International Journal of Modern Manufacturing Technologies
ISSN 2067–3604, Vol. XIV, No. 3 / 2022
https://doi.org/ 10.54684/ijmmt.2022.14.3.299

A NEW PSO-BASED ALGORITHM FOR AN OPERATIONAL MANAGEMENT PROBLEM

Adrian Vilcu, Ionut Herghiligiu, Ion Verzea, Raluca Lazarescu

"Gheorghe Asachi" Technical University of Iasi, Department of Economical Engineering,
Blvd. Mangeron, No. 59A, 700050, Iasi, Romania

Corresponding author: Adrian Vilcu, adrian.vilcu@academic.tuiasi.ro

Abstract. Operational management issues represent a permanent challenge for the current economic environment and the research activity. This research will model a Travelling Salesman Problem (TSP). The complexity of this fundamental problem (np-hard) allows a chance to apply and develop heuristic methods and evolutionary algorithms along with exact methods (dynamic programming, branch & bound). This paper proposes a new discrete algorithm to solve the TSP based on the Particle Swarm Optimization (PSO) technique. The features of this method are fast determination through an iterative process of the optimal problem, the generalised search in all the solutions, and the avoidance of the local optimal. To avoid premature convergence, we have introduced a new operator with a new mathematical function, and we have proposed new techniques for measuring and maintaining population diversity. We tested the algorithm's performance by applying it to numerical instances and compared it to the solutions and performance provided by other evolutionary algorithms. By delaying the convergence process, the new algorithm PSO offers reasonable solutions in terms of quality comparable to those offered by different evolutionary algorithms tested. At the end of the research, we highlighted the conclusions, limitations, and new techniques based on the PSO algorithm.

Key words: algorithm, particle swarm optimization, management.

1. INTRODUCTION

Travelling Salesman Problem (TSP) is a well-studied reference in the field of combinatorial optimisation. Given a lot of cities $V = \{v_1, \dots, v_n\}$, a set of edges $A = \{(i,j); i,j \in V\}$, and a cost associated with the edge, the travel commissioner's problem is the problem of finding a minimum cost road through which to visit all cities, going through each city only once. In the real world, it solves job scheduling problems, vehicle routing problems with pick-up and delivery constraints, or transportation problems in flexible production systems. Many evolutionary algorithms have been proposed to solve them, including Simulated Annealing, Taboo Search, and genetic or "ant colony" algorithms [1][2][3].

Compared to these, PSO benefits from a "smart" knowledge base, the solution of the problem at a higher level than a collection of idealised agents, without the intention of solving by the individual. This optimisation technique is inspired by the social behaviour (primarily related to the transmission and sharing of information) of living beings such as flocks of birds, swarms of bees or shoals of fish. Although initially designed to optimise continuous nonlinear functions, it was later adapted to solve permutation problems, constraint problems, and optimisation in dynamic or multi-objective environments. In artificial variants, the search process is provided by a set of "particles" whose movement is characterised by a "speed" that changes over time depending on the characteristics of the entire system [4]. This algorithm allows to find the optimal one quickly but encounters difficulties in avoiding the local minimums [5]. The decrease in the search space's diversity leads to total implosion and stagnation of the swarm's fitness [6].

The method consists of a group of particles that move in the search space. Their trajectory is determined by the value of the best fitness found. Each particle is represented by a vector *x* of length *n* indicating the position in n-dimensional space and has a velocity vused to update the current status. The speed vector is calculated taking into account a set of rules, which will be presented in one of the following sections. The algorithm introduces a coefficient for measuring diversity, which is used to calculate the diversity level of the current swarm. A new particle rejection procedure, called a depressor, has been introduced to prevent premature convergence. The coefficient of diversity is used to interchange the process of attraction (*attractor*) and that of rejection (*depressor*). When the diversity index of the current swarm is below a certain level, the depressor is turned on until the diversity index passes that level. At this point, the attraction phase begins [7].

2. MATERIAL AND METHOD

2.1. Representation of the solution and the cost function

Let $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ - the position of the particle $x_{i1} -> x_{i2} -> \dots -> x_{im} -> x_{i1}$ is the order of passage through cities. Velocity is defined as a list of pairs (i, j), i jthat are indices of the elements of the permutation vector. Finally, we describe the matrix: as cost[i][j]=0 if there is no edge between x_i nod x_j and the cost of the edge - if there is an arc between x_i nod x_j. The problem is determining a permutation of *m* nodes for which the cost function is minimal.

2.2. Description of the Hybridized PSO algorithm (HPSO) for TSP

Over time, various algorithms have been proposed to solve the traveller's problem: Maurice Clerc tried to implement the PSO method in the standard and discrete version, but the results showed a lower efficiency than other known algorithms [8]. Rameshkumar has improved this variant by adding a permutation function that interchanges the elements in the locations corresponding to the indices [9]. The algorithm proposed by us introduces the concept of a permutation into an HPSO algorithm. Next, we will use the following notations to describe the problem and the notions with which it operates.

For a problem of size *m*, the position of the particle i representing the path starting from $x_{i1} => x_{i2} => \dots => x_{im} => x_{i1}$, is denoted $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$

Variables used by the algorithm:

$P_{k,best}$ - the best position of the particle *k*;

$P_{k,worst}$ - the worst position of the particle *k*;

$P_{k,current}$ - the current position of the particle *k*;

G_{best} - the current best global position;

v_k^t - velocity of the particle *k* during iteration *t*.

HPSO algorithm is presented in figure 1.

```
step 1: initialise parameters and population with random positions and speeds repeat
{     step 2: evaluate the fitness function for each particle
      step 3: local best calculation for each particle
      step 4: best global calculation
      step 5: speed calculation
      step 6: calculate the position
}
until (stop condition == true);
```

Fig. 1. The HPSO algorithm

The mechanism for generating new solutions is presented in figure 2. The HPSO algorithm, at each iteration, improves the best n/2 solutions (*), rejects the least competitive solutions and generates other solutions (**) (figure 2).

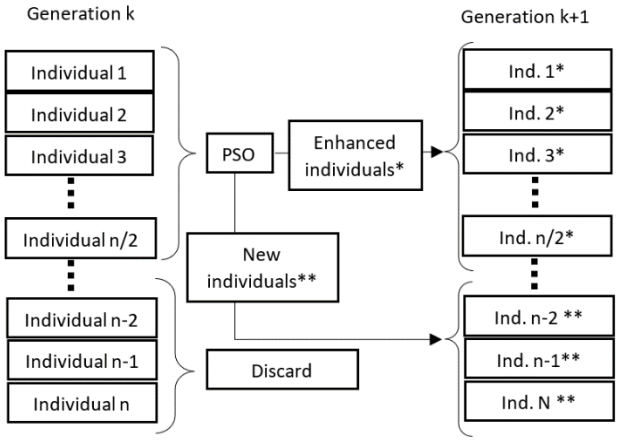

Fig. 2. The mechanism for generating new solutions

HPSO algorithm parameters are as follows:

- N - population size
- cp and cg (cp + cg <= 4) - learning factors - represent the weight of the terms acceleration
- cp - cognitive parameter - the tendency to duplicate one's own past actions that have proved successful
- cg - social parameter - the tendency to follow the success of other individuals
- w - the weight of inertia - ensures the balance between exploration/exploitation; it is also used as a decreasing weight over time (as in Simulated Annealing)

HPSO algorithm operators are presented below:

- *Slide Operator (SL).* For an *m*-dimensional sequence $X = \{x_1, x_2, \dots, x_m\}$, the slide operator acting on the particle X is defined: $SL(X, k) = \{x_{(1+k)\%,m}, x_{(2+k)\%,m}, \dots, x_{(m+k)\%,m}\}$
- *Reverse Operator (RE).* For an m - dimensional sequence $X = \{x_1, x_2, \dots, x_m\}$, the inverse operator acting on the particle X is: $RE(X) = \{x_m, x_{m-1}, \dots, x_1\}$
- *Two Particle Position Submission (SUB) Operator.* Given the position of two particles $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and $X_j = \{x_{j1}, x_{j2}, \dots, x_{jm}\}$ we define the decrease of the two as: $SUB(X_1, X_2) = X_1\text{-}X_2 = \cup_{k=1}^{m} P(X_{ik}, X_{jk})$

where: $P(x_{ik}, x_{jm}) = \begin{cases} \{(x_{ik}, x_{jk})\}, \text{if } x_{ik} \neq x_{jk} \\ \emptyset, \text{if } x_{ik} = x_{jk} \end{cases}$

- *Operator (COMPOSE).* Given the position of two particles $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and $X_j = \{x_{j1}, x_{j2}, \dots, x_{jm}\}$ we define the operator $\odot$ on the two as follows:

$$X_i \odot X_j = SUB\left(X_i, \left(SL(RE(X_j), l)\right)\right)$$

or

$$X_i \odot X_j = SUB\left(X_i, RE\left(SL(X, l)\right)\right)$$

where l takes values between 0 and m-1, which can minimise | Xi ⊙ Xj |

- *Two-particle addition operator (ADD).* Given the positions of two particles $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and $X_j = \{x_{j1}, x_{j2}, \dots, x_{jm}\}$ we define the sum of the two as follows:

$$ADD(X_i, X_j) = X_i + X_j = \bigcup_{k=1}^{m} E(X_{ik}, X_{jk})$$

where $E(x_{ik}, x_{jk}) = \begin{cases} \{x_{ik}\}, & \text{if } x_{ik} = x_{jk} \\ \emptyset, & \text{if } x_{ik} \neq x_{jk} \end{cases}$

- *Random generation of a set* R. For a set number of times$S = \{e_1, e_2, \dots, e_s\}$, the result of the operation G(S)is to randomly generate a set Ras follows$G(S) = R = \{r_1, r_2, \dots, r_{|2|}\}$, where: $r_i = \{(r_{il}, r_{ik}) | r_{ik} \in S \wedge r_{il} \in S \wedge r_{ik} \neq r_{il}\}$.
- *Operator (PLUS)* $\oplus$. Given the positions of two particles $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and $X_j = \{x_{j1}, x_{j2}, \dots, x_{jm}\}$ we define the operator $\oplus$ acting on them as follows:

$$X_i \oplus X_j = G\left(ADD(X_i, SL(RE(X_j), l)\right) \text{ or}$$

$$X_i \oplus X_j = G\left(ADD(X_i, RE\left(SL(X_j, l)\right)\right)$$

where l takes a value between 0 and m-1, which can maximise| R |.

Swarm diversity coefficient. For a given swarm S, its divergence coefficient div is defined as follows:

$$div = D(S) = \frac{2}{1 + e^{\frac{\sum_{j=1}^{N} \left| \cup_{i=1}^{|S|} \{p_{ij}\} \right|}{N \cdot |S|}}} - 1$$

where

- | S | - the size of the swarm;
- N - is the size of the problem;
- p_{ij} - represents the component j of the current position in particle i .

Speed update. The velocity vector is calculated according to the following rules:

- each particle tends to keep its current direction (inertia)
- each particle is attracted to the best particle it has reached so far p
- each particle is attracted by the best particle in the population g (or in the neighbourhood)

The v_k velocity of each particle is calculated using the equator and depressor equations:

$v_k^{t+1} = w \times v_k^t + \phi_1(P_{k,best} \odot P_{k,current}) + \phi_2(P_{best} \odot P_{k,current})$, $div > c_h$ or $Iter_{max}$-$Iter_{curr} < num$

$v_k^{t+1} = w \times v_k^t + \phi_1(P_{k,worst} \oplus P_{k,current}) + \phi_2(P_{worst} \oplus P_{k,current})$, $div \leq c_l$ and $Iter_{max}$-$Iter_{curr} \geq num$

where:

- *w* inertial weight
- Φ1 and Φ2 are learning coefficients
- c_l and c_h represent the lower and upper limit
- num is a constant that has to do with the concrete problem
- $Iter_{max}$ and $Iter_{curr}$ represent the maximum number of iterations and the current number of iterations

Each particle is updated using the equation $P_k^{t+1} = P_k^t + v_k^{t+1}$. *New Position = Current Position + Particle Speed.*

Position update. The formulas used to update each individual in the population from t iteration to the $t+1$ iteration are:

- Rule 1: for the given position of a particle $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and the velocity containing a number of qtransposed pairs $V_i = \{\{k_{i1}, l_{i1}\}, \{k_{i2}, l_{i2}\}, \dots, \{k_{iq}, l_{iq}\}\}$, $X_i + v$ generates the position of a new particle X new by first applying the transposition V_i on X_i, then on the result, and so on.
- Rule 2: Let Φ be the learning coefficient and $V_i = \{\{k_{i1}, l_{i1}\}, \{k_{i2}, l_{i2}\}, \dots, \{k_{iq}, l_{iq}\}\}$ the speed.

The operation Φ x Vi is a new velocity V_k and is used to find the number of velocity components that will be applied to the position.

- Rule 3: Either: $V_i = \{\{k_{i1}, l_{i1}\}, \{k_{i2}, l_{i2}\}, \dots, \{k_{iq}, l_{iq}\}\}$ and $V_j = \{\{k_{j1}, l_{j1}\}, \{k_{j2}, l_{j2}\}, \dots, \{k_{jq}, l_{jq}\}\}$ then $V_i + V_j = V_i \cup V_j$

Fitness function. The fitness function is used to measure the quality of the particles involved in the algorithm. It is formulated according to the numerical problem to be optimised. It must be positive and it is built to minimise (adapted individuals get low values of fitness function). If we have the particle $X_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ then the fitness function is: $\sum(d(x_{i1}, x_{i2}), d(x_{i2}, x_{i3}), \dots)$ where $d(x_{ij}, x_{ik})$ represents the cost from x_{ij} to x_{ik}.

Implementation. Input data is read from files, and the application can interpret both geographical and Euclidean coordinates. These files are the files used by the combinatorial optimisation research group at the University of Heidelberg, Germany [10, 11], being standard files used in testing HPSO algorithms applied over TSP.

The structure of such a file is presented in figure 3:

In the first place, each file has a series of information about the problem, the name, the type of the problem, the size, possibly comments, the type of geographical coordinates (GEO) or Euclidean (EUC_2D). Then, based on this information, the nods in the statement of that issue can be found in the file.

The classic PSO algorithm hybridisation (HPSO). Difference from the original version of the PSO, in the hybridised version, the velocity was considered as a list of pairs representing the positions in the particle that will be exchanged.

```
TYPE: TSP
COMMENT: 14-Staedte in Burma (Zaw Win)
DIMENSION: 14
EDGE_WEIGHT_TYPE: GEO
EDGE_WEIGHT_FORMAT: FUNCTION
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
  1  16.47      96.10
  2  16.47      94.44
  3  20.09      92.54
  4  22.39      93.37
.....
```

Fig. 3. The TSP file structure

Thus, each pair consists of 2 positions p1 and p2, which will be interchanged.

Three rules, described above (R1, R2, R3) were also used to update the current position. Because in Yang's work [7] these rules were described and no instructions were given on how most efficiently they could be used in the algorithm, it was decided that each rule should have a chance of success on the iteration: r_1, r_2, r_3 . Once these rules were used as independent rules and each with its own chance of success, the problem that arose was that the right values could not be found for each coefficient of the rule to receive. Thus, a genetic algorithm was built over the PSO algorithm to solve this problem. The genetic algorithm generates the appropriate values for r_1, r_2, r_3 and in the PSO algorithm, for each of the rules, values between 0 and 1 are randomly generated. If the randomly generated value of a_i is less than r_i then rule "i" will apply.

In addition to speed, another element that differs from the classic PSO is the use in updating the speed of the worst ("worst") position ever held by the particle. The classical algorithm suffers from the problem of locking in a local minimum. Considering the use of the worst position of the current particle, it helps the algorithm when blocking in a local minimum, managing to get it out of that block.

3. NUMERICAL RESULTS

In order to verify the efficiency of the algorithm, we used several instances of the problem (burmal14, eil101), and the obtained results were compared with those of other three algorithms (a genetic algorithm, an "ant colony" algorithm and one of the classical PSO type). Table 1 and table 2 show the values obtained for each algorithm at an average of n=10 runs of each of them.

In the first case (Rurma14), the HPSO algorithm finds the optimal solution of the numerical instance, as well as the ant-type algorithm and in the second case (Eil101) the HPSO algorithm finds the best solutions (on average and best solution) among the four evolutionary algorithms.

Table 1. Comparative results for Burma14 test

Algorithm	Burma14		
	Mean value	The best value	Optimum value
Genetic	39.9715	31.5573	30.8785
Ant colony	31.8075	30.8785	
PSO	31.5551	31.1245	
HPSO	32.3314	30.8785	

Table 1. HSPO results on Eil101 numerical instance

Algorithm	Eil101		
	Mean value	The best value	Optimum value
Genetic	724.6385	723.8950	629
Ant colony	724.5137	720.6885	
PSO	695.5432	691.2341	
HPSO	643.4293	634.3423	

4. CONCLUSIONS

The TSP problem is recognised as a typical combinatorial optimisation and, in addition, np-complete. This paper proposes a way to solve it through the class of hybrid PSO algorithms, introducing as novelty elements other formulas for operators and a heuristic for their application. The performed experiments demonstrated the efficiency of the presented algorithm, obtaining results very close to the optimal values obtained by applying the already existing algorithms. In addition, it can be used to solve problems of permutation or routing of vehicles, job planning, constraint problems, and optimisation in dynamic or multi-objective environments.

5. REFERENCES

1. Henderson, D., et al., (2006). *The Theory and Practice of Simulated Annealing, in* Handbook of Metaheuristics. eds. Glover, F., Kochenberg, G.A., Springer, pp 287-319.

2. Díaz, E., Tuya, J., Blanco, R., (2003). *Automated Software Testing Using a Metaheuristic Technique Based on Tabu Search,* Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03), 1527-1366/03.

3. Xu, Z, et. al., (2003). *Ant algorithm-based task scheduling in grid computing*, CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology, Montreal, QC, Canada, 3, 1107-1110.

4. Deng, W., et. al., (2017). *A novel intelligent diagnosis method using optimal LS-SVM with improved PSO algorithm*, Soft Computing, 23, 2445–2462.

5. Oca, M.A.M., et. al., (2000). *Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm,*

IEEE Transactions on Evolutionary Computation, 13 (5), 1120 – 1132.
6. Goldbarg, E.F.G., Goldbarg, M.C., de Souza, G.R. (2008). *Particle Swarm Optimization Algorithm for the Traveling Salesman Problem*, Traveling Salesman Problems, ed. Greco, F., InTech, Rijeka, 75-96.
7. Wang, C., Yang, Q., (2007). *An Improved Discrete Particle Swarm Optimization Algorithm for TSP*, IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology – Workshops.
8. Clerc, M., (2004). *Discrete particle swarm optimisation, illustrated by the Traveling Salesman Problem*, New Optimization Techniques in Engineering, Heidelberg, Germany, pp 219-239.
9. Rameshkumar, K., Suresh, R.K., Sundaram, K.M.M., (2005). *Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flowshop Scheduling to Minimise Makspan*, Proc. ICNC 2005, LNCS 3612, pp 572-581.
10. G. Reinelt, http://www.iwr.uniheidelberg.de/ groups /comopt/software/TSPLIB95, accessed date: 10.01.2022.
11. https://github.com/coin-or/jorlib/tree/master/ jorlib-core/src/test/resources/tspLib/tsp, accessed date: 10.01.2022.

Received: April 27, 2022 / Accepted: December 15, 2022 / Paper available online: December 20, 2022 © International Journal of Modern Manufacturing Technologies