PARTICLE SWARM OPTIMIZATION
APPLIED TO TASK ASSIGNMENT PROBLEM

Jean L. Pierobom, Myriam R. Delgado, Celso A.A. Kaestner
Universidade Tecnológica Federal do Paraná
Curitiba - PR - Brasil
jean@pierobom.com, {myriamdelg, celsokaestner}@utfpr.edu.br

Abstract – Particle Swarm Optimization (PSO) is a metaheuristic inspired on the emerging social behavior found in nature. PSO has shown good results in some recent works of discrete optimization, even though it was originally designed for continuous optimization problems. This paper models the problem of allocating a set of cabs to some customers as a combinatorial problem whose optimization goal is to minimize the distance traveled by the fleet. This problem can be categorized as a Task Assignment Problem, being optimized in this paper by two versions of the discrete PSO: the first approach that is based on a binary codification and the second one which uses permutations to encode the solution. The obtained results show that the second approach (permutation encoding) is superior than the first one (binary encoding) in terms of quality of the solutions and computational time, and it is capable of achieving the known optimal values whenever they are available.

Keywords – Swarm Intelligence, Particle Swarm Optimization, Discrete Optimization, Task Assignment Problem.

1. INTRODUCTION

This paper focuses on the development and application of the Particle Swarm Optimization (PSO) algorithm, an optimization technique developed in 1995 by Kennedy and Eberhart [1], which is based on the analysis of the smart behavior of flocking birds. PSO can be classified as being part of the Swarm Intelligence area, which includes a series of algorithms that simulate the social behavior found in nature and tries to optimize computational problems using these techniques.

Even though the PSO method was originally designed for continuous problems, it has been successfully applied to combinatorial problems, which present great practical applicability and stand among the most common optimization problems. However, the number of applications of the method under such category is still not significant [2]. The first version of the discrete problems algorithm was created by Kennedy and Eberhart in 1997 [3], and since then other combinatorial optimization methods that use PSO have been developed. Some of these methods showed interesting results in problems that have discrete search space, when compared to other bio-inspired optimization such as the Genetic Algorithm (GA) [4–12] and Ant Colony Optimization (ACO) [8, 13] and some well known metaheuristics such as Tabu Search (TS) [6] and Simulated Annealing (SA) [14].

The Task Assignment Problem (TAP) was initially proposed by Tank and Hopfield [15] to illustrate the use of Hopfield networks on combinatorial optimization problems. In TAP there are $N$ tasks that must be accomplished by $N$ workers; each worker performs better some tasks and worse at others; the goal is to optimize the total cost for accomplishing all tasks. There are no algorithms that can find an optimal solution in polynomial time for TAP [4], requiring the development of heuristic search methods to solve it. This paper employs PSO to the optimization of an application of the TAP – named the problem of cabs allocation to cab users – using two discrete implementations: one based on a binary codification and the second one that uses permutations of position sequences. We conduct a series of experiments in order to evaluate our proposal. The obtained results of both PSO-based approaches are compared between each other and with the optimal solutions provided by an exhausting search.

2. PROBLEM FORMULATION

Organization and planning of transportation systems are fundamental topics not only for the fine operation of big cities, but also for the success of events such as the FIFA World Cup and the Olympic Games [16], which Brazil will respectively host in 2014 and 2016. Taxi cabs are an important way of urban transportation and their optimized allocation along with a smaller number of cabs per inhabitant can bring benefits and improvements in terms of urban mobility and reduction in the emission of pollutant gases.

The Cab-Customer Allocation Problem (CCAP) consists in allocating $N$ cabs (service offer agents) to serve $N$ customers (demand service agents) in such a way that the total distance traveled by the cabs to get to the customers is minimal. The searching space $S$ for this problem is the set of different allocation combinations that can be formed, with dimension $|S| = N!$. For example, in a scenario with 10 cabs and 10 customers, there are 3.628.800 possible allocation solutions. The evaluation of each of these solutions by enumeration is infeasible for high $N$ values, since the dimension of the searching space of the problem enlarges exponentially. The CCAP, categorized in this paper as a TAP problem, can be formally defined as follows: let $A$ be the allocation function that maps the set $V$ of service offer agents to the set $P$ of demand service agents; we are considering the case where $|V| = |P| = N$.

$$A : V \rightarrow P$$

(1)
where \( A(i) = j \) if the offer agent \( i \) is allocated to demand agent \( j \). Let \( C(A) \) be the cost function of a solution \( A \):

\[
C(A) = \sum_{i=1}^{n} \text{distance}(i, A(i))
\]

where \( \text{distance}(i, j) \) is the geographic distance between two points in the city; in this case, the distance between the agents \( i \) and \( j \), and \( j = A(i) \). In this paper, \( C(A) \) is calculated as the Euclidean distance between a cab and a customer in the city map disregarding the streets and other urban traffic characteristics. The problem is to find the optimal solution \( A^* \) with minimal cost in the set \( \Omega \), i.e.:

\[
A^* = \arg \min A \in \Omega C(A)
\]

This paper does not consider the dynamic aspects of the problem, and the optimization occurs at a fixed time where there are some cabs that must be allocated to some demanding customers.

3. PARTICLE SWARM OPTIMIZATION

The PSO population, called cloud (or swarm), is composed by particles that are candidate solutions to the problem. Drawing an analogy with the flocks of birds, each particle acts as a bird from the flock looking for food. A swarm particles system begins the process of optimization with a population of random solutions, and searches for the optimal solution by updating the potential solutions through the iterations, similarly to Genetic Algorithms (GA) [17]. However, PSO doesn’t have mutation operators and crossovers like the GA’s. Instead, the particles “fly” over the searching area looking for better solutions [5]. Differently from the GA’s, whose solutions act in a competitive way to perpetuate their characteristics in the next generation, the PSO solutions cooperate among themselves and look for what’s called an optimal solution [18]. One of the reasons why PSO is an attractive solution for optimization problems is that little effort is demanded for parametrization, once a version of the algorithm with few adjustments can present a wide applicability [5].

As can be seen in Algorithm 1, the cloud is initialized through the random distribution of the particles in the searching space. Afterward, an interactive process begins, such as the position of every particle is changed according to its velocity, making the particle to move around the search space looking for a better solution. The velocity of each particle depends on its previous adjustments can present a wide applicability [5].

As can be seen in Algorithm 1, the cloud is initialized through the random distribution of the particles in the searching space. Afterward, an interactive process begins, such as the position of every particle is changed according to its velocity, making the particle to move around the search space looking for a better solution. The velocity of each particle depends on its previous position and velocity are considered as vectorial quantities in the search space

\[
\text{fitness}(x_k) = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

\[
v_k^t = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

in which \( w \) is the inertia factor that forces the particle to move in the same direction of the previous iteration, \( c_1 \) is the cognitive factor that indicates the self-confidence of the particle, \( c_2 \) is the social factor that forces the particle to follow the same way of

\[
\text{fitness}(x_k) = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

\[
v_k^t = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

in which \( w \) is the inertia factor that forces the particle to move in the same direction of the previous iteration, \( c_1 \) is the cognitive factor that indicates the self-confidence of the particle, \( c_2 \) is the social factor that forces the particle to follow the same way of

\[
\text{fitness}(x_k) = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

\[
v_k^t = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

in which \( w \) is the inertia factor that forces the particle to move in the same direction of the previous iteration, \( c_1 \) is the cognitive factor that indicates the self-confidence of the particle, \( c_2 \) is the social factor that forces the particle to follow the same way of

\[
\text{fitness}(x_k) = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

\[
v_k^t = w \cdot v_k^{t-1} + c_1 r_1 (p_{best_k} - x_k^{t-1}) + c_2 r_2 (g_{best} - x_k^{t-1})
\]

in which \( w \) is the inertia factor that forces the particle to move in the same direction of the previous iteration, \( c_1 \) is the cognitive factor that indicates the self-confidence of the particle, \( c_2 \) is the social factor that forces the particle to follow the same way of
the best particle of the cloud, \( r1 \) and \( r2 \) are random numbers between \([0, 1]\), \( p_{\text{best}}_k \) is the position with best fitness found by the particle \( k \), and \( g_{\text{best}} \) is the best fitness state of the whole population. To prevent the particle from driving too far away, we can adopt a velocity bound to keep it in the interval of \( V_{\text{MIN}} \) and \( V_{\text{MAX}} \), which are system parameters.

The particle position is then updated according to the equation 5.

\[
x_k^t = x_k^{t-1} + v_k^t,
\]

The process is repeated until the algorithm reaches one of its stop conditions, which can be achieving an acceptable value for the optimal solution, or the accomplishment of a predefined maximum number of iterations.

4. PSO APPLIED TO COMBINATORIAL OPTIMIZATION PROBLEMS

The PSO classical approach needs some adjustments in order to be applied to combinatorial optimization problems, such as redefining the particle in a discrete model, and adapting velocity operators [12]. The original equations of PSO previously presented are kept in some of the suggested algorithms, but this is not a unanimity.

Kennedy and Eberhart [3] encoded a particle \( k \) as a binary matrix \( X_k = (x_{k,11}, x_{k,12}, ..., x_{k,nn}) \), with velocity and trajectory of the particles being defined as a probability matrix \( V_k = (v_{k,11}, v_{k,12}, ..., v_{k,nn}) \), \( v_{k,ij} \in \mathbb{R} \), in which the binary values can change from 0 to 1. In the binary space, moving particle can be seen as the numbers of bits changed at each iteration. The particle movement was defined based on the probability of a position choosing one of two possible status, considering that the velocity is restricted to the interval \([0, 1]\) by the application of the sigmoid function \( S(v_k,ij) \). According to the authors’ example, if \( v_{k,ij} = 0.20 \), then there is a 20% chance that the bit \( x_{k,ij} \) will become 1, and a 80% chance that it will become 0. The original PSO equation 4 for continuous problems remains the same. We use this version of the PSO algorithm in our first experiment with the CCAP/TAP problem.

Hu, Eberhart and Shi [5] developed a discrete PSO that codifies the particles as numeric sequences of positions, and defines the movement of the particles as swap operations in these sequences. Again, the classic PSO equation 4 was preserved in this algorithm. However, the velocity is normalized to the interval \([0, 1]\), considering that it represents the probability that changes in the numeric sequence of positions might happen. If such change happens, the current position will be switched with the position that stores the same \( g_{\text{best}} \) value. The update of the particle velocity equation (equation 4) makes the movement happen independently in the three components – inertia, cognitive and social factors – and therefore it’s possible that one or more positions will show the same value after the update, resulting in invalid solutions for the problem. The use of swap operations, as it has been done in this paper, eliminates the conflicts. Furthermore, a perturbation operator that causes a random positions will show the same value after the update, resulting in invalid solutions for the problem. The use of swap operations, as redefining the particle in a discrete model, and adapting velocity operators [12]. The original equations of PSO previously presented are kept in some of the suggested algorithms, but this is not a unanimity.

5. THE PROPOSED APPROACHES

In the next subsections we present the two versions of the PSO algorithm employed in our experiments. They are based on the previously described models and are used to solve the CCAP/TAP optimization problem.

5.1 BINARY CODIFICATION (PSO-B)

The first version of discrete PSO is based on the algorithm suggested by Kennedy and Eberhart [3]. It uses a binary codification to represent each particle in the discrete searching space, and it will be called PSO with binary codification (PSO-B). Considering that the searching space of the problem is discrete, a particle is defined as a bi-dimensional matrix of binary values, as shown in Table 1. One of the dimensions of the matrix represents the service offer agents (cabs) and the other dimension represents those demand service agents (customers); we remember that as \( ||V|| = ||P|| \), this is a square matrix.

<table>
<thead>
<tr>
<th>Offer Agent (i)</th>
<th>Demand Agent (j)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table: Representation of the Binary Particle (4x4)
The matrix $X^t_k$ represents a particle $k$ made of $n^2$ bits, which is considered a potential solution to the problem. When $x^t_{k,ij} = 1$, the $i$-th cab will be allocated to the $j$-th customer, $x^t_{k,ij} = 0$ otherwise.

$$X^t_k = \begin{bmatrix}
    x^t_{k,11} & x^t_{k,12} & \cdots & x^t_{k,1n} \\
    x^t_{k,21} & x^t_{k,22} & \cdots & x^t_{k,2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    x^t_{k,n1} & x^t_{k,n2} & \cdots & x^t_{k,nn}
\end{bmatrix} \quad (6)$$

The fitness $f$ of a particle $x^t_k$ is calculated by summing up the geographical distances between those service offer agents and those demand agents, as defined by equation 7.

$$f(x^t_k) = \sum_{i,j}^{n} \text{distance}(i,j) \cdot x^t_{k,ij} \quad (7)$$

The particle’s velocity, calculated by equation 4, is represented as the probabilities (Table 2) of the position of the matrix assuming the value 1 in the next iterations. The higher the value of the velocity, the higher is the probability that the binary variable will be 1.

<table>
<thead>
<tr>
<th>Demand Agent ($j$)</th>
<th>Offer Agent (i) 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>0.20</td>
<td>0.50</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.13</td>
<td>0.67</td>
<td>0.41</td>
</tr>
<tr>
<td>3</td>
<td>0.72</td>
<td>0.99</td>
<td>0.12</td>
<td>0.55</td>
</tr>
<tr>
<td>4</td>
<td>0.13</td>
<td>0.02</td>
<td>0.63</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Equation 8 is used to normalize the velocity, keeping it limited to the interval $[0, 1]$.

$$N(v^t_k) = \frac{1}{1 + \exp(-v^t_k)} \quad (8)$$

Then, the particle’s position is updated by the Equation 9, adding the normalized velocity to the previous particle’s position.

$$x^t_{k,i,j} = x^{t-1}_{k,i,j} + N(v^t_k) \quad (9)$$

In this paper, each particle constructs its new allocation matrix based on its probability of changing position. At each iteration of the algorithm, a particle $x^t_k$ starts with a matrix filled with 0 and places the value 1 by applying this changes at positions chosen randomly; the higher the value of the velocity related to position $(i, j)$, the higher is the probability that the change will happen in $x^t_{k,i,j}$. The binary matrix must present only one value per line and per column due to a problem restriction to be imposed to the binary encoding, so extra caution is required: only one random draw of the lines and columns for the application of the probabilities of the values exchange is necessary. This is required to avoid the creation of unfeasible solutions for the problem, therefore, a method inspired in a tabu search is used to store the “tabu states”, i.e. the method stores the lines and columns that were already “visited” in previous draws (states that have their value changed from 0 to 1). The construction of the particle is completed when all lines and columns of the matrix have one (and only one) position with value 1. The process performs until a certain number of iterations is reached.

### 5.2 CODIFICATION WITH POSITION SEQUENCES (PSO-P)

The second version of the discrete PSO uses the representation of the particles as position permutations (PSO-P), and is based on Hu, Eberhart and Shi’s proposition [5]. In this implementation, the particle’s position is represented by a vector $x^t_k = (x_1, x_2, \ldots, x_n)$ of integer numbers whose indexes identify the service offer agents and whose values represent the demand agents. A particle whose position $i$ in the vector $x$ stores the value of $j$, indicates that the $i$-th cab will be allocated to the $j$-th customer. In the example of Table 3, cab 1 would be allocated to customer 4, cab 2 would be allocated to customer 3, and so on.

<table>
<thead>
<tr>
<th>Demand Agent ($j = x_i$)</th>
<th>Offer Agent (i) 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

The particle’s velocity is also calculated by equation 4. The velocity vector is then normalized, by dividing all its position values by the highest value of this vector; so, the velocity vector coordinates belong to the interval $[0, 1]$. The particle movement occurs due to permutations of the values of vector $x^t_k$, considering that the velocity indicates the probability of the swap operation happening in each position of $x^t_k$. If a swap is supposed to happen in a certain position $i$ of $x^t_k$, this position will be exchanged with the value of the position that stores the corresponding gbest value. This process is shown in Figure 1. As in the previous method, the process is finished when a certain number of iterations is reached.
Like in PSO-B, the particle $x^t_k$ fitness $f$ is calculated by summing up the geographical distance between the offer and demand agents (equation 10).

$$f(x^t_k) = \sum_i distance(i, X^t_{k,i})$$  (10)

**5.3 EXHAUSTIVE SEARCH**

The process of exhaustive search (ES) was also implemented in order to identify the optimal solution of some instances of the problem and to permit a qualitative evaluation of the solutions obtained by both PSO implementations. Also, we can evaluate computational feasibility, by calculating effective processing time. For this purpose, we implement an exhaustive search using breadth first and a backtracking algorithm. All possible solutions to the problem are obtained and evaluated, by using the same function “distance” that was applied in the PSO algorithms to measure the fitness of solutions. The ES execution was carried out for $10 \leq N \leq 13$, because running time is excessively high for bigger instances of the problem.

**6. OBTAINED RESULTS**

The two PSO implementations were tested in a simulation environment that encompasses a series of cabs and customers allocated at random in the Curitiba city map. The environment employs direct Euclidean geographic distances, disregarding streets and other urban characteristics.

Two valid solutions to the problem are presented on Listing 1 and 2, according to the log extracted from the simulated environment.

**Listing 1: Log of the Simulated Environment - Binary Particle**

```
23:25:59 [PSO-B] Particle Swarm Optimization - Binary Codification
23:25:59 [PSO-B] Run up to 100 iterations.
23:26:02 [PSO-B] Lower Cost (gbest): 39.6092
Best allocation found:
|0|0|0|0|0|0|0|0|1|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|0|1|0|
|0|0|1|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|1|0|0|0|0|0|0|0|0|
|1|0|0|0|0|1|0|0|0|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|1|0|0|
|0|0|0|1|0|0|0|0|0|0|0|0|0|
|0|1|1|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|1|0|0|0|0|0|0|0|0|
|0|0|0|0|1|0|0|0|0|0|0|0|0|
```
Table 4 shows the results obtained in the experiments conducted with PSO-B, PSO-P and the optimal solution of the CCAP/TAP problem obtained by the exhaustive search. The PSO-B and PSO-P algorithms were run 30 times, as proposed in [19], in the same problem scenario – with the same agents who offered and demanded services and the same geographical locations – for each N value, ranging from 10 to 100. The table shows the running time T(s) in which the best solution among the 30 runs was found, fitness (Fitness) of the best solution, the average of fitness and computational time among 30 runs. Moreover, the standard deviation (Std) indicating the differences in the results during the runs. The ES computational time was obtained from one single run, since this execution is deterministic. This comparison is also illustrated as a chart (Figure 2), where the fitness is normalized by dividing it by the problem size.

<table>
<thead>
<tr>
<th>N</th>
<th>PSO-B (Best) T(s)</th>
<th>PSO-B (Avg) T(s)</th>
<th>PSO-P (Best) T(s)</th>
<th>PSO-P (Avg) T(s)</th>
<th>ES T(s)</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.298 1.342 2.4921 1.0624</td>
<td>0.101 22.1717 0.074 22.5494 0.3535</td>
<td>2.800 22.1717</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1.559 1.521 26.8888 0.9301</td>
<td>0.071 23.9063 0.078 24.5624 0.7451</td>
<td>37.415 23.9063</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1.778 1.777 31.3413 1.3043</td>
<td>0.097 26.6074 0.085 27.2730 0.8892</td>
<td>549.213 26.6074</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2.007 2.000 41.5564 1.0807</td>
<td>0.110 38.0663 0.086 38.5768 0.8877</td>
<td>7879.010 38.0663</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2.236 37.8782 2.293 40.4563 1.5718</td>
<td>0.088 32.6583 0.098 34.3854 1.5879</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2.599 31.5324 2.570 36.3549 2.0210</td>
<td>0.104 24.1569 0.102 26.7855 1.7945</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2.887 40.3216 2.883 46.4788 2.1744</td>
<td>0.141 28.4227 0.117 31.4730 1.7984</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>3.315 43.4542 3.233 50.6632 2.6707</td>
<td>0.166 34.0799 0.126 37.2665 2.0879</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3.736 47.8339 3.694 51.5818 1.7893</td>
<td>0.132 32.4218 0.134 36.0528 2.6917</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>4.016 51.4295 4.004 55.9041 2.0595</td>
<td>0.121 31.7753 0.128 37.8368 3.1598</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>4.392 55.0074 4.368 62.4103 2.8493</td>
<td>0.133 37.3467 0.136 42.8159 1.9549</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>6.701 68.5871 6.686 78.5949 3.4476</td>
<td>0.160 38.7568 0.173 47.0397 4.8071</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>9.614 87.2124 9.797 95.5777 4.0157</td>
<td>0.199 47.0326 0.202 57.8749 4.5788</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>12.225 113.6777 12.895 121.7078 3.5224</td>
<td>0.220 59.9536 0.232 71.2950 5.2466</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>16.451 141.5243 16.624 149.0679 3.8478</td>
<td>0.276 77.4291 0.270 89.5462 7.2157</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>21.651 163.3039 20.745 174.5440 3.8950</td>
<td>0.289 81.7629 0.293 99.4174 9.0979</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>24.937 181.7488 24.859 190.2445 4.0551</td>
<td>0.338 101.7060 0.338 119.3804 8.0808</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>30.493 198.4549 30.717 209.0799 4.0688</td>
<td>0.361 105.7905 0.404 119.9533 6.2966</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>36.214 225.1250 35.815 238.4322 6.2219</td>
<td>0.378 125.5063 0.382 150.5347 9.9181</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>46.653 241.4454 42.618 253.4043 4.7395</td>
<td>0.402 130.7460 0.416 152.2496 10.1805</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>47.943 266.8712 47.979 281.9845 6.8302</td>
<td>0.445 141.4213 0.447 167.4429 14.8515</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>55.817 283.5852 55.414 293.3672 5.4883</td>
<td>0.501 155.8125 0.485 177.7850 10.3175</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>63.501 309.0156 62.787 322.6606 5.6532</td>
<td>0.682 170.3120 0.538 192.0580 13.3802</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>69.047 332.9203 70.871 350.4157 6.0664</td>
<td>0.531 190.3909 0.541 209.3188 10.1399</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>78.323 345.3876 78.210 357.2171 6.2374</td>
<td>0.592 198.6034 0.596 220.6628 10.9162</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>89.666 366.8325 90.855 383.2419 7.1417</td>
<td>0.595 214.1308 0.636 234.4459 15.6866</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>106.421 408.2418 99.395 427.6848 9.6944</td>
<td>0.661 235.5792 0.664 265.5063 14.7411</td>
<td>- -</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Obviously, the exhausting search finds the optimal solution to the problem, because it evaluates all feasible problem solutions. However, its applicability becomes impracticable when the size of the problem increases, due to processing time; so, the ES experiments could only be performed for N ≤ 13.

PSO-B presents acceptable solutions for the problem when N ≤ 20. However, in bigger instances of the problem, the best solution of PSO-B is far from the solution provided by PSO-P. Furthermore, the execution time of the PSO-B algorithm increases considerably according to the value of N. On the other hand, the PSO-P algorithm finds the optimal solution for the problem in all cases in which the optimal value was available. The average fitness of the 30 executions is really close to the optimal value, and to all values of N tried in this paper, the time of the execution was below 1 second. The program was coded in Java and run on Intel Core 2 Duo 2.40GHz PC.
7. CONCLUSIONS AND FUTURE WORK

This paper presented the application of the discrete PSO algorithm for the optimization of a combinatorial problem formalized as a Task Assignment Problem. Two versions of the discrete PSO were considered: the first one uses binary codification and the second one employs position permutations. The results obtained with these two algorithms were compared with the optimal solution obtained by an exhausting search method, in order to evaluate their quality and performance; the optimal solutions were found for problem sizes \( N \) ranging from 10 to 13.

For \( N > 13 \), it was impossible to find out optimal values, due to computational cost (e.g., for \( N = 13 \), the computational time was longer than two hours and increases exponentially depending on the problem size). In spite of this, the solutions found by the ES are important to measure the quality of the solutions obtained by the other methods.

PSO-P algorithm found the optimal solutions in all the instances of the problem when they were available. On the other hand, the PSO-B only got close to optimal solutions in instances with lower sizes. To illustrate, PSO-B algorithm reached an average fitness 10.96\% worse than the average fitness achieved by PSO-P algorithm when \( N = 10 \), rising to a value of 61.08\% worse in the case of \( N = 100 \).

Therefore, the results obtained with PSO-P are promising: we argue that this algorithm can find the optimal solutions in less time. So, this algorithm is a serious candidate to be used in real-time on-line applications.

We intend to expand our algorithms and experiments in two directions:

- By considering the case \(|V| \neq |P|\) (different number of cabs and customers). This extension can be treated internally to the model—using non-square matrices—or by adding a previous filter based on queue priority; and
- By updating the optimization scenario to consider the dynamic nature of the real problem. Up to now in our modeling the optimization occurs in a fixed time, where some free cabs must be allocated to some demanding customers. In a real scenario the cab agents positions change with time. There are three situations to consider: (a) the cab positions change because they are moving to the allocated costumer; (b) when the cab arrives to the costumer position a pair (cab, costumer) is created, and these elements must be eliminated from the optimization scenario (the cab is “occupied”); (c) when the pair (cab, costumer) arrives to its destination the service ends, and a new service offer agent appears “suddenly” (the cab becomes “free”). These dynamic characteristics of the problem must be considered in future simulation experiments.

We also remark that our proposal must be applied to a real allocation system, where the agent positions (cab and costumer) are signaled to a central control station in real-time by mobile devices, using GPS coordinates.

REFERENCES


