems, 2003.
22. Tinós, R.; Whitley, D.; Howe, A. Use of Explicit Memory in the Dynamic Traveling Salesman Problem. Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation; ACM: New York, NY, USA, 2014; GECCO '14, pp. 999–1006.
23. Zhang, Y.; Zhao, G., Research on Multi-service Demand Path Planning Based on Continuous Hopfield Neural Network. In *Proceedings of China Modern Logistics Engineering: Inheritance, Wisdom, Innovation and Cooperation*; Springer Berlin Heidelberg: Berlin, Heidelberg, 2015; pp. 417–430.
24. Eaton, J.; Yang, S.; Mavrovouniotis, M. Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays. *Soft Computing* **2016**, *20*, 2951–2966. doi:10.1007/s00500-015-1924-x.
25. Mavrovouniotis, M.; Shengxiang, Y. Empirical study on the effect of population size on MAX-MIN ant system in dynamic environments. 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 853–860. doi:10.1109/CEC.2016.7743880.
26. Mavrovouniotis, M.; Müller, F.M.; Shengxiang, Y.M.M. Ant Colony Optimization With Local Search for Dynamic Traveling Salesman Problems. *IEEE Transactions on Cybernetics* **2017**, *47*, 1743–1756. doi:10.1109/TCYB.2016.2556742.
27. Cáceres, L.P.; López-Ibánez, M.; Stützle, T. Ant colony optimization on a budget of 1000. In *Swarm Intelligence*; Springer, 2014; pp. 50–61.