Borna & Khezri, *Cogent Mathematics* (2015), 2: 1048581
http://dx.doi.org/10.1080/23311835.2015.1048581

cogent mathematics

COMPUTATIONAL SCIENCE | RESEARCH ARTICLE

A combination of genetic algorithm and particle swarm optimization method for solving traveling salesman problem

Keivan Borna[1]* and Razieh Khezri[2]

Abstract: Traveling salesman problem (TSP) is a well-established NP-complete problem and many evolutionary techniques like particle swarm optimization (PSO) are used to optimize existing solutions for that. PSO is a method inspired by the social behavior of birds. In PSO, each member will change its position in the search space, according to personal or social experience of the whole society. In this paper, we combine the principles of PSO and crossover operator of genetic algorithm to propose a heuristic algorithm for solving the TSP more efficiently. Finally, some experimental results on our algorithm are applied in some instances in TSPLIB to demonstrate the effectiveness of our methods which also show that our algorithm can achieve better results than other approaches.

Subjects: Algorithms & Complexity; Applied Mathematics; Computer Mathematics; Computer Science; Linear Programming; Mathematics & Statistics; Science; Technology

Keywords: traveling salesman problem; particle swarm optimization; genetic algorithm; optimization

Received: 26 December 2014
Accepted: 30 April 2015
Published: 18 June 2015

*Corresponding author: Keivan Borna, Faculty of Mathematics and Computer Science, Kharazmi University, Tehran, Iran
E-mail: borna@khu.ac.ir

Reviewing editor:
Cedric K.F. Yiu, Hong Kong Polytechnic University, Hong Kong

Additional information is available at the end of the article

1. Related works

The origins of the traveling salesman problem (TSP) are unclear. A handbook for traveling salesmen from 1832 mentions the problem and includes example tours through Germany and Switzerland, but contains no mathematical treatment.

Mathematical problems related to the TSP were treated in the 1800s by the Irish mathematician W. R. Hamilton and by the British mathematician Thomas Kirkman. Hamiltons Icosian Game was a

ABOUT THE AUTHORS

Keivan Borna joined the Department of Computer Science at the Faculty of Mathematics and Computer Science of Kharazmi University as an Assistant Professor in 2008. He earned his PhD in Computational Commutative Algebra from the Department of Mathematics, Statistics and Computer Science of the University of Tehran. His research interests include Computer Algebra, Cryptography, Approximation Algorithms, and Computational Geometry. He is the author of the "Advanced Programming in JAVA" (in Persian) and is a life member of "Elite National Foundation of Iran".

Razieh Khezri was graduated from Faculty of Engineering at Kharazmi University, Tehran, Iran. Her research interests include evolutionary computations.

PUBLIC INTEREST STATEMENT

This paper was benefited from a clean idea of combining two important methods in the famous "Traveling Salesman Problem":

Genetic Algorithm + Particle Swarm Optimization

This study enables us to make use of capabilities of both approaches. The practical results also admit the improvement of our approach.

recreational puzzle based on ending a Hamiltonian cycle. The general form of the TSP appears to have been first studied by mathematicians during the 1930s in Vienna and at Harvard, notably by Karl Menger, who defines the problem, considers the obvious brute-force algorithm, and observes the non-optimality of the nearest neighbor heuristic (Pang, Wang, Zhou, & Dong, 2004; Zhang, Guan, & Liu, 2007).

A salesman spends his time visiting *n* cities (or nodes) cyclically. In one tour, he visits each city just once, and finishes up where he started. The aim of the problem is to minimize the distance traveled (Lin & Kernighan, 1973).

Several techniques to obtain the optimal solution for the TSP have been provided by researchers, such as genetic algorithms (GAs), ant colony optimization (ACO), simulated annealing, neural networks, particle swarm optimization (PSO), evolutionary algorithms, mimetic computing, etc. Among them, PSO has gained much attention and been successfully applied in a variety of fields mainly for optimization problems (Zhang et al., 2007).

In Xu, Cheng, Yang, Yang, and Wang (2013), the concepts of mobile operators and mobile sequence are introduced, with which it reddened the rate of PSO algorithm and the formula of position updating. In Liao, Yau, and Chen (2012), the improved version of the PSO approach to solve TSP, which uses Fuzzy C-Means clustering, a rule-based route permutation, a random swap strategy and a cluster merge procedure is introduced. In Pang et al. (2004), fuzzy matrices were used to represent the position and velocity of the particles in PSO. In elsewhere, a discrete PSO is presented to solve TSP on a set of benchmark instances. The discrete PSO algorithm exploited the basic features of its continuous counterpart (Tasgetiren, Suganthan, & Pan, 2007). In another study, the generalized TSP by employing the generalized chromosome is solved. The subtraction operator between two particle positions is modified and a discrete PSO method is constructed for the TSP (Shi, Liang, Lee, Lu, & Wang, 2007). In Yan et al. (2012), the authors proposed an improved PSO by using a self-organizing construction mechanism and dynamic programming algorithm; their optimized model for the balanced multiple-salesman problem with time and capacity constraints is presented. It requires that a salesman visits each vertex at least once and returns to the starting vertex within given time. In Pang, Li, Dai, and Yu (2013), the traveling salesman problem was solved using GA approach.

More precisely, the system starts from a matrix of the calculated Euclidean distances between the cities to be visited by the traveling salesman and randomly chosen city order as the initial population. Then new generations are created repeatedly until the proper path is reached upon reaching a stopping criterion. In Gupta and Panwar (2013), the recent TSP-based re-indexing technique with PSO was enhanced. In this study, the color re-indexing is done by solving the problem as a TSP using PSO. In this paper, we combine the capability of PSO algorithm and heuristic crossover operator of GA to gain more efficiency.

2. A brief introduction to PSO

PSO is an approach to problems whose solutions can be represented as a point in an n-dimensional solution space. A number of particles are randomly set into motion through this space. At each iteration, they observe the "fitness" of themselves and their neighbors and "emulate" successful neighbors (those whose current position represents a better solution to the problem than theirs) by moving towards them. Various schemes for grouping particles into competing, semi-independent flocks can be used, or all the particles can belong to a single global flock. This extremely simple approach has been surprisingly effective across a variety of problem domains. PSO was developed by James Kennedy and Russell Eberhart in 1995 after being inspired by the study of bird flocking behavior by biologist Frank Heppner. It is related to evolution-inspired problem-solving techniques such as GAs (see Ai & Kachitvichyanukul, 2008; Arumugam & Rao, 2008; Chen et al., 2010; Hu, 2006; Kennedy & Eberhart, 1995, 2001; Labed, Gherboudj, & Chikhi, 2012; Li & Zhang, 2007; Liao et al., 2012, for more details and some recent advances using PSO).

cogent mathematics

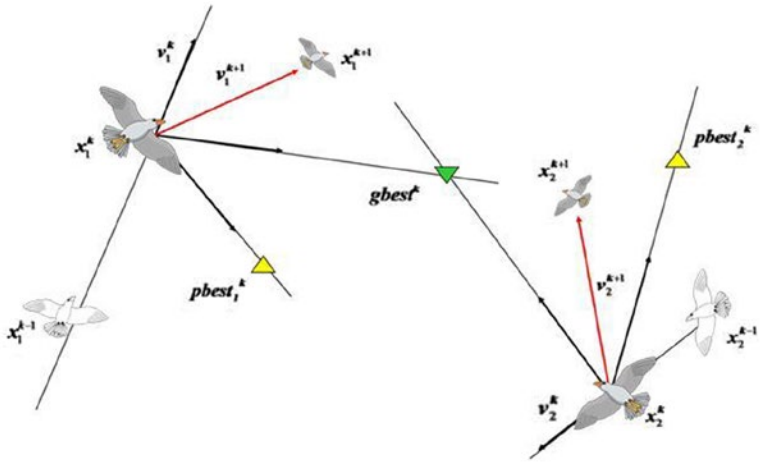


Figure 1. What is the best strategy to find the food?

PSO simulates the behaviors of bird flocking (Figure 1). Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles. PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. At each iteration, each particle is updated by the following two values: the first one is the best solution (fitness) it has achieved so far, which is called *pbest*. Another value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the population. This best value is a global best and called *gbest*. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called *lbest*. After finding the two best values, the particle updates its velocity and positions with following Equation 1 and 2.

For each $1 \le i \le s$, where s is the size of swarm:

$$v[i] = v[i] + c1 \times rand() \times \left(pbest[i] - \text{present}[i]\right) + c2 \times rand() \times \left(gbest[i] - \text{present}[i]\right) \quad (1)$$

$$\text{present}[i] = \text{present}[i] + v[i] \quad (2)$$

where v[i] is the ith particle velocity, present[i] is ith particle solution, *pbest*[i] and *gbest*[i] are defined as stated before, and *rand*() is a random number in range [0; 1). $c1$, $c2$ are learning factors usually $c1 = c2 = 2$.

Particles' velocities on each dimension are clamped to a maximum velocity. If the sum of accelerations would cause the velocity on that dimension to exceed V_{max} which is a parameter specified by the user. Then the velocity on that dimension is limited to V_{max} (see Hu, 2006, for more details.

***Algorithm 1**: Pseudo code for PSO*

1. ***Initialization***

   Parameters and size of the swarm (S)

   Randomly initialize particles positions and velocities

   ***For each** particle, let pbest = x*

Calculate f(x) of each particle

Calculate gbest

2. ***While*** (*termination criterion is not met*)

For *i = 1 to S*

Calculate the new velocity using equation (*1*)

Calculate the new position using equation (*2*)

Calculate f(x) of each particle

If *(f(x) < f(pbest)) pbest = x*

If *(f(pbest) < f(gbest)) gbest = pbest*

3. ***Show*** *the best solution found gbest*

3. A comparisons between GA and PSO

According to Kachitvichyanukul (2012) and van Hook, Sahin, and Arnavut (2008), most of evolutionary techniques have the following procedure:

(1) Random generation of an initial population.

(2) Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.

(3) Reproduction of the population based on fitness values.

(4) If requirements are met, then stop. Otherwise go back to 2.

From the procedure, we can learn that PSO shares many common points with GA. Both algorithms start with a group of a randomly generated population, both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee success.

However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm.

Compared with GA, the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only *gbest* gives out the information to others. It is a one-way information sharing mechanism. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases.

4. Crossover operation

In optimization, *k*-opt is a simple local search algorithm first proposed by Cores in 1958 for solving the TSP (Labed et al., 2012). The main idea behind it is to take a route that crosses over itself and reorder it so that it does not.

The 2-opt is one of the models of *k*-opt. The 2-opt operator will have to be slightly modified in order to be used in multi-objective TSP. A simple way to apply the 2-opt operator would be to randomly apply it either to the first objective, or the second objective, and so on, with equal probabilities. Let's call this method A.

But this yields an uneven non-dominated set, with a majority of the points flying at the extremities. An alternative would be to sum up the objective function and use $d(x; y)$ as the sum of the individual costs or distances between the cities x and y. Let's call this method B. This method results in a dense midsection in the non-dominated set, but values do not reach extreme values of the objectives.

During the improvement process, the algorithm examines whether the exchange of two edges produces a shorter tour. The algorithm continues until no improvement is possible Figure 2. The steps of the algorithm are presented below:

Figure 2. The diagram of PSO algorithm.

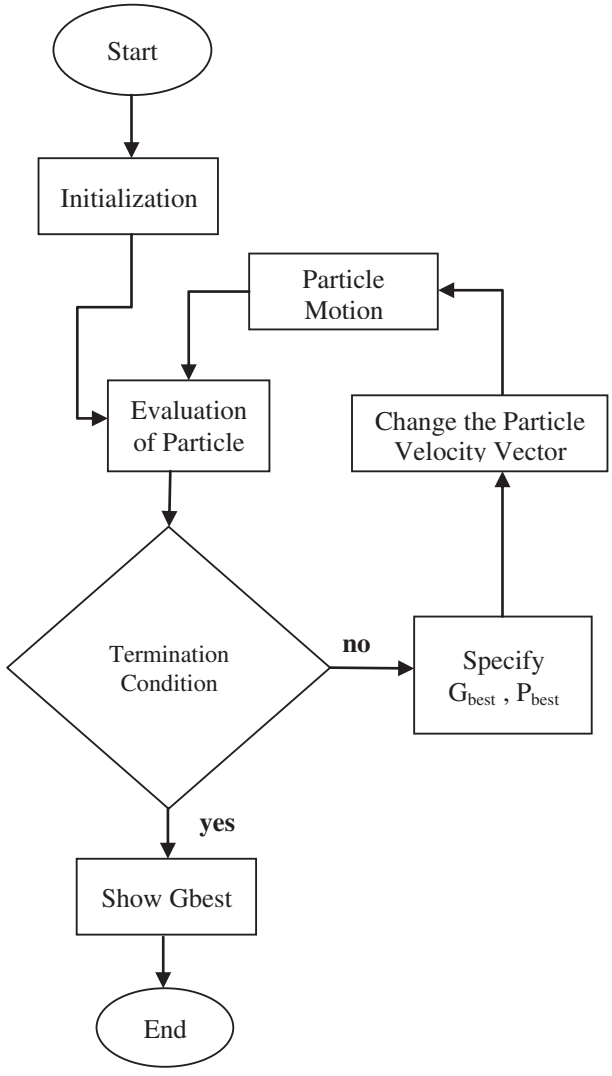
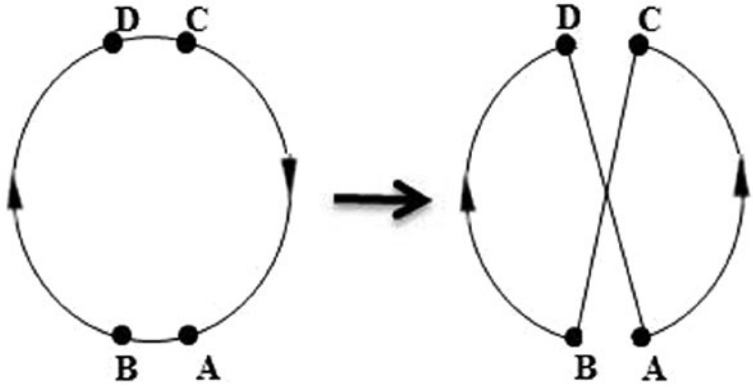


Figure 3. The 2-opt operator.



(1) Take two pairs of consecutive nodes, pairs (A, B) and (C, D) from a tour (see Figure 3).
(2) Check if the distance (AB + CD) is higher than (AD + BC).
(3) If that is the case, swap A and C, resulting in reversing the tour between the two nodes.
(4) If improvement of the tour, then go to 1, otherwise stop (see Labed et al., 2012).

The time complexity for the 2-opt algorithm is $O(n^2)$.

For the ease of reader and also for the sake of completeness, we have include an explanation of the Crossover Operation mainly from Genetic Server and Genetic Library (NeuroDimension: www.nd.com/products/genetic.htm) in the next section.

5. Crossover operation

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. In our proposed algorithm, we have introduced the crossover operator in the aim to produce a new population.

5.1. Types of crossover

1. One Point: A crossover operator that randomly selects a crossover point within a chromosome then interchanges the two parent chromosomes at this point to produce two new offspring. Consider the following two parents which have been selected for crossover. The symbol | indicates the randomly chosen crossover point.

Parent 1: 11001/010

Parent 2: 00100/111

After interchanging the parent chromosomes at the crossover point, the following offspring are produced:

Offspring 1: 11001/111

Offspring 2: 00100/010

2. Two Point: A crossover operator that randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring. Consider the following two parents which have been selected for crossover.

Parent 1: 110/010/10

Parent 2: 001|001/11

After interchanging the parent chromosomes between the crossover points, the following offspring are produced:

Offspring 1: 110/001|10

Offspring 2: 001|010/11

5.2. Uniform

A crossover operator decides (with some probability known as the mixing ratio) which parent will contribute each of the gene values in the offspring chromosomes. This allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one and two point crossover). For some problems, this additional flexibility outweighs the disadvantage of destroying

building blocks. Single and multi-point crossovers define cross points as places between loci where a individual can be split. Uniform crossover generalizes this scheme to make every locus a potential crossover point. A crossover mask, the same length as the individual structure, is created at random and the parity of the bits in the mask indicates which parent will supply the offspring with which bits.

Consider the following two parents which have been selected for crossover:

Parent 1: 01110011010
Parent 2: 10101100101

For each variable, the parent who contributes its variable to the offspring is chosen randomly with equal probability. Here, the offspring 1 is produced by taking the bit from parent 1 if the corresponding mask bit is 1 or the bit from parent 2 if the corresponding mask bit is 0. Offspring 2 is created using the inverse of the mask, usually.

Offspring 1: 111/011/1111
Offspring 2: 001/100/0000

Uniform crossover, like multi-point crossover, has been claimed to reduce the bias associated with the length of the binary representation used and the particular coding for a given parameter set. This helps to overcome the bias in single-point crossover towards short substrings without requiring precise understanding of the significance of the individual bits in the individual's representation. It is known how uniform crossover may be parameterized by applying a probability to the swapping of bits. This extra parameter can be used to control the amount of disruption during recombination without introducing a bias towards the length of the representation used.

5.3. Arithmetic

A crossover operator that linearly combines two parent chromosome vectors to produce two new offspring according to the following equations:

$$\text{Offspring } 1 = a \times \text{ Parent1} + (1 - a) \times \text{ Parent2}$$
$$\text{Offspring } 2 = (1 - a) \times \text{ Parent1} + a \times \text{ Parent2}$$

where a is a random weighting factor (chosen before each crossover operation).

Consider the following two parents (each consisting of four oat genes) which have been selected for crossover:

Parent 1: (0:3)(1:4)(0:2)(7:4)
Parent 2: (0:5)(4:5)(0:1)(5:6)

If $a = 0.7$, the following two offspring would be produced:

Offspring 1: (0:36)(2:33)(0:17)(6:86)
Offspring 2: (0:402)(2:981)(0:149)(6:842)

5.4. Heuristic

A crossover operator that uses the fitness values of the two parent chromosomes to determine the direction of the search. The offspring are created according to the following equations:

$$\text{Offspring } 1 = \text{BestParent} + r \times (\text{BestParent} - \text{WorstParent})$$
$$\text{Offspring } 2 = \text{BestParent}$$

where *r* is a random number between 0 and 1. It is possible that offspring 1 will not be feasible. This can happen if *r* is chosen such that one or more of its genes fall outside of the allowable upper or lower bounds. For this reason, heuristic crossover has a user settable parameter *n* for the number of times to try and find results in a feasible chromosome. If a feasible chromosome is not produced after *n* tries, the worst parent is returned as offspring 1.

6. Our algorithm for TSP using PSO

The famous TSP is a typical combination optimization problem, which is often used to assess the performance of the metahuristic algorithms. Currently, metahuristic algorithm mainly consists of GA, ACO algorithm and PSO algorithm. Each of these algorithms for solving TSP has advantages and disadvantages. For example, the GA may spend a lot of time to search for the optimal solution. The ACO spends less time but the ACO falls in local optimum easily. PSO is a relatively simple procedure due to the lack of these problems and has attracted researchers from various fields. TSP can be formulated such that we assume *n* is the number of cities we shall pass through them, and $D = (dij)$ is the distance matrix between each city *i* and *j* where $i, j = 1, 2, 3, \ldots, n$ (see Figure 4 for an example).

In our proposed algorithm for TSP, we use heuristic crossover (HC), which as we can see in the next section, will improve the performance of our algorithm.

***Algorithm 2**: Pseudo code for* heuristic crossover (HC),
Input: Two solutions ***x1*** *and* ***x2***
Output: One solution ***x***
Steps:
Choose a random city ***v***
Move the city ***v*** *in the beginning of* ***x1*** *and* ***x2***
Initialize ***x*** *by* ***v***
i = 2; j = 2
While *[(i & j) <= n]*
  If *(x1(i) & x2(j) ∈ x))*
    i = i + 1
    j = j + 1
  Else If *(x1(i) ∈ x)*
    Concatenate x2(j) to x
    j = j + 1
  Else If *(x2(j) ∈ x)*
    Concatenate x1(i) to x
    i = i + 1
  Else
    Let ***u*** *be the last city in* ***x***

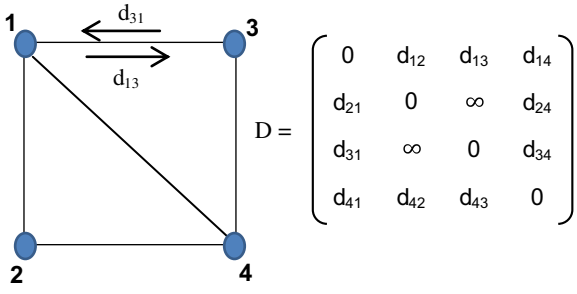


Figure 4. A simple graph and its adjacency matrix as the input for TSP.

If *(distance [u, x1(i)]) < (distance[u, x2(j)])*
concatenate x1(i) to x
i = i + 1
Else
concatenate x2(j) to x
j = j + 1

In this method, HC first chooses a random starting city such as *v*, then compares the two edges emanating from this city in two parents (*x*1, *x*2) and selects the shorter edge. We moved between them to obtain a new position.

The city on the other end of the selected edge becomes the starting city and the process is repeated, again, a random edge is introduced to avoid a cycle. Our modification of HC uses the following two rules:

- If the shorter of the two edges causes a cycle, then try to take the latter edge.
- If the longer edge also causes a cycle, then choose the shortest of the q randomly selected edges.

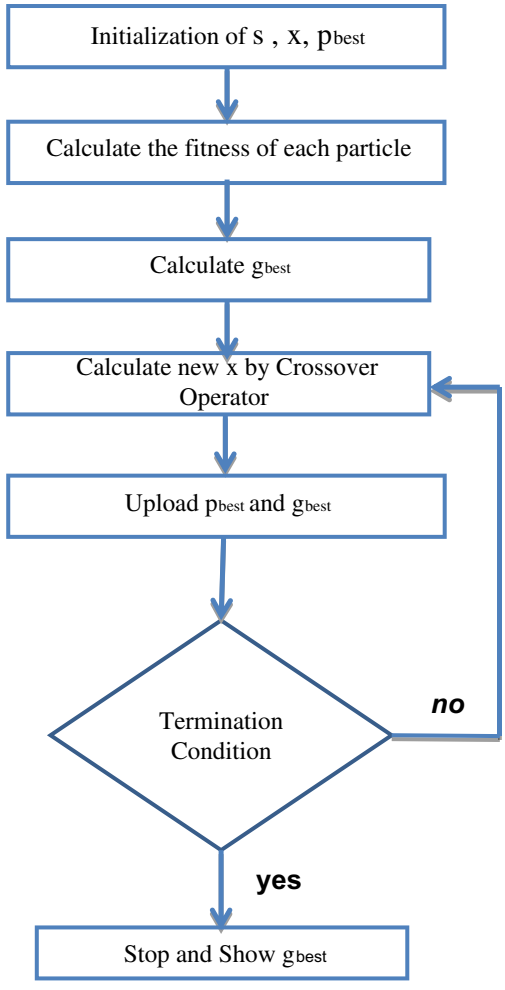


Figure 5. The diagram of our proposed algorithm.

This heuristic tries to combine short sub-paths from the parents. It may leave undesirable crossings of edges.

Algorithm 3: *Steps of our proposed algorithm (MPSO) (see also Figure 5)*

Step 1.

Initialize a swarm of size s

Random position of each particle

For each particle, pbest = x

Step 2.

Calculate the fitness of each particle

Step 3.

Calculate gbest

Step 4.

For each particle, calculate new position:

x = (pbest) © (*gbest*)

//© is the crossover operator (HC) and ***x***

computes the best position (offspring).

Step 5.

Update pbest and gbest:

If (f (x) < f(pbest)) pbest = x;

If (f (pbest) < f (gbest)) gbest = pbest;

Step 6.

Stop iterations if stop condition is verified. Otherwise go to Step 4.

In this algorithm, we first initialize a swarm of size *s*. Next, randomly one position is set for each particle. Then, according to the pbest value, we compute fitness and *gbest* values for each particle. Next, in order to obtain new positions, each particle uses crossover operator which crosses between *gbest* and pbest values. Then for obtaining less distance and to modify *gbest* and pbest, we compare fitness function with *x* and pbest values. The Stop condition occurs when we achieve maximum iterations or the minimum error criterion is attained. Otherwise the algorithm continues to calculate new positions.

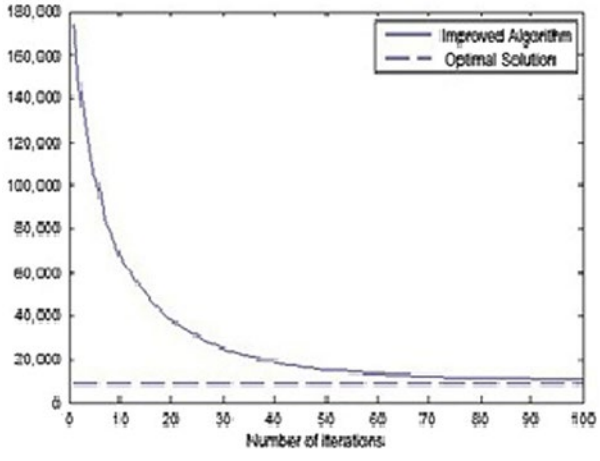


Figure 6. Algorithm converges after about 50 iterations.

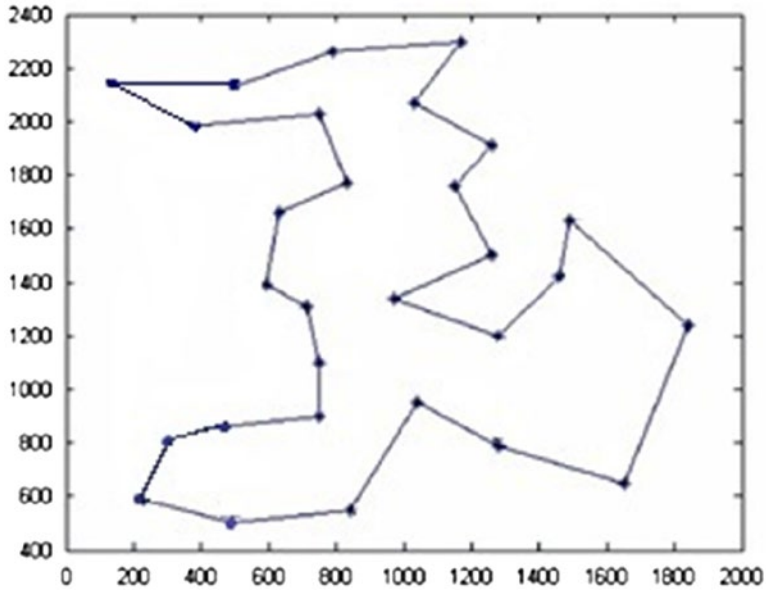


Figure 7. Output for Bays29 instance.

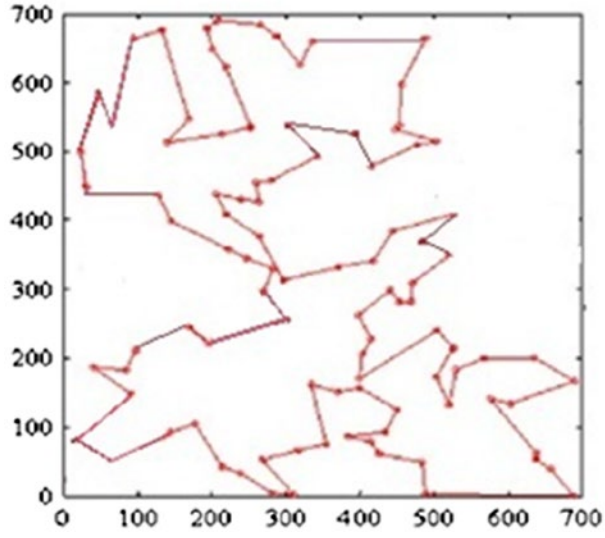


Figure 8. The obtained route from our proposed algorithm.

Table 1. MPSO results

Instance	Instance problem size	Best known TSPLIB	MPSO result
Bays29	29	2,020	2,020
Berlin52	52	7,542	7,540
Dantzig42	42	699	698
Rat99	99	1,211	1,210
Eli76	76	538	538
Pr124	124	59,030	59,028
Fri26	26	937	637

7. Conclusions and experimental results

In this paper, we used a heuristic algorithm based on the PSO method for solving the TSP.

Our algorithm to solve the problem has been inspired by PSO and the crossover operator of GAs. The proposed algorithm is described due to its simplicity and good balance between exploitation and exploration in the search space. The algorithm for solving the problem needs to set some parameters. In this section, our goal is to test and demonstrate the efficiency of our proposed algorithm to deal with other algorithms for TSP as a combinatorial optimization NP-hard problem.

We implemented our algorithm in MATLAB software with 50 bit parameters and maximum 200 iterations. The algorithm was tested with examples of TSP and the obtained values were compared with the solutions obtained by other methods in Table 1. This comparison shows that our algorithm has good performance. Simulation results indicate that PSO algorithm can increase the accuracy of the TSP network response and reduce training time. Table 1 shows the experimental results of the algorithm using the HC on TSPLIB instances. The first column contains the name of the sample, the second column contains the size of cities, for example, the third column and the fourth column show the best results from the TSPLIB and the results from the HC algorithm. Figure 6, shows algorithm converges after about 50 iterations. Figure 7 shows the output, for example Bays29 instance in TSPLIB. Finally, in Figure 8, the obtained route from the proposed algorithm is presented.

Acknowledgments
The authors would like to thank the anonymous referees for their helpful comments which improved the exposition of this paper.

Funding
Keivan Borna was partially supported by a grant from "National Elite Foundation of Iran" dedicated to the elite youth assistant professors [grant number 101048-15-3519].

Author details
Keivan Borna[1]
E-mail: borna@khu.ac.ir
ORCID ID: http://orcid.org/0000-0002-2941-6021
Razieh Khezri[2]
E-mail: raziehkhezri@gmail.com
ORCID ID: http://orcid.org/0000-0003-3807-4577
[1] Faculty of Mathematics and Computer Science, Kharazmi University, Tehran, Iran.
[2] Faculty of Engineering, Kharazmi University, Tehran, Iran.

Citation information
Cite this article as: A combination of genetic algorithm and particle swarm optimization method for solving traveling salesman problem, Keivan Borna & Razieh Khezri, *Cogent Mathematics* (2015), 2: 1048581.

References

Ai, T. J., & Kachitvichyanukul, V. (2008, November). Recent advances in adaptive particle swarm optimization algorithms. In *Proceedings of the Korea Institute of Industrial Engineering Conference*. Seoul.

Arumugam, M. S., & Rao, M. V. C. (2008). On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems. *Applied Soft Computing, 8*, 324–336. http://dx.doi.org/10.1016/j.asoc.2007.01.010

Chen, W., Zhang, J., Chung, S. H., Zhong, W., Wu, W., & Shi, Y. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation, 14*, 278–300.

Gupta, S., & Panwar, P. (2013). Solving travelling salesman problem using genetic algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*, 376–380.

Hu, X. (2006). *PSO tutorial*. Retrieved from www swarmintelligence.org/tutorials.php

Kachitvichyanukul, V. (2012). Comparison of three evolutionary algorithms: GA, PSO, and DE. *Industrial Engineering & Management Systems, 11*, 215–223.

Kennedy, J., & Eberhart, R. (2001). *Swarm intelligence* (1st ed.). San Diego, CA: Academic Press.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (Vol. 4, pp. 1942–1948), Perth. Piscataway, NJ: IEEE Service Center.

Labed, S., Gherboudj, A., & Chikhi, S. (2012). A modified hybrid particle swarm optimization algorithm for solving the traveling salesmen problem. *Journal of Theoretical and Applied Information Technology, 39*, 132–138.

Li, L., & Zhang, Y. (2007). An improved genetic algorithm for the traveling salesman problem. *Communications in Computer and Information Science, 2*, 208–216. http://dx.doi.org/10.1007/978-3-540-74282-1

Liao, Y. F., Yau, D. H., & Chen, Ch. L. (2012). Evolutionary algorithm to traveling salesman problems. *Computers and Mathematics with Applications, 64*, 788–797. http://dx.doi.org/10.1016/j.camwa.2011.12.018

Lin, S., & Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*, 498–516. http://dx.doi.org/10.1287/opre.21.2.498

Pang, S., Li, T., Dai, F., & Yu, M. (2013). Particle swarm optimization algorithm for multi-salesman problem with time and capacity constraints. *Applied Mathematics & Information Sciences, 6*, 2439–2444.

Pang, W., Wang, K. P., Zhou, Ch. G., & Dong, L. J. (2004). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In *Proceedings of the Fourth International Conference on Computer and Information Technology (CIT04)*. Jilin.

Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., & Wang, Q. X. (2007). Particle swarm optimization-based algorithms for TSP and generalized TSP. *Science Direct, 103*, 169–176.

Tasgetiren, M. F., Suganthan, P. N., & Pan, Q. K. (2007, July). A discrete particle swarm optimization algorithm for the

generalized traveling salesman problem. In *GECCO07* (pp. 7–11). London.

van Hook, J., Sahin, F., & Arnavut, Z. (2008). *Application of particle swarm optimization for traveling salesman problem to lossless compression of color palette images*. Rochester, NY: Rochester Institute of Technology.

Xu, X. L., Cheng, X., Yang, Z. C., Yang, X., & Wang, W. L. (2013). Improved particle swarm optimization for traveling salesman problem. In *Proceedings 27th European Conference on Modelling and Simulation (ECMS)*. Ålesund.

Yan, X., Zhang, C., Luo, W., Li, W., Chen, W., & Liu, H. (2012). Solve traveling salesman problem using particle swarm optimization algorithm. *International Journal of Computer Science Issues, 9*, 264–271.

Zhang, D., Guan, Z., & Liu, X. (2007). An adaptive particle swarm optimization algorithm and simulation. *Proceedings of IEEE International Conference on Automation and Logistics, 2007*, 2399–2402.

***Cogent Mathematics* (ISSN: 2331-1835) is published by Cogent OA, part of Taylor & Francis Group.**