4

Particle Swarm Optimization Algorithm for the Traveling Salesman Problem

Elizabeth F. G. Goldbarg, Marco C. Goldbarg and Givanaldo R. de Souza
Universidade Federal do Rio Grande do Norte
Brazil

1. Introduction

Particle swarm optimization, PSO, is an evolutionary computation technique inspired in the behavior of bird flocks. PSO algorithms were first introduced by Kennedy & Eberhart (1995) for optimizing continuous nonlinear functions. The fundamentals of this metaheuristic approach rely on researches where the movements of social creatures were simulated by computers (Reeves, 1983; Reynolds, 1987; Heppner & Grenander, 1990). The research in PSO algorithms has significantly grown in the last few years and a number of successful applications concerning single and multi-objective optimization have been presented (Kennedy& Eberhart, 2001; Coello et al., 2004). This popularity is partially due to the fact that in the canonical PSO algorithm only a small number of parameters have to be tuned and also due to the easiness of implementation of the algorithms based on this technique. Motivated by the success of PSO algorithms with continuous problems, researchers that deal with discrete optimization problems have investigated ways to adapt the original proposal to the discrete case. In many of those researches, the new approaches are illustrated with the Traveling Salesman Problem, TSP, once it has been an important test ground for most algorithmic ideas.

Given a graph $G = (N,E)$, where $N = \{1,...,n\}$ and $E = \{1,...,m\}$, and costs, c_{ij}, associated with each edge linking vertices i and j, the TSP consists in finding the minimal total length Hamiltonian cycle of G. The length is calculated by the summation of the costs of the edges in the considered cycle. If for all pairs of nodes {i,j}, the costs c_{ij} and c_{ji} are equal then the problem is said to be symmetric, otherwise it is said to be asymmetric. The main importance of TSP regarding applicability is due to its variations, nevertheless some applications of the basic problem in real world problems are reported for different areas such as VLSI chip fabrication, X-ray crystallography, genome map and broadcast schedule, among others.

Although, a great research effort has been done to accomplish the task of adapting PSO to discrete problems, many approaches still obtain results very far from the best results known for the TSP. Some of those works are summarized in section 2.

An effective PSO approach for the TSP is presented by Goldbarg et al. (2006a), where distinct types of velocity operators are considered, each of them concerning one movement the particles are allowed to do. This proposal is presented and extended in this chapter, where search strategies for Combinatorial Optimization problems are associated with the velocity operators. Rather than a metaheuristic technique, the PSO approach in this context

Source: Travelling Salesman Problem, Book edited by: Federico Greco, ISBN 978-953-7619-10-7, pp. 202, September 2008, I-Tech, Vienna, Austria

can be thought as a framework for heuristics hybridization. The extension of the approach proposed previously comprehends methods to combine the distinct velocity operators. Computational experiments with a large set of benchmark instances show that the proposed algorithms produce high quality solutions when compared with effective heuristics for the TSP.

The chapter begins with a brief review of Particle Swarm Optimization. Some proposals for applying this metaheuristic technique to discrete optimization problems and, in particular, to the Traveling Salesman Problem are presented in section 2. In section 3, our proposal for velocity operators in the discrete context is presented. Computational experiments compare the results of the proposed approach with other PSO heuristics presented previously for the TSP. In section 4, the combination of velocity operators is investigated. Conclusions and directions for future works are presented in sections 5 and 6, respectively.

2. Particle swarm optimization

Kennedy & Eberhart (1995) proposed the bio-inspired PSO approach, which can be seen as a population-based algorithm that performs a parallel search on a space of solutions. In the optimization context, several solutions of a given problem constitute a population (the swarm). Each solution is seen as a social organism, also called particle. The method attempts to imitate the behavior of real creatures making the particles "fly" over a solution space. These particles search the problem's solution space balancing the intensification and the diversification efforts. Each particle has a value associated with it. In general, particles are evaluated with the objective function of the considered optimization problem. A velocity is also assigned to each particle in order to direct the "flight" through the problem's solution space. The artificial creatures have a tendency to follow the best ones among them. At each iteration step, a new velocity value is calculated for each particle. This velocity value is used to update the particle's position. The process iterates until reaching a stopping condition.

In the classical PSO algorithm, each particle

- has a position and a velocity
- knows its own position and the value associated with it
- knows the best position it has ever achieved, and the value associated with it
- knows its neighbors, their best positions and their values

The best position a given particle has ever achieved is called *pbest*. In some versions of particle swarm algorithms the particles also track the best position achieved so far by any particle of the swarm. This position is called *gbest*. By changing their velocities with individualistic moves or toward *pbest* and *gbest*, the particles change their positions. The move of a particle is a composite of three possible choices (Onwubolu & Clerc, 2004):

- To follow its own way
- To go back to its best previous position
- To go towards its best neighbor's previous or present position

The neighborhood may be physical or social. Physical neighborhoods take distances into account, thus a distance metric has to be established. This approach tends to be time consuming, since each iteration distances must be computed. In general, social neighborhoods are based upon "relationships" defined at the very beginning of the algorithm.

A general framework of a particle swarm optimization algorithm is presented in figure 1. Initially, a population of particles is generated. After, all particles are evaluated and, if

necessary, $pbest_p$ is replaced by x_p, p's position. The best position achieved so far by any of the p's neighbors is set to $gbest_p$. Finally, the velocities and positions of each particle are updated. The procedure *compute_velocity*( ) receives three inputs. This is done to show that, in general, p's position, x_p, $pbest_p$ and $gbest_p$ are used to update p's velocity, v_p. The process is repeated until some stopping condition is satisfied.

```
procedure PSO
   Initialize a population of particles
   do
      for each particle p with position x_p do
         if (x_p is better than pbest_p) then
            pbest_p ← x_p
         end_if
      end_for
      Define gbest_p as the best position found so far by any of p's neighbors
      for each particle p do
         v_p ← Compute_velocity(x_p, pbest_p, gbest_p)
         x_p ← update_ position(x_p, v_p)
      end_for
   while (a stop criterion is not satisfied)
```

Fig. 1. Framework of a particle swarm optimization algorithm

Kennedy & Eberhart (1995) suggest equations (1) and (2) to update the particle's velocity and position, respectively. In these equations, $x_p(t)$ and $v_p(t)$ are the particle's position and velocity at instant t, $pbest_p(t)$ is the best position the particle achieved up to instant t, $gbest_p(t)$ is the best position that any of p's neighbors has achieved up to instant t, c_1 is a cognitive coefficient that quantifies how much the particle trusts its experience, c_2 is a social coefficient that quantifies how much the particle trusts its best neighbor, $rand_1$ and $rand_2$ are random numbers.

$$v_p(t) = v_p(t\text{-}1) + c_1.rand_1.(pbest_p(t\text{-}1) - x_p(t\text{-}1)) + c_2.rand_2.(gbest_p(t\text{-}1) - x_p(t\text{-}1)) \quad (1)$$

$$x_p(t) = x_p(t\text{-}1) + v_p(t) \quad (2)$$

An inertia factor is introduced in equation (1) by Shi & Eberhart (1998). Considering the inertia factor w, equation (3) replaces equation (1). The inertia factor multiplies the velocity of the previous iteration. It is decreased throughout the algorithm execution. The inertia factor creates a tendency for the particle to continue moving in the same direction it was going previously. The motivation for the use of the inertia factor was to be able to better control intensification and diversification. Shi & Eberhart (1998) observed that suitable values for the inertia factor yielded a good trade-off between exploration and exploitation.

$$v_p(t) = wv_p(t\text{-}1) + c_1.rand_1.(pbest_p(t\text{-}1) - x_p(t\text{-}1)) + c_2.rand_2.(gbest_p(t\text{-}1) - x_p(t\text{-}1)) \quad (3)$$

Constriction factors were introduced by Clerc (1999) who observed that the use of a constriction factor was necessary to insure the convergence of the PSO algorithm. A simple way to incorporate a constriction factor in PSO algorithms is to replace equation (1) by equations (4) and (5), where K is the constriction factor. In equation (5), c_1 and c_2 are usually set to 1.49445 (Eberhart & Shi, 2001).

$$v_p(t) = K[v_p(t\text{-}1) + c_1.rand_1.(pbest_p(t\text{-}1) - x_p(t\text{-}1)) + c_2.rand_2.(gbest_p(t\text{-}1) - x_p(t\text{-}1))] \quad (4)$$

$$K = \frac{2}{2 - \alpha - \sqrt{\alpha^2 - 4\alpha}}, \quad \alpha = c_1 + c_2, \quad \alpha > 4 \quad (5)$$

The canonical PSO algorithm, however, needs an adaptation in order to be applied to discrete optimization problems. Kennedy & Eberhart (1997) propose a discrete binary PSO version, defining particles' trajectories and velocities in terms of changes of probabilities that a bit is set to 0 or 1 (Shi et al., 2007). The particles move in a state space restricted to 0 and 1 with a certain probability that is a function of individual and social factors. The probability of $x_p(t) = 1$, $Pr(x_p = 1)$, is a function of $x_p(t\text{-}1)$, $v_p(t\text{-}1)$, $pbest_p(t\text{-}1)$ and $gbest_p(t\text{-}1)$. The probability of $x_p(t) = 0$ equals 1 - $Pr(x_p = 1)$. Thus equation (2) is replaced by equation (6), where $rand_3$ is a random number, $\psi(v_p(t))$ is a logistic transformation which can constrain $v_p(t)$ to the interval [0,1] and can be considered as a probability.

$$x_p(t) = \begin{cases} 1, \ if \ \ rand_3 < \psi(v_p(t)) \\ 0, \ otherwise \end{cases} \quad (6)$$

PSO for permutation problems is investigated by several researchers. In several of these research works the TSP is the target problem.

Hu et al. (2003) define velocity as a vector of probabilities in which each element corresponds to the probability of exchanging two elements of the permutation vector that represents a given particle position. Pairwise exchanging operations, also called 2-swap or 2-exchange, are very popular neighborhoods in local search algorithms for permutation problems. Let V be the velocity of a particle whose position is given by the permutation vector P. Given integers i and j, $V[i]$ is the probability of elements $P[i]$ and $P[j]$ be exchanged. The element $P[j]$ corresponds to $P_{nbest}[i]$, where P_{nbest} is the vector that represents the permutation associated with the position of the best neighbor of the considered particle. The authors introduce a *mutation* operator in order to avoid premature convergence of their algorithm. The mutation operator does a 2-swap move with two elements chosen at random in the considered permutation vector.

Another approach is proposed by Clerc (2004) that utilizes the Traveling Salesman Problem to illustrate the PSO concepts for discrete optimization problems. In the following we list the basic ingredients Clerc (2004) states that are necessary to construct a PSO algorithm for discrete optimization problems:

- a search space, $S = \{s_i\}$
- an objective function f on S, such that $f(s_i) = c_i$
- a semi-order on $C = \{c_i\}$, such that for every c_i, $c_j \in C$, we can establish whether $c_i \geq c_j$ or $c_j \geq c_i$
- a distance d in the search space, in case we want to consider physical neighborhoods.

S may be a finite set of states and f a discrete function, and, if it is possible to define particles' positions, velocity and ways to move a particle from one position to another, it is possible to use PSO. Clerc (2004) presents also some operations with position and velocity such as: the opposite of a velocity, the addition of position and velocity (move), the subtraction of two positions, the addition and subtraction of two velocities and the multiplication of velocity by a constant. A distance is also defined to be utilized with physical neighborhoods. To illustrate his ideas about tackling discrete optimization

problems with PSO, Clerc (2004) develops several algorithm variants with those operations and methods and applies them to the asymmetric TSP instance br17.atsp. In his algorithm the positions are defined as TSP tours represented in vectors of permutations of the $|N|$ vertices of the graph correspondent to the considered instance. These vertices are also referred as cities, and the position of a particle is represented by a sequence (n_1, …, $n_{|N|}$, $n_{|N|+1}$), $n_1 = n_{|N|+1}$. The value assigned to each particle is calculated with the TSP objective function, thus corresponding to the tour length. The velocity is defined as a list of pairs (i,j), where i and j are the indices of the elements of the permutation vector that will be exchanged. This approach was applied to tackle the real problem of finding out the best path for drilling operations (Onwubolu & Clerc, 2004) .

Wang et al. (2003) present a PSO algorithm for the TSP utilizing, basically, the same structure proposed by Clerc (2004) and apply their algorithm to the benchmark instance burma14.

Hendtlass (2003) proposes the inclusion of a memory for the particles in order to improve diversity. The memory of each particle is a list of solutions (target points) that can be used as an alternative for the current local optimal point. There is a probability of choosing one of the points of the particle's memory instead of the current $gbest_p$. The size of the memory list and the probability are new parameters added to the standard PSO algorithm. The algorithm is applied to the benchmark TSP instance burma14. The results obtained with algorithmic versions with several parameter settings are compared with the results of an Ant Colony Optimization algorithm. The author shows that his algorithm outperformed the PSO version without the use of memory and presented quality of solution comparable to the results produced by the ACO algorithm, for instance burma14.

Pang et al. (2004a) extend the work of Wang et al. (2003). Their algorithm alternates among the continuous and the discrete (permutation) space. $|N|$-dimensional vectors in the continuous Cartesian space are used for positions and velocities. The discrete representation of the particles' positions is done in the permutation space. They present methods to transform the positions from one space to the other. They alternate between the two spaces until a stopping condition is reached. The particle's position and velocity are updated in the continuous space. Then, they move to the discrete space, where a local search procedure is applied to all particles' positions. Two local search procedures are tested in their algorithms: the 2-swap and the 2-opt (Flood, 1956). After that, they make the reverse transformation to the continuous space. In order to avoid premature convergence, Pang et al. (2004a) use a chaotic operator. This operator changes randomly the position and velocity in the continuous space, multiplying these vectors by a random number. Four versions of their algorithm are applied to four benchmark instances with 14 to 51 cities: burma14, eil51, eil76 and berlin52. The algorithm variations comprise the presence or not of chaotic variables and the two local search procedures. In the set of instances tested, the results showed that the version that includes chaotic variables and the 2-opt local search presented the best results.

Pang et al. (2004b) present a fuzzy-based PSO algorithm for the TSP. The position of each particle is a matrix $P = [p_{ij}]$, where $p_{ij} \in (0,1)$ represents the degree of membership of the i-th city to the j-th position of a given tour. The velocity is also defined as a matrix and the operations resulting from equations (2) and (3) are defined accordingly. A method to decode the matrix position to a tour solution is presented. The value associated with each particle is the length of the tour represented by the particle's position. They apply their algorithm to instances burma14 and berlin52. No average results or comparisons with other algorithms are reported.

A hybrid approach that joins PSO, Genetic Algorithms and Fast Local Search is presented by Machado & Lopes (2005) for the TSP. The positions of the particles represent TSP tours as permutations of $|N|$ cities. The value assigned to each particle (fitness) is the rate between a constant D_{min} and the cost of the tour represented in the particle's position. If the optimal solution is known, then D_{min} equals the optimal tour cost. If the optimum is not known, D_{min} is set to 1. Velocity is defined regarding only $pbest_p$ and $gbest_p$ and the equation of velocity is reduced to equation (7). The distance between two positions is calculated with a version of the Hamming distance for permutations. With the use of equation (7) for velocity, the particles tend to converge to $pbest_p$ and $gbest_p$. At each iteration step, the average distance between all particles and the best global solution is computed. If this distance is lower than $0.05|N|$, then random positions are generated for all particles. The same occurs when some subset of particles is close enough. If a subset of particles is close enough to the best local solution, then the positions of the particles of the considered subset are generated randomly. The solutions are recombined by means of the OX operator and then submitted to the fast local search procedure introduced by Voudouris & Tsang (1999). The hybrid PSO is applied to the following symmetric TSP benchmark instances: pr76, rat195, pr299, pr439, d657, pr1002, d1291, rl1304, d2103.

$$v_p(t) = c_1.rand_1.(pbest_p(t\text{-}1) - x_p(t\text{-}1)) + c_2.rand_2 .(gbest_p(t\text{-}1) - x_p(t\text{-}1)) \quad (7)$$

Goldbarg et al. (2006a) present a PSO algorithm for the TSP where the idea of distinct velocity operators is introduced. The velocity operators are defined according to the possible movements a particle is allowed to do. In the previous section, three alternatives for movements are identified. The three alternatives can be divided into two categories: independent and dependent moves. The independent move concerns the first parcel of equations (1) and (3). The other two parcels of those equations depend on $pbest_p$ and $gbest_p$, thus referring to dependent moves. Based on those movement classes, Goldbarg et al. (2006a) use local search procedures as velocity operators for independent moves and path-relinking (Glover et al., 2000) for dependent moves. At each iteration step, one of the three alternative moves is assigned to a particle and the correspondent velocity operator is applied in order to modify the particles position. For each particle, only one type of movement is allowed per iteration. A probability is assigned to each movement alternative. Initially, independent moves are more likely to occur than dependent moves. During the algorithm execution, the probabilities are modified, such that the probabilities assigned to the dependent moves are increased and the probability assigned to independent moves is decreased. This algorithmic proposal obtained very promising results. It was applied to 35 benchmark TSP instances with 51 to 7397 cities. The results were comparable to the results of state-of-the-art algorithms for the TSP. A detailed discussion of this approach and the results it obtained is presented in section 3.

Yuan et al. (2007) and Shi et al. (2007) propose extensions for the approach presented by Wang et al. (2003). Both algorithms define subtraction in terms of sequences of 2-swap operations as defined in the path-relinking velocity operator presented by Goldbarg et al. (2006a), including some uncertainty for the exchange of two elements.

Yuan et al. (2007) propose new concepts for "chaos variables" and memory for particles. The memory of each particle is an $|N|$-dimensional vector of chaos variables. The chaos variables are numbers in the interval (0,1) and are generated with a method proposed by the authors. Based on the memory list of a particle p, they define the permutation that

represents p's position. They sort the elements of the memory list. The resulting order leads to a permutation of the elements in the memory list. This permutation is the representation of p's position. They apply their algorithm to four benchmark instances with 14 to 51 cities: burma14, oliver30, att48, eil51. The results obtained for instances oliver30 and att48 are compared with the results obtained by algorithms based on: Simulated Annealing, Genetic Algorithm and Ant Colony Systems. Their algorithm outperforms the others regarding quality of solution of these two instances.

Shi et al. (2007) adds to their algorithm a procedure that aims at eliminating edge crossings in the TSP tours represented by the particles' positions. They apply their algorithm to five benchmark instances: eil51, berlin52, st70, eil76 and pr70.

Zhong et al. (2007) present a PSO approach where a mutation factor (c_3) is introduced in the formula that updates the particle's position (equation (2)). The new formula is presented in equation (8). The factor introduces some diversity in the algorithm. The position of a particle is represented as a set of edges instead of a permutation as in the previous approaches. The velocity is defined as a list of edges with a probability associated with each element of the list. During the iterations if $pbest_p$ is identical to $gbest_p$ then, $pbest_p$ is not replaced by the current position of p. The authors apply their algorithm to six benchmark TSP instances: burma14, eil51, eil76, berlin52, kroA100 and kroA200. The results are compared with the results of Pang et al. (2004a) and with an Ant Colony Optimization algorithm. They show that their algorithm outperforms the others regarding average solutions.

$$x_p(t) = c_3.rand.x_p(t\text{-}1) + v_p(t) \tag{8}$$

Fang et al. (2007) present a PSO algorithm for the TSP where an annealing scheme is used to accept the movement of a particle. They apply their algorithm to instances oliver30 and att48. The results are compared with the results of algorithms based on: Simulated Annealing, Genetic Algorithms and Ant Colony. In the two instances tested, their algorithm presents the best average results.

A comparison among some of the previous algorithms and the approach proposed in this chapter is presented in the next section.

3. New velocity operators for discrete PSO

In PSO algorithms the velocity is the basic mechanism for accomplishing the search in the space of solutions of optimization problems. In most applications, the particles' positions represent the solutions of the investigated problem. The positions are updated by means of velocity operators that direct the search to promising regions of the space of solutions. There are two classes of movement a particle is allowed to do: independent and dependent moves. Independent moves are those in which the particle moves without knowing any other positions besides its own on the current instant. This type of movement depends only on the current particle position and on a velocity operator. The other case arises when the particle needs to know the position of *pbest* or *gbest*. This distinction between the movements leads us to a unary and a binary concept for velocity operators. In the unary operations only one particle is accepted as input. The particle's position is transformed according to a unary velocity operator. The binary operations accept two particles and alter the position of one of them considering the position of the other. In this context, m-ary operations can be defined where m particles are accepted and the position of one of them is altered considering the positions of the remaining m-1 particles, in accordance with an m-ary velocity operator.

In order to modify the position of a given particle, the velocity operators are identified with heuristic methods. Basically, two approaches are utilized for designing the search strategies: the improvement methods and the metaheuristic techniques. As defined by Burkard (2002), the local search algorithms constitute the class of improvement methods. Given a neighborhood structure defined over a search space, a local search procedure begins with a solution and search the neighborhood of the current solution for an improvement. The metaheuristics are general frameworks for heuristics design. A review of the TSP and some well known methods utilized to solve it are presented by Gutin & Punnen (2002).

In this chapter, any search strategy where a given solution is transformed with no knowledge of other solutions is a unary velocity operator. Search strategies where a solution interacts with other m-1 solutions are classified as m-ary velocity operators. For example, local search and mutation are defined as unary velocity operators, recombination of two solutions, such as crossover in Genetic Algorithms, and path-relinking are defined as binary velocity operators and recombination operations among m solutions, such as in Scatter Search algorithms (Glover et al., 2000), are defined as m-ary velocity operators.

The proposed approach is illustrated with unary and binary velocity operators utilizing local search and path-relinking strategies, respectively.

Path-relinking is an intensification technique which ideas were originally proposed by Glover (1963) in the context of methods to obtain improved local decision rules for job shop scheduling problems (Glover et al., 2000). The strategy consists in generating a path between two solutions creating new intermediary solutions. This idea is very close to the movement of a particle from one position to another. Given an origin solution, x_1, and a target solution, x_2, a path from x_1 to x_2 leads to a sequence x_1, x_1 (1), x_1 (2), …, x_1 (r) = x_2, where $x_1(i+1)$ is obtained from $x_1(i)$ by a move that introduces in $x_1(i+1)$ an attribute that reduces the distance between attributes of the origin and target solutions.

The framework of PSO for discrete optimization problems proposed by Goldbarg et al. (2006a, 2006b) is shown in figure 2. In this proposal equation (3) is replaced by equation (9), where v_1 is a unary velocity operator, v_2 and v_3 are binary velocity operators. The coefficients c_0, c_1 and c_2 have the same meaning stated previously and the signal $\oplus$ represents a composition.

$$v_p(t) = c_0 v_1(x_p(t\text{-}1)) \oplus c_1 v_2(pbest_p(t\text{-}1), x_p(t\text{-}1)) \oplus c_2 v_3(gbest_p(t\text{-}1), x_p(t\text{-}1)) \quad (9)$$

In initial applications of the proposed approach, only one of the three primitive moves is associated to each particle of the swarm at each iteration step (Goldbarg et al., 2006a, 2006b). Thus, c_0, c_1, $c_2 \in \{0,1\}$ and $c_0 + c_1 + c_2 = 1$ in equation (9). The assignment is done randomly. Initial probabilities are associated with each possible move and, during the execution, these probabilities are updated. Initially, a high value is set to pr_1, the probability of particle p to follow its own way, a lower value is set to pr_2, the probability of particle p goes towards $pbest_p$ and the lowest value is associated with the third option, to go towards $gbest_p$. The algorithm utilizes the concept of social neighborhood and the $gbest_p$ of all particles is associated with the best current solution, *gbest*. The initial values set to pr_1, pr_2 and pr_3 are 0.9, 0.05 and 0.05, respectively. As the algorithm runs, pr_1 is decreased and the other probabilities are increased. At the final iterations, the highest value is associated with the option of going towards *gbest* and the lowest probability is associated with the first move option.

```
procedure Discrete_PSO
    /* Define initial probabilities for particles' moves:*/
    pr1 ← a1 /*to follow its own way*/
    pr2 ← a2 /*to go towards pbest*/
    pr3 ← a3  /*to go towards gbest*/
    /* a1+ a2+ a3=1 */
    Initialize the population of particles
    do
      for each particle p
          valuep ← Evaluate(xp)
          if (value(xp) < value(pbestp)) then
             pbestp ← xp
          if (value(xp) < value(gbest) ) then
             gbest ← xp
       end_for
       for each particle p
          velocityp ← define_velocity(pr1, pr2, pr3)
          xp ← update(xp,velocityp)
       end_for
       /* Update probabilities*/
       pr1 = pr1×0.95; pr2 = pr2×1.01; pr3 = 1-( pr1+ pr2)
    while (a stop criterion is not satisfied)
```

Fig. 2. Pseudo-code of PSO for discrete optimization problems

In the application to the TSP, Goldbarg et al. (2006a) implement two versions of the PSO algorithm defined by two local search procedures utilized to implement v_1. In the first version a local search procedure based on an inversion neighborhood is used. The Lin-Kernighan (Lin & Kernighan, 1973) neighborhood is used in the second version. In both versions v_2 and v_3 are implemented with the same path-relinking procedure. The particles' positions are represented as permutations of the $|N|$ cities.

In the inversion neighborhood, given a sequence $x_1 = (n_1, \ldots, n_i, n_{i+1}, \ldots, n_{j-1}, n_j, \ldots, n_{|N|})$ and two indices i and j, the sequence x_2 is x_1's neighbor if $x_2 = (n_1, \ldots, n_j, n_{j-1}, \ldots, n_{i+1}, n_i, \ldots, n_{|N|})$. The difference between indices i and j varies from 1 to $|N|$-1. When v_1 is applied to a particle p, the local search procedure starts inverting sequences of two elements in p's position, then sequences of three elements are inverted, and so on.

The Lin-Kernighan neighborhood is a recognized efficient improvement method for the TSP. The basic LK algorithm has a number of decisions to be made and depending on the strategies adopted by programmers distinct implementations of this algorithm may result on different performances. The literature contains reports of many LK implementations with widely varying behavior (Johnson & McGeoch, 2002). The work of Goldbarg et al. (2006a) uses the LK implementation of Applegate et al. (1999).

The path-relinking implemented for the binary velocity operators exchanges adjacent elements of the origin solution. The permutations are considered as circular lists. At first, the origin solution is rotated until its first element be equal the first element of the target solution. Then the second element of the target solution is considered. The correspondent element in the origin solution is shifted left until reaching the second position in the sequence that represents the solution. The process continues until the origin solution reaches

the target solution. This procedure leads to time complexity $O(n^2)$. The path-relinking is applied simultaneously from the origin to the target solution and vice-versa (back and forward). Swap-left and swap-right operations are used. The permutation sequence representing the best solution found replaces the position of the considered particle. An example of the path-relinking procedure is shown in figure 3.

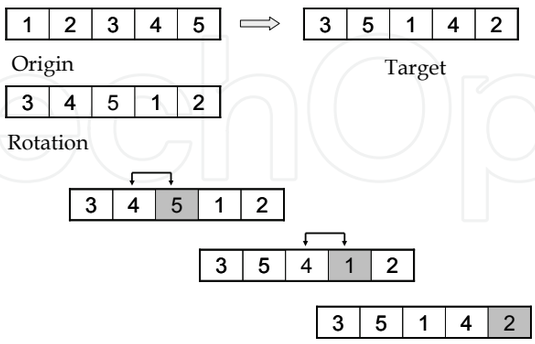


Fig. 3. Path-relinking

In the following, a discussion about the results obtained by PSO proposals for the TSP is presented. PSO-INV and PSO-LK denote the two algorithmic versions of the proposed approach with the inversion and the LK neighborhoods, respectively. These algorithms run on a Pentium IV with 3.0 GHz, 1 Gb using Linux. The maximum processing times are 60 seconds for instances with $|N| < 1000$ and 300 seconds for instances with $1000 \leq |N| < 5000$. Other three stop criteria are used: to find the optimal solution, to reach a maximum number of iterations (200) or to reach a maximum number of iterations with no improvement of the best current solution (20). The population has 20 particles. Once most papers report results for instance eil51, berlin52 and eil76, table 1 shows a comparison between the proposed approach and other PSO algorithms concerning these instances. The compared algorithms are listed in the first column of table 1. The traced lines represent results not reported in the correspondent work. Results in table 1 are given in terms of the percent difference from the optimal solution (*gap*), calculated with equation (10), where *av* and *optimal* denote, respectively, the average solution found by the investigated algorithm and the best solution known for the correspondent instance.

$$gap = \frac{av - optimal}{optimal} \times 100 \tag{10}$$

Only Pang et al. (2004) and Zhong et al. (2007) report average processing times. Pang et al. (2004) use a Pentium IV with 2 GHz, 256 Mb running Windows 2000. Zhong et al. (2007) use a Celeron with 2.26 GHz, 256 Mb, running Windows XP. Running time comparisons are, in general, difficult to make, even when the codes are developed in the same machines and the same compiler options are used. A re-implementation of those algorithms could introduce errors and the results obtained with the new implementations could produce results that differ largely from the published ones. The proposed algorithm was executed in a platform superior than the other algorithms of table 1. Nevertheless, even if the processing times of the other algorithms were divided by a factor of 3 (an estimate that favors those algorithms), table 1 shows that the two versions of the proposed algorithm exhibit processing times significantly lower than the others.

Instance	Algorithm	Min	Average	T(s)
eil51	Pang et al. (2004a)	---	3.498	30
	Shi et al. (2007)	0.235	2.575	---
	Zhang et al. (2007)	---	2.529	---
	Zhong et al. (2007)	0.235	1.793	4.06
	PSO-INV	0.704	2.582	0.16
	PSO-LK	0	0	< 0.01
berlin52	Pang et al. (2004a)	---	2.151	120
	Shi et al. (2007)	0	3.846	---
	Zhong et al. (2007)	0	0.753	4.12
	PSO-INV	0	2.592	0.17
	PSO-LK	0	0	< 0.01
eil76	Pang et al. (2004a)	---	4.222	60
	Shi et al. (2007)	1.487	4.167	---
	Zhong et al. (2007)	0.372	2.550	11.59
	PSO-INV	2.416	4.656	0.40
	PSO-LK	0	0	0.01

Table 1. Results of distinct PSO approaches

Although the inversion neighborhood is not specialized for the TSP, table 1 shows that PSO-INV exhibits better average results than the algorithms of Pang et al. (2004) and Shi et al. (2007) for instances eil51 and berlin52, respectively. Concerning the group of tested instances PSO-INV presents results that are comparable with the results presented by Pang et al. (2004a), Zhang et al. (2007) and Shi et al. (2007). Except for the PSO-LK, the algorithm presented by Zhong et al. (2007) outperforms the others regarding quality of solution. A comparison between the results obtained for instances with more than 50 cities by the PSO-LK and the algorithm presented by Zhong et al. (2007) is shown in table 2. The proposed algorithm outperforms the algorithm of Zhong et al. (2007) regarding quality of solution and processing times in the five tested instances.

Instance	Zhong et al. (2007)			PSO-LK		
	Min	Average	T(s)	Min	Average	T(s)
eil51	0.002	1.793	4.06	0	0	0
berlin52	0	0.753	4.12	0	0	0
eil76	0.004	2.550	11.59	0	0	0.01
kroA100	0.001	1.914	23.95	0	0	0.02
kroA200	0.007	3.427	198.55	0	0	0.08

Table 2. Comparison between PSO-LK and the algorithm of Zhong et al (2007)

PSO-INV performs poorly when compared with PSO-LK. Table 3 presents a comparison, in terms of percent deviation from the optimal solution, between the best and average results found by these two algorithms for 8 instances with 195 to 2103 cities. Table 3 shows that the PSO-LK outperforms PSO-INV with a significant difference among the results reported. This is not a surprise, since the local search procedure embedded in the former version is more powerful than the local search procedure of the latter.

Among the PSO approaches for the TSP, the hybrid algorithm presented by Machado & Lopes (2005) presents results for the largest instances. A comparison between the quality of

solutions obtained by this algorithm (M&L) and the PSO-LK is shown in table 4, where is shown that the proposed approach outperforms the algorithm of Machado & Lopes (2005) in all tested instances. The average differences from the optimal solution obtained by Machado & Lopes (2005) and the PSO-LK regarding the tested instances are, respectively, 3.832 and 0.005.

Instances	PSO-INV		PSO-LK	
	Min	Av	Min	Av
rat195	5.8114	8.7581	0	0
pr299	5.8476	7.9952	0	0
pr439	4.4200	8.0111	0	0
d657	6.9656	9.6157	0	0
pr1002	9.8574	11.1900	0	0
d1291	13.2104	15.5505	0	0.0113
rl1304	10.4432	11.9942	0	0
d2103	16.7383	18.4180	0.0087	0.0267

Table 3. Quality of solutions obtained by the two versions of the proposed algorithm

Instance	M & L	PSO-LK
rat195	0.983	0
pr299	0.590	0
pr439	2.956	0
d657	3.849	0
pr1002	6.699	0
d1291	4.581	0.0113
rl1304	3.245	0
d2103	7.749	0.0267

Table 4. Quality of solutions obtained by Machado & Lopes (2005) and PSO-LK

Although the LK is a powerful neighborhood for the TSP, the good performance exhibited by the PSO-LK is not only due to the use of this neighborhood. The differences between the results obtained by the LK procedure and the PSO-LK algorithm are shown in table 4. This experiment aimed at finding out if the proposed PSO approach was able to improve the LK results. Table 5 shows the results for 30 symmetric instances. The cells with dark background show the results where an improvement with the PSO approach is obtained. Twenty independent runs of each algorithm were performed. Table 5 shows that all average solutions are improved. A statistical analysis shows that, in average, improvements of 88% and 89% were achieved on the best and average results, respectively. The Mann-Whitney U-test was applied to verify if the average solutions are statistically different. The Mann-Withney U-test, also called Mann-Whitney-Wilcoxon test or Wilcoxon rank-sum test, is a non-parametric test used to verify the null hypothesis that two samples come from the same population (Conover, 1971). The p-values obtained are shown in the last column of table 5.
Let av_{LK} and $av_{PSO\text{-}LK}$ denote the average solution obtained by the LK and the PSO-LK algorithms, respectively, then the p-values show that, with a level of significance of 0.05, the null hypothesis that verifies if $av_{LK} = av_{PSO\text{-}LK}$ is rejected for all instances.

Instance	LK		PSO-LK		p-value
	Min	Average	Min	Average	
pr439	0.0000	0.0463	0.0000	0.0000	0.004233
pcb442	0.0000	0.1119	0.0000	0.0000	0.018562
d493	0.0029	0.1216	0.0000	0.0000	0.000000
rat575	0.0295	0.1277	0.0000	0.0052	0.000000
p654	0.0000	0.0078	0.0000	0.0000	0.001932
d657	0.0020	0.1500	0.0000	0.0000	0.000000
rat783	0.0000	0.0704	0.0000	0.0000	0.000000
dsj1000	0.0731	0.2973	0.0027	0.0041	0.000000
pr1002	0.0000	0.1318	0.0000	0.0000	0.000000
u1060	0.0085	0.1786	0.0000	0.0049	0.000000
vm1084	0.0017	0.0669	0.0000	0.0052	0.000000
pcb1173	0.0000	0.1814	0.0000	0.0003	0.000000
d1291	0.0039	0.4333	0.0000	0.0113	0.000000
rl1304	0.0202	0.3984	0.0000	0.0000	0.000000
rl1323	0.0463	0.2300	0.0000	0.0079	0.000001
nrw1379	0.0547	0.1354	0.0018	0.0160	0.000000
fl1400	0.0000	0.1215	0.0000	0.0000	0.000021
fl1577	0.7371	2.2974	0.0000	0.0420	0.000000
vm1748	0.0903	0.1311	0.0000	0.0009	0.000000
u1817	0.1976	0.5938	0.0454	0.1408	0.000000
rl1889	0.1836	0.3844	0.0000	0.0165	0.000000
d2103	0.0597	0.3085	0.0087	0.0267	0.000000
u2152	0.2381	0.5548	0.0062	0.1135	0.000000
pr2392	0.0775	0.3904	0.0000	0.0112	0.000000
pcb3038	0.1598	0.2568	0.0123	0.0686	0.000000
fl3795	0.5665	1.0920	0.0000	0.0403	0.000000
fnl4461	0.0882	0.1717	0.0794	0.1155	0.000000
rl5915	0.3528	0.5343	0.0755	0.1554	0.000000
rl5934	0.2221	0.4761	0.0309	0.1545	0.000000
pla7397	0.1278	0.2912	0.0075	0.0253	0.000000

Table 5. Comparison between LK and PSO-LK

4. Composing velocity operators

The composition of velocities can be thought as an arrangement of velocity operators. This arrangement defines the sequence that determines the order of application of each velocity operator to a given particle. For example, let v_1, v_2, v_3 be the three velocity operators defined in the last section, A_1 and A_2 be two sequences of application of these velocity operators, $A_1=(v_1, v_2, v_3)$, $A_2=(v_3, v_1, v_2)$. Regardless the coefficients of equation (9), two examples of algorithms for the composition of the velocities are presented in figures 4(a) and 4(b). The meaning of those coefficients is explained further. In the algorithm shown in figure 4(a), the composition of velocities is implicit, once the results of the application of the velocity operators v_1 and v_2 are inputs for the next operation. A possible implementation for the

composition of velocities with sequence A_2 is illustrated in figure 4(b), where a method to compose the results of each application of the velocity operators has to be defined.

```
procedure update_position(x_p, pbest_p, gbest_p)
    y1 ← v1(x_p)
    y2 ← v2(y1, pbest_p)
    y3 ← v3(y2, gbest_p)
    return(y3)
```

(a)

```
procedure update_position(x_p, pbest_p, gbest_p)
    y1 ← v3(x_p, gbest_p)
    y2 ← v1(x_p)
    y3 ← v2(x_p, pbest_p)
    return(y1 ⊕ y2 ⊕ y3)
```

(b)

Fig. 4. Composition of velocities to update x_p with sequences (a) A_1 and (b) A_2

Besides the six ways to combine velocities v_1, v_2 and v_3, there is, still, the possibility of repeating velocity operators in the same sequence. For example, the sequence $A = (v_1,v_2,v_3,v_1)$ can be implemented with the algorithm of figure 4(a), replacing the statement $return(y_3)$ by the statements $y_4 \leftarrow v_1(y_3)$ and $return(y_4)$.

In order to accomplish the task of composing velocities, stopping conditions for the application of each velocity operator can also be defined. Let $A = (a_1, a_2, \ldots, a_m)$ be a sequence where each a_i, $1 \le i \le m$, is a pair (v_j, s_k), $v_j \in V$, the set of velocity operators, and s_k is v_j's stopping condition. Thus, given a sequence A with q elements, the first velocity operator is applied to particle p until reaching its corresponding stopping condition, then the process continues with the second velocity operator until the q-th element of sequence A.

In this work two velocity operators are considered: local search (v_1) and path-relinking (v_2=v_3). Some stopping conditions that can be adopted for v_1 are: to execute a maximum number of local search iterations, to find a solution that improves the input solution by a given amount, to find a local optimum (corresponds to a standard local search run). Some stopping conditions for v_2 are: to reach the target solution (corresponds to the standard path-relinking), to find a solution better than the origin and target solutions, to find a solution better than the worst among the two input solutions, to stop after a maximum number of iterations, or, given the distance d between the two input solutions, to stop after doing $\lfloor d/z \rfloor$ iterations, where z is an integer $z \le d$.

In this context, the coefficients of equation (9) can be thought as representing stopping conditions. For example, let c_0, c_1, c_2 be three numbers in the interval [0,1] and $itmax_1$, $itmax_2$, $itmax_3$ be the maximum number of iterations for the operations with velocities v_1, v_2 and v_3, respectively. Then $c_i \times v_{i+1}(\cdot)$, i = 0,1,2, represents the application of velocity operator v_{i+1} with a maximum of $c_i \times itmax_{i+1}$ iterations.

Consider the algorithm of figure 2, with the following modifications:

- pr_1, pr_2, pr_3 are the probabilities associated with compositions represented by sequences A_1, A_2 and A_3, respectively.
- The statements

  $velocity_p \leftarrow$ define_velocity(pr_1, pr_2, pr_3)

  $x_p \leftarrow$ update(x_p,$velocity_p$)

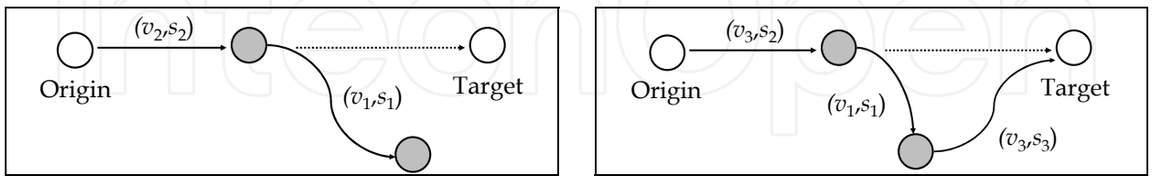
  are replaced by

  $comp_p \leftarrow$ define_composition(pr_1, pr_2, pr_3)

  $x_p \leftarrow$ update(x_p,$comp_p$)

In order to test the potential of composing velocities, two variants of the basic algorithm shown in figure 2 are investigated. The sequences A_1, A_2 and A_3 of the first algorithmic version are: $A_1 = ((v_1,s_1))$, $A_2 = ((v_2,s_2), (v_1,s_1))$, $A_3 = ((v_3,s_2), (v_1,s_1))$. The stopping conditions s_1

and s_2 are, respectively, to find a local optimum and to find a solution better than the worst among the two input solutions. Figure 5(a) shows an illustrative scheme of sequence A_2. Once the path-relinking is considered for v_2 and v_3, the scheme of figure 5(a) is also valid if v_2 is replaced by v_3. In the second variant of the basic algorithm the sequences are: $A_1 = ((v_1,s_1))$, $A_2 = ((v_2,s_2), (v_1,s_1), (v_2,s_3))$, $A_3 = ((v_3,s_2), (v_1,s_1),(v_3,s_3))$. The stopping condition s_3 is to reach the target solution. The illustrative scheme of the sequence A_3 (also valid for A_2) is shown in figure 5(b).



(a) (b)

Fig. 5. Sequences (a) $A_2 = ((v_2,s_2), (v_1,s_1))$ and (b) $A_3 = ((v_3,s_2), (v_1,s_1),(v_3,s_3))$.

Tables 6 and 7 show a comparison between the results obtained by the basic PSO-LK and the first and second algorithmic versions, respectively. The elements of columns *Min* and *Av* are the percent deviation from the best known solution of the best and average solutions found by the correspondent algorithm in 20 independent runs. The average processing times in seconds are presented in column $T(s)$. The cells with the best results have a dark background. The p-values shown in the last column of tables 6 and 7 are the result of the hypothesis test with the average values presented for each instance.

In preliminary experiments the values 10, 15, 20 and 25 were tested for the size of the swarm and the values 20, 50 and 100 were tested for the maximum number of iterations. The best trade-off between quality of solution and processing time was reached with 20 particles and maximum of 20 iterations. The tests were done in a Pentium IV, 3.0 GHz, 1 Gb of RAM.

Table 6 shows that both algorithmic versions find the best average solutions of 10 instances, the PSO-LK finds 1 best solution and the PSO-LK-C1 finds 6 best solutions. Observing the p-values of the 20 instances where different average solutions were found, the table shows that, with a level of significance 0.05, significant differences exist only for instances nrw1379 and pr2392. Thus, both versions present similar performance regarding quality of solution for the majority of the tested instances. Nevertheless, the processing times of the algorithmic version with the composition of velocities are significantly lower than those presented by the basic algorithmic version at 27 instances. The algorithm with the composition of velocities spends, in average, half the processing time spent by the basic algorithm. Thus with half of the processing effort, the algorithm is able to find solutions as good as the basic PSO-LK.

Similar results are observed in table 7. The PSO-LK and the PSO-LK-C2 find the best average solutions of 10 and 11 instances, respectively. Regarding the best solution found by each algorithm, table 7 shows that the PSO-LK-C2 finds 6 best results and the PSO-LK does not find any best result. The p-values of the 21 instances for which the algorithms found different average solutions show that a significant difference exists only for instance pr2392.

In average, the processing times of PSO-LK-C2 are 1.27 times better than the ones presented by the PSO-LK.

Instances	PSO-LK			PSO-LK-C1			p-level
	Min	Av	T(s)	Min	Av	T(s)	
pr439	0	0	0.78	0	0	0.38	-----
pcb442	0	0	0.80	0	0	0.39	-----
d493	0	0	19.38	0	0	13.52	-----
rat575	0	0	6.47	0	0.0007	3.83	0.317318
p654	0	0	1.90	0	0	0.87	-----
d657	0	0	12.42	0	0	8.35	-----
rat783	0	0	5.25	0	0	1.92	-----
dsj1000	0.0027	0.0031	178.48	0.0027	0.0027	82.27	0.077143
pr1002	0	0	9.50	0	0	3.32	-----
u1060	0	0	38.18	0	0.0008	22.87	0.151953
vm1084	0	0.0010	34.74	0	0.0016	25.05	0.958539
pcb1173	0	0.0001	48.18	0	0.0003	32.65	0.156717
d1291	0	0	29.86	0	0	8.81	-----
rl1304	0	0	21.62	0	0	5.57	-----
rl1323	0	0.0092	225.32	0	0.0030	66.60	0.068481
nrw1379	0.0017	0.0085	417.80	0	0.0058	181.75	0.041205
fl1400	0	0	15.42	0	0	5.68	-----
fl1577	0	0.0135	461.99	0	0.0200	248.85	0.237805
vm1748	0	0.0018	854.17	0	0	382.28	0.317318
u1817	0	0.0863	789.18	0.0367	0.1068	410.16	0.297390
rl1889	0	0.0073	894.43	0	0.0037	348.68	0.229728
d2103	0	0.0043	1137.53	0	0.0123	417.53	0.751641
u2152	0	0.0717	1415.32	0	0.0711	512.12	0.989112
pr2392	0	0.0021	577.78	0	0	86.43	0.018578
pcb3038	0.0101	0.0396	323.94	0	0.0343	1772.8	0.336582
fl3795	0	0.0142	621.63	0	0.0214	131.10	0.636875
fnl4461	0.0296	0.0462	583.78	0.0104	0.0421	952.61	0.386402
rl5915	0.0122	0.0633	1359.25	0.0025	0.0435	1029.21	0.083396
rl5934	0.0012	0.0650	983.04	0	0.0797	1443.5	0.645471
pla7397	0.0075	0.0253	1563.22	0.0004	0.0348	826.38	0.158900

Table 6. Comparison between PSO-LK and PSO-LK-C1

Instances	PSO-LK			PSO-LK-C2			p-level
	Min	Av	T(s)	Min	Av	T(s)	
pr439	0	0	0.78	0	0	0.59	----
pcb442	0	0	0.80	0	0	0.6	----
d493	0	0	19.38	0	0	16.3	----
rat575	0	0	6.47	0	0.0007	4.17	0.317318
p654	0	0	1.90	0	0	1.46	----
d657	0	0	12.42	0	0	9.72	----
rat783	0	0	5.25	0	0	3.76	----
dsj1000	0.0027	0.0031	178.48	0.0027	0.0028	103.01	0.097603
pr1002	0	0	9.50	0	0	6.33	----
u1060	0	0	38.18	0	0.0013	26.88	0.075373
vm1084	0	0.0010	34.74	0	0.0016	29.57	0.958539
pcb1173	0	0.0001	48.18	0	0.0003	34.53	0.297961
d1291	0	0	29.86	0	0.0073	27.46	0.152088
rl1304	0	0	21.62	0	0	10.44	----
rl1323	0	0.0092	225.32	0	0.0055	127.55	0.618230
nrw1379	0.0017	0.0085	417.80	0	0.0080	259.99	0.587686
fl1400	0	0	15.42	0	0	11.2	----
fl1577	0	0.0135	461.99	0	0.1144	303.77	0.102963
vm1748	0	0.0018	854.17	0	0	485.22	0.317318
u1817	0	0.0863	789.18	0	0.0811	454.81	0.684114
rl1889	0	0.0073	894.43	0	0.0070	389.12	0.844488
d2103	0	0.0043	1137.53	0	0.0128	443.39	0.655928
u2152	0	0.0717	1415.32	0	0.0609	680.38	0.390349
pr2392	0	0.0021	577.78	0	0	145.84	0.018578
pcb3038	0.0101	0.0396	323.94	0.0036	0.0387	1930.7	0.849722
fl3795	0	0.0142	621.63	0	0.0285	408.86	0.381866
fnl4461	0.0296	0.0462	583.78	0.0148	0.0452	1148.8	0.108256
rl5915	0.0122	0.0633	1359.25	0.0109	0.0499	984.11	0.194137
rl5934	0.0012	0.0650	983.04	0	0.0659	1142.78	0.913724
pla7397	0.0075	0.0253	1563.22	0.0007	0.0298	763.47	0.684311

Table 7. Comparison between PSO-LK and PSO-LK-C2

A comparison between the performance, regarding quality of solution, of PSO-LK-C1 and four effective heuristics for the TSP is shown in tables 8 and 9, where 23 symmetric instances with $|N|$ ranging from 1000 to 7397 are considered. The heuristics are: the Nguyen, Yoshihara, Yamamori and Yasunada iterated Lin-Kernighan variant (reported at http://www.research.att.com/~dsj/chtsp/), ILK-NYYY, the iterated Lin-Kernighan variant presented by Johnson & McGeoch (1997), ILK-JM, the Tourmerge (Cook & Seymour, 2003) and the LK implementation presented by Helsgaun (2000), ILK-H. The results of the first three heuristics were obtained in the DIMACS Challenge page (at http://www.research.att.com/~dsj/chtsp/results.html).

The columns of table 8 corresponding to the ILK-NYYY and the ILK-JM show the best tours obtained in ten $|N|$ iterations runs. The table shows that the PSO-LK-C1 obtains better values than the ILK-NYYY and the ILK-JM at 13 and 16 instances, respectively. The ILK-NYYY presents the best minimal solution for instance dsj1000. The last line of table 8 shows the average results of the three algorithms. It is observed that, in average, the solutions obtained by the PSO-LK-C1 are, approximately, 8 and 24 times better than the solutions presented by the ILK-NYYY and the ILK-JM, respectively.

Instance	PSO-LK-C1	ILK-NYYY Nb10	ILK-JM Nb10
dsj1000	0.0027	0	0.0063
pr1002	0	0	0.1482
u1060	0	0.0085	0.0210
vm1084	0	0.0217	0.0217
pcb1173	0	0	0.0088
d1291	0	0	0
rl1304	0	0	0
rl1323	0	0.01	0
nrw1379	0	0.0247	0.0018
fl1400	0	0	0
fl1577	0	0	0
vm1748	0	0	0
u1817	0.0367	0.1643	0.2657
rl1889	0	0.0082	0.0041
d2103	0	0.0559	0
u2152	0	0	0.1743
pr2392	0	0.0050	0.1495
pcb3038	0	0.0247	0.1213
fl3795	0	0	0.0104
fnl4461	0.0104	0.0449	0.1358
rl5915	0.0025	0.0580	0.0168
rl5934	0	0.0115	0.1723
pla7397	0.0004	0.0209	0.0497
Mean	0.0023	0.0199	0.0569

Table 8. Best solutions of PSO-LK-C1 and two iterative LK

Table 9 shows a comparison between the best and average results found by the PSO-LK-C1, the Tourmerge and the ILK-H. Regarding the minimal values, the proposed algorithm presents better results than the Tourmerge at 6 of the 21 instances the latter algorithm reports results. The Tourmerge presents one minimal result better than the proposed algorithm. The ILK-H presents 4 minimal results that are better than the ones presented by the PSO-LK-C1. Compared with the former, the latter algorithm presents the best minimal results of 2 instances. Considering the average solutions, the PSO-LK-C1 presents better results than the Tourmerge and the ILK-H at 14 and 12 instances, respectively. The Tourmerge and the ILK-H present better average results than the PSO algorithm for 5 and 8 instances, respectively. The last line of table 9 summarizes the results of each column. The proposed algorithm presents the best statistics regarding the average solutions.

Instance	PSO-LK-C1		Tourmerge		ILK-H	
	Min	Average	Min	Average	Min	Average
dsj1000	0.0027	0.0027	0.0027	0.0478	0	0.035
pr1002	0	0	0	0.0197	0	0
u1060	0	0.0008	0	0.0049	0	0
vm1084	0	0.0016	0	0.0013	0	0.007
pcb1173	0	0.0003	0	0.0018	0	0.002
d1291	0	0	0	0.0492	0	0.033
rl1304	0	0	0	0.1150	0	0.019
rl1323	0	0.0030	0.01	0.0411	0	0.018
nrw1379	0	0.0058	0	0.0071	0	0.006
fl1400	0	0	0	0	0	0.162
fl1577	0	0.0200	0	0.0225	0	0.046
vm1748	0	0	0	0	0	0.023
u1817	0.0367	0.1068	0.0332	0.0804	0	0.078
rl1889	0	0.0037	0.0082	0.0682	0	0.002
d2103	0	0.0123	0.0199	0.3170	---	---
u2152	0	0.0711	0	0.0794	0	0.029
pr2392	0	0	0	0.0019	0	0
pcb3038	0	0.0343	0.0036	0.0327	0	0
fl3795	0	0.0214	0	0.0556	0	0.072
fnl4461	0.0104	0.0421	---	---	0	0.001
rl5915	0.0025	0.0435	0.0057	0.0237	0.009	0.028
rl5934	0	0.0797	0.0023	0.0104	0.005	0.089
pla7397	0.0104	0.0348	---	---	0	0.001
Mean	0.002291	0.019926	0.004076	0.046652	0.000636	0.029591

Table 9. Minimal and average results presented by PSO-LK-C1 and Tourmerge

5. Conclusion

This chapter summarized the research done to develop PSO algorithms for the TSP. Many of the PSO algorithms presented previously for the investigated problem do not tackle large instances and present results far from the best known heuristic solutions obtained by effective algorithms. The chapter presented an approach to design effective PSO algorithms

for the TSP that can be extended to other discrete optimization problems. The new approach, first introduced by Goldbarg et al. (2006a), differentiates velocity operators according to the type of move the particle does. Additionally, methods to compose the velocity operators were proposed. Computational experiments with instances up to 7397 cities were presented. The results of those experiments show that the proposed method produces high quality solutions, when compared with four effective heuristics designed specifically for the investigated problem.

The composition of velocities allows building a number of possible implementations for the search strategies chosen to be used in the PSO algorithm. Therefore, rather than a metaheuristic, the Particle Swarm approach can be thought as a framework for heuristics hybridization in the context of discrete optimization problems.

6. Future works

In future works other methods to compose velocities and heuristics hybridization under the PSO framework will be investigated. Another idea to be explored in future researches is variable velocities. The proposed approach will be applied to the Generalized TSP and to the Bi-objective TSP.

7. References

Applegate, D.; Bixby, R.; Chvatal, V. & Cook, W. (1999). *Finding Tours in the TSP*, Technical Report TR99-05, Department of Computational and Applied Mathematics, Rice University

Burkard, R. E. (2002). The traveling salesman problem, In: *Handbook of Applied Optimization*, Pardalos, P.M. & Resende, M.G.C. (Ed.), pp. 616-624, Oxford University Press, ISBN: 0195125940, USA

Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization, *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 1951-1957, ISBN: 0780355369, Washington, DC, June 1999, IEEE

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: *Studies in Fuzziness and Soft Computing New optimization techniques in engineering*, Babu, B.V. & Onwubolu, G.C. (Eds.), Vol. 141, , pp. 219–239, Springer ISBN: 978-3-5402-0167-0, Berlin

Coello, C.A.C.; Pulido, G.T. & Lechuga, M.S. (2004). Handling multiple objectives with particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 256-279, ISSN: 1089-778X

Conover, W.J. (1971). *Practical Nonparametric Statistics*, Wiley, ISBN: 978-0-4711-6068-7, New York

Cook, W.J. & Seymour, P. (2003). Tour merging via branch-decomposition, *INFORMS Journal on Computing*, Vol. 15, pp. 233-248, ISSN: 1091-9856

Eberhart, R.C. & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the 2000 Congress on Evolutionary Computation*, Vol. 1, No. 2, pp. 84-88, ISBN: 0780363752, La Jolla, San Diego, CA , July 2000, IEEE, Piscataway, NJ

Eberhart, R. C. & Shi, Y. (2001) Particle swarm optimization: developments, applications and resources, *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1, pp. 81-86, ISBN: 0780366573, Seoul, South Korea, May 2001, IEEE, Piscataway, NJ

Fang, L.; Chen, P. & Liu, S. (2007). Particle swarm optimization with simulated annealing for TSP, *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, Long, C. A.; Mladenov, V. M. & Bojkovic Z. (Eds.), pp. 206-210, ISBN: 978-9-6084-5759-1, Corfu Island, Greece, February 2007

Flood, M.M. (1956). The traveling-salesman problem, *Operations Research*, Vol. 4, pp. 61-75, ISSN: 0030-364X

Glover, F. (1963). Parametric *Combinations of Local Job Shop Rules*, Chapter IV, ONR Research Memorandum No. 117, Carnegie Mellon University, Pittsburgh

Glover, F.; Laguna, M. & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, Vol. 29, No. 3, pp. 653-684, ISSN: 0324-8569.

Goldbarg, E.F.G; Souza, G.R. & Goldbarg, M.C. (2006a). Particle swarm for the traveling salesman problem, *Proceedings of the EvoCOP 2006*, Gottlieb, J. & Raidl, G.R. (Ed.), Lecture Notes in Computer Science, Vol. 3906, pp. 99-110, ISBN: 3540331786, Budapest, Hungary, April 2006, Springer, Berlin

Goldbarg, E.F.G; Souza, G.R. & Goldbarg, M.C. (2006b). Particle swarm optimization for the bi-objective degree-constrained minimum spanning tree, *Proceedings of the 2006 Congress on Evolutionary Computation*, Vol. 1, pp. 420-427, ISBN: 0780394879, Vancouver, BC, Canada, July 2006, IEEE

Gutin, G. & Punnen, A.P. (2002). *Traveling Salesman Problem and Its Variations*, Kluwer Academic Publishers, ISBN: 0387444599, Dordrecht.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic, *European Journal of Operational Research*, Vol. 126, pp. 106-130, ISSN: 0377-2217

Hendtlass, T. (2003). Preserving diversity in particle swarm optimization, Developments in Applied Artificial Intelligence, *Proceedings of the 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE 2003*, Laughborough, UK, June 2003, In: Lecture Notes in Computer Science, Vol. 2718, pp. 4104-4108, ISBN: 978-3-5404-0455-2, Springer, Berlin

Heppner, F. & Grenander, U. (1990). A stochastic nonlinear model for coordinated bird flocks, In: *The Ubiquity of Caos*, Krasner, S. (Ed.), pp.233-238, ISBN: 0871683504, AAAS Publications, Washington, DC

Hu, X.; Eberhart, R.C. & Shi, Y. (2003). Swarm intelligence for permutation optimization: A case study of n-queens problem, *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pp. 243-246, ISBN: 0780379144, Indianapolis, USA, April 2003, IEEE

Johnson, D. S. & McGeoch, L.A. (1997). The traveling salesman problem: A case study in local optimization, In: *Local Search in Combinatorial Optimization*, Aarts, E.H.L. & Lenstra, J.K., pp. 215-310, ISBN: 978-0-6911-1522-1, John Wiley & Sons, New York

Johnson, D. S. & McGeoch, L.A. (2002). Experimental analysis of heuristics for the STSP, In: *Traveling Salesman Problem and Its Variations*, Guttin, G. & Punnen, A.P. (Eds.), pp. 369-443 , ISBN: 1402006640, Kluwer Academic, Dordrecht

Kennedy, J. & Eberhart, R.C. (1995). Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, pp. 1942-1948, ISBN: 0780327683, Perth, Western Australia November 1995, IEEE

Kennedy, J. & Eberhart, R.C. (1997) A discrete binary version of the particle swarm algorithm, *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 5, No. 2, pp. 4104 - 4109, ISBN: 0780340531, Orlando, Florida, October 1997, IEEE

Kennedy, J. & Eberhart, R.C. (2001) *Swarm Intelligence*, Morgan Kaufmann, ISBN: 1558605959, San Francisco, CA.

Lin, S. & Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem, *Operations Research*, Vol. 21, pp. 498-516, ISSN: 0030-364X

Machado, T.R. & Lopes, H.S. (2005). A hybrid particle swarm optimization model for the traveling salesman problem, In: *Natural Computing Algorithms*, Ribeiro, H.; Albrecht, R.F. & Dobnikar, A. (Eds.), pp. 255-258, ISBN: 3211249346, Springer, Wien

Onwubolu, G.C. & Clerc, M. (2004). Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization, *International Journal of Production Research*, Vol. 42, No. 3, pp. 473–491, ISSN: 0020-7543

Pang, W.; Wang, K.; Zhou, C.; Dong, L.; Liu, M.; Zhang, H. & Wang, J. (2004a) Modified particle swarm optimization based on space transformation for solving traveling salesman problem, In: *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shangai, China, August 2004, pp. 2342-2346, ISBN: 0780384032, IEEE

Pang, W.; Wang, K.; Zhou, C. & Dong, L. (2004b) Fuzzy discrete particle swarm optimization for solving traveling salesman problem, In: *Proceedings of the Fourth International Conference on Computer and Information Technology*, Wuhan, China, September 2004, pp. 796-800, ISBN: 0769522165, IEEE

Reeves, W.T. (1983). Particle systems technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, Vol. 17, No. 3, pp. 359-376, ISSN: 0730-0301

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioural model. *Computer Graphics*, Vol. 21, No. 4, pp. 24-34, ISBN: 0897912276

Shi, Y. & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization, In *Evolutionary Programming VII: Proceedings of Seventh International Conference on Evolutionary Programming - EP98*, San Diego, California, USA, March 1998, Lecture Notes in Computer Science, Vol. 1447, pp. 591-600, ISBN: 3540648917, Springer-Verlag, New York

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, *Information Processing Letters*, Vol. 103, pp. 169-176, ISSN: 0020-0190

Voudouris, C. & Tsang, E. (1999). Guide local search and its application to the traveling salesman problem, *European Journal of Operational Research*, Vol. 113, pp. 469-499, ISSN: 0377-2217

Yuan, Z.; Yang, L.; Wu, Y.; Liao, L. & Li, G. (2007). Chaotic particle swarm optimization algorithm for Traveling Salesman Problem, In: *Proceedings of the IEEE International Conference on Automation and Logistics*, Jinan, China, August 2007, pp. 1121-1124., ISBN: 978-1-4244-1531-1, IEEE

Zhong, W.; Zhang, J. & Che, W. (2007). A novel discrete particle swarm optimization to solve traveling salesman problem, In: *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, Singapore, September 2007, pp. 3283-3287, ISBN: 978-1-4244-1340-9, IEEE

Traveling Salesman Problem

Edited by Federico Greco

ISBN 978-953-7619-10-7
Hard cover, 202 pages
Publisher InTech
Published online 01, September, 2008
Published in print edition September, 2008

The idea behind TSP was conceived by Austrian mathematician Karl Menger in mid 1930s who invited the research community to consider a problem from the everyday life from a mathematical point of view. A traveling salesman has to visit exactly once each one of a list of m cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. Thus, which is the tour of least possible cost the salesman can take? In this book the problem of finding algorithmic technique leading to good/optimal solutions for TSP (or for some other strictly related problems) is considered. TSP is a very attractive problem for the research community because it arises as a natural subproblem in many applications concerning the every day life. Indeed, each application, in which an optimal ordering of a number of items has to be chosen in a way that the total cost of a solution is determined by adding up the costs arising from two successively items, can be modelled as a TSP instance. Thus, studying TSP can never be considered as an abstract research with no real importance.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Elizabeth F. G. Goldbarg, Marco C. Goldbarg and Givanaldo R. de Souza (2008). Particle Swarm Optimization Algorithm for the Traveling Salesman Problem, Traveling Salesman Problem, Federico Greco (Ed.), ISBN: 978-953-7619-10-7, InTech, Available from: http://www.intechopen.com/books/traveling_salesman_problem/particle_swarm_optimization_algorithm_for_the_traveling_salesman_problem

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen