X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro de 2011, Fortaleza, Ceará

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 method that was 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 presents and solves an application of the combinatorial problem of allocation – consisting of cabs and customers – whose optimization goal is to minimize the distance traveled by the fleet. This problem can be categorized as a Task Assignment Problem, and it is optimized in this paper with two implementations of the discrete PSO: the first aprroach 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, besides it is capable of achieving all the optimal values calculated by an exhaustive search.

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 takes part in an area of studies called Swarm Intelligence, 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. In most of these methods, PSO showed the best results 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 workings 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 aplication of the TAP – namely 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 with the optimal value, found in the exhausting search, for problem sizes in which this is computationally possible.

2. PROBLEM FORMULATION

The organization and the 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 gases.

The Cab-Customer Allocation Problem (CCAP) consists of the allocation of 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 $||V|| = ||P|| = N$.

$$A : V \rightarrow P \tag{1}$$

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:

$$C(A) = \sum_{i=1}^{n} distance(i, A(i)) \tag{2}$$

where $distance(i, j)$ 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 not considering the streets and other urban traffic characteristics. The problem is to find the optimal solution A^o with minimal cost, i.e.:

$$A^o = argmin\ C(A) \qquad \forall A \in \Omega \tag{3}$$

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 Particle Swarm Optimization algorithm, proposed by James Kennedy and Russel Eberhart in 1995 [1], was inspired on the social behavior of flocks of birds. 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.

In 1983, Reeves [17] already used a particles system in his graphic animation projects at Lucasfilm. He created a stochastic system that moved many dots to form diffuse objects, such as explosions and clouds. After that, Reynolds [18] modified the component that made the particles to be move by adding orientation and communication to the objects of the system. In the paper that originated PSO [1], Kennedy and Eberhart extended Reynold's model to incorporate some social behavior to the system.

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) [19]. 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 [20]. One of the reasons why PSO is an attractive solution for optimization problems is that little effort is demanded for parameterization, 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 some velocity, making the particle move around the search space looking for a better solution. The velocity of each particle depends on its previous velocity, its best attained solution and the best solution attained by the cloud, as presented in the following. We remark that position and velocity are considered as vectorial quantities in the search space S.

Algoritmo 1 Pseudo-code of the PSO algorithm for minimization

1: $t = 0$
2: $\mathbf{gbest} \leftarrow$ a big initial value
3: **for** $k = 0$ to $swarmSize - 1$ **do**
4: $\quad \mathbf{x}_k^0 \leftarrow$ a random solution
5: $\quad \mathbf{v}_k^0 \leftarrow$ a random velocity $\in [V_{MIN}, V_{MAX}]$
6: $\quad \mathbf{pbest}_k \leftarrow \mathbf{x}_k^0$
7: **end for**
8: $\mathbf{gbest} \leftarrow$ the better solution $\in [\mathbf{x}_0, \mathbf{x}_{swarmSize-1}]$
9: **while** not reach a stop condition **do**
10: $\quad$ **for** $k = 0$ to $swarmSize - 1$ **do**
11: $\quad\quad$ **if** fitness($\mathbf{x}_k^t$) < fitness($\mathbf{pbest}_k$) **then**
12: $\quad\quad\quad \mathbf{pbest}_k \leftarrow \mathbf{x}_k^t$
13: $\quad\quad$ **end if**
14: $\quad\quad$ **if** fitness($\mathbf{x}_k^t$) < fitness($\mathbf{gbest}$) **then**
15: $\quad\quad\quad \mathbf{gbest} \leftarrow \mathbf{x}_k^t$
16: $\quad\quad$ **end if**
17: $\quad\quad \mathbf{v}_k^t = w\mathbf{v}_k^{t-1} + c_1 r_1(\mathbf{pbest}_k - \mathbf{x}_k^{t-1}) + c_2 r_2(\mathbf{gbest} - \mathbf{x}_k^{t-1})$
18: $\quad\quad \mathbf{v}_k^t \in [V_{MIN}, V_{MAX}]$
19: $\quad\quad \mathbf{x}_k^t \leftarrow \mathbf{x}_k^{t-1} + \mathbf{v}_k^t$
20: $\quad$ **end for**
21: $\quad t = t + 1$
22: **end while**

A fitness evaluation function measures the quality of each position occupied by the particle over the search space. This function indicates how good is the current position of the particle in this iteration. Each particle k memorizes its best position ($\mathbf{pbest}_k$) and the best position reached among all particles of the cloud ($\mathbf{gbest}$). These two components in addition to an inertia factor are considered in the update of velocity (and direction) of the particle, as defined by equation 4:

$$\mathbf{v}_k^t = w.\mathbf{v}_k^{t-1} + c_1.r_1(\mathbf{pbest}_k - \mathbf{x}_k^{t-1}) + c_2.r_2(\mathbf{gbest} - \mathbf{x}_k^{t-1}), \tag{4}$$

in which w is the inertia factor that forces the particle to move in the same direction of the previous iteration, $c1$ is the cognitive factor that indicates the self-confidence of the particle, $c2$ 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 $[0, 1]$, $\mathbf{pbest}_k$ is the position with best fitness found by the particle k, and $\mathbf{gbest}$ is the best fitness state of the whole population. To prevent the particle from driving too far away, it's possible to apply a velocity bound to keep it in the interval of V_{MIN} and V_{MAX}, that are system parameters.

The position of the particle is then updated according to the equation 5, which considers that the movement of the particle consists simply in adding the velocity to its current position.

$$\mathbf{x}_k^t = \mathbf{x}_k^{t-1} + \mathbf{v}_k^t, \tag{5}$$

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

Originally, PSO was conceived to be applied in continuous problems with search space given by the real numbers. However, many practical problems can be represented as combinatorial optimization problems, and their decision variables must be codified in a discrete way. The first version of PSO for discrete problems was developed in Kennedy and Eberhart's paper [3]. Since then, other papers which suggested algorithms based on PSO for the optimization of combinatorial problems have been published. The classical approach of the technique needs to suffer some adjustments in order to be applied. The main adjustments are: the redefinition of the particle in a discrete model, and the redefinition of the velocity operators [12]. The original equations of PSO previously presented are kept in some of the suggested algorithms, but that's not an unanimity.

Kennedy and Eberhart [3] codified a particle k as a binary matrix $\mathbf{X}_k = (x_{k,11}, x_{k,12}, ..., x_{k,nn}), x_{k,ij} \in \{0, 1\}$, and the velocity and trajectory of the particles were defined as a probability matrix $\mathbf{V}_k = (v_{k,11}, v_{k,12}, ..., v_{k,nn}), v_{k,ij} \in \mathbb{R}$, 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 movement of the particle was defined based on the probability of a position choosing one status or another, 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 of positions. Again, the classic PSO equation 4 was preserved in this algorithm. However, the velocity is normalized to the interval $[0, 1]$ after the calculation, considering that it represents the probability that alterations in the numeric sequence of positions might happen. If it's randomly determined that the positions swap must happen, the current position will be switched with the position that stores the same $\mathbf{gbest}$ 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 swap everytime the particle stagnates in the same $\mathbf{gbest}$ position is included. The authors show that the suggested algorithm is competitive according to the comparisons with GA in n-queens problems [5]. So, we choose this proposal for the second experiment with the CCAP/TAP problem.

5. 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 on Table 1. One of the dimensions of the matrix represents the service offer agents (cabs) and the other dimension represents those demand service agents (costumers); we remember that as $||V|| = ||P||$, this is a square matrix.

Table 1: Representation of the Binary Particle (4x4)

	Demand Agent (j)			
Offer Agent (i)	1	2	3	4
1	0	0	1	0
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1

The matrix X_k^t represents a particle k made of n^2 bits, which is considered a potential solution to the problem. When $x_{k,ij}^t = 1$, the i-th cab will be allocated to the j-th customer, $x_{k,ij}^t = 0$ otherwise.

$$X_k^t = \begin{bmatrix} x_{k,11}^t & x_{k,12}^t & \dots & x_{k,1n}^t \\ x_{k,21}^t & x_{k,22}^t & \dots & x_{k,2n}^t \\ \dots & & & \\ x_{k,n1}^t & x_{k,n2}^t & \dots & x_{k,nn}^t \end{bmatrix} \tag{6}$$

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

$$f(\mathbf{x}_k^t) = \sum_{i,j} \; distance(i,j) \; . \; x_{k,ij}^t \tag{7}$$

The velocity of the particle, calculated by equation 4 of the classic PSO for continuous problems, 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 bigger the probability that the binary variable will be 1.

Table 2: Binary Particle Velocity (4x4)

	Demand Agent (j)			
Offer Agent (i)	1	2	3	4
1	0.10	0.20	0.50	0.73
2	0.99	0.13	0.67	0.41
3	0.72	0.99	0.12	0.55
4	0.13	0.02	0.83	0.17

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

$$S(\mathbf{v}_k^t) = \frac{1}{1 + \exp(-\mathbf{v}_k^t)} \tag{8}$$

In this paper, each particle constructs its new allocation matrix based on its changes of probabilities. At each iteration of the algorithm, a particle $\mathbf{x}_k^t$ 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 bigger the probability that the change will happen in $x_{k,ij}^t$. The binary matrix must present only one value 1 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 POSITIONS SEQUENCES (PSO-P)

The second version of the discrete PSO uses the representation of the particles as positions permutations (PSO-P), and is based on Hu, Eberhart and Shi's proposition [5]. In this implementation the position of the particle consists of a vector $\mathbf{x_k^t} = (x_1, x_2, \dots x_n)$ of integer numbers whose indexes represent the service offer agents and whose values represent the demand agents. A particle whose position i in the vector $\mathbf{x}$ stores the value of j, indicates that the i-th cab will be allocated by the j-th customer. On the example of Table 3, cab 1 would be allocated by customer 4, cab 2 would be allocated by customer 3, and so on.

Table 3: Representation of the Particle as a Positions Sequence

Offer Agent (i)	1	2	3	4	5	...
Demand Agent ($j = x_i$)	4	3	7	9	2	...

The velocity of the particles is also calculated by equation 4, which came from the classic PSO proposal. 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 movement of the particles occurs due to the change of the positions of the values of vector $\mathbf{x_k^t}$, considering that the velocity indicates the probability of the swap operation happening in each position of $\mathbf{x_k^t}$. If it is defined that the swap operation must happen in a certain position i of $\mathbf{x_k^t}$, this position will be exchanged with the value of the position that stores the corresponding $\mathbf{gbest}$ 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.

						▼	
v	...	0.15	0.22	0.63	0.94	0.51	...
						▼	
gbest	...	7	2	3	1	8	...
		▼				▼	
x	...	8	1	3	4	2	...
		└				↑	
x + v	...	2	1	3	4	8	...

Figure 1: Movement of the Particle Codified as a Positions Sequence

Like in PSO-B, the particle $\mathbf{x}_k^t$ fitness f is calculated by summing up the geographical distance between the offer and demand agents (equation 9).

$$f(x_k^t) = \sum_i distance(i, X_{k,i}^t) \tag{9}$$

5.3 EXHAUSTIVE SEARCH

The process of exhaustive search (ES) was also implemented in order to identify the optimal solution of some applications of the problem and to permit a qualitative evaluation of the solutions obtained by the 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 until the problem size $N = 13$, because the execution periods are impracticable 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 costumers allocated at random in the Curitiba city map. The environment employs direct Euclidean geographic distances, not considering streets and other urban characteristics.

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] Swarm initialized with 20 particles.
23:25:59 [PSO-B] Run up to 100 iterations.
23:26:02 [PSO-B] Lower Cost (gbest): 35,9479
           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|0|1|0|0|0|0|0|0|0|
           |1|0|0|0|0|0|0|0|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|0|1|
           |0|0|0|0|0|0|0|1|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|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|0|0|0|
           |0|0|0|0|0|0|0|0|0|1|0|0|0|
23:26:03 [PSO-B] Time: 1889 milliseconds.
```

Listing 2: Log of the Simulated Environment - Particle with Positions Sequence

```
02:12:33 [PSO-P] Particle Swarm Optimization - Sequences of Positions
02:12:33 [PSO-P] Swarm initialized with 20 particles.
02:12:33 [PSO-P] Run up to 100 iterations.
02:12:33 [PSO-P] Lower Cost (gbest): 32,2159
           Best allocation found:
           |12|2|11|4|3|1|10|7|9|5|8|2|6|
02:12:33 [PSO-P] Time: 121 milliseconds.
```

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 exhaustive search. The PSO-B and PSO-P algorithms were performed 10 times in the same problem scenario – with the same agents who offered and demanded and the same geographical locations – for each N value, ranging from 1 to 100. The table shows the time period of the execution $T(s)$ in which the best solution among the 10 runs was found, the fitness (Fitness) of the best solution, and the average of fitness and computational time among 10 runs. The computational time of the ES was obtained from one single run, since this execution is fixed, not involving random variations. This comparison is also illustrated as a chart (Figure 2).

Table 4: Results of the Optimization of the CCAP/TAP problem

N	PSO-B (Best)		PSO-B (Avg)		PSO-P (Best)		PSO-P (Avg)		ES	
	T(s)	Fitness	T(s)	Fitness	T(s)	Fitness	T(s)	Fitness	T(hh:mm:ss.zzz)	Fitness
10	1.229	23.7141	1.209	24.8715	**0.089**	**23.3600**	0.083	23.4662	00:00:03.129	**23.3600**
11	1.418	22.2994	1.430	23.7310	**0.154**	**19.4472**	0.097	19.6784	00:00:37.432	**19.4472**
12	1.725	36.9372	1.742	38.4492	**0.093**	**35.8272**	0.101	35.9286	00:09:10.764	**35.8272**
13	1.889	35.9479	1.918	38.7318	**0.121**	**32.2159**	0.110	32.5263	02:11:20.092	**32.2159**
14	2.329	39.6775	2.257	42.2710	**0.118**	**33.8634**	0.117	34.4242	-	-
15	2.580	32.8460	2.576	37.9584	**0.123**	**27.1603**	0.128	28.2201	-	-
16	2.807	44.0949	2.864	48.2039	**0.128**	**31.3534**	0.133	33.3330	-	-
17	3.125	43.4480	3.180	47.7794	**0.141**	**30.7462**	0.144	33.2285	-	-
18	3.634	46.8600	3.615	49.3912	**0.133**	**29.3217**	0.147	30.6117	-	-
19	3.858	60.0431	3.991	62.6760	**0.165**	**45.2218**	0.162	47.3199	-	-
20	4.213	55.5933	4.354	63.1136	**0.163**	**41.1878**	0.195	43.6537	-	-
25	6.472	75.8533	6.562	80.6036	**0.230**	**41.9651**	0.219	46.6270	-	-
30	9.172	99.5714	9.294	105.0270	**0.410**	**57.5728**	0.263	62.5235	-	-
35	12.091	113.5682	12.183	122.5816	**0.313**	**58.4400**	0.296	64.8434	-	-
40	15.663	143.2669	15.850	148.0703	**0.343**	**77.0390**	0.317	86.8538	-	-
45	19.689	163.2162	20.278	166.6210	**0.359**	**84.6674**	0.396	91.8101	-	-
50	24.009	183.6746	24.471	190.6380	**0.407**	**81.3636**	0.410	94.0227	-	-
55	28.930	219.5384	29.334	223.5993	**0.405**	**116.7213**	0.431	125.3830	-	-
60	34.992	203.5088	34.505	226.5455	**0.516**	**109.4617**	0.485	121.3424	-	-
65	39.001	239.3880	39.275	248.0134	**0.532**	**124.1192**	0.505	136.4366	-	-
70	49.174	259.8147	47.085	271.1215	**0.515**	**134.5329**	0.540	147.7733	-	-
75	53.305	286.3620	54.226	296.9904	**0.593**	**138.7339**	0.599	156.7370	-	-
80	57.191	303.3248	58.085	311.2290	**0.624**	**169.3441**	0.619	178.4513	-	-
85	65.147	340.3844	65.028	350.2784	**0.608**	**186.9750**	0.624	198.0973	-	-
90	72.604	364.4554	72.695	368.5378	**0.687**	**185.7383**	0.691	199.6386	-	-
95	80.138	375.0624	80.567	385.9539	**0.703**	**217.3337**	0.693	234.5874	-	-
100	99.572	404.6762	96.066	415.8354	**0.889**	**225.0785**	0.789	238.1581	-	-

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 $N \leq 13$.

PSO-B presents relatively good solutions for the problem when $N \leq 20$. 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. The PSO-P algorithm finds the optimal solution for the problem in all cases in which the optimal value was found by the exhaustive search ES. The average fitness of the 10 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.

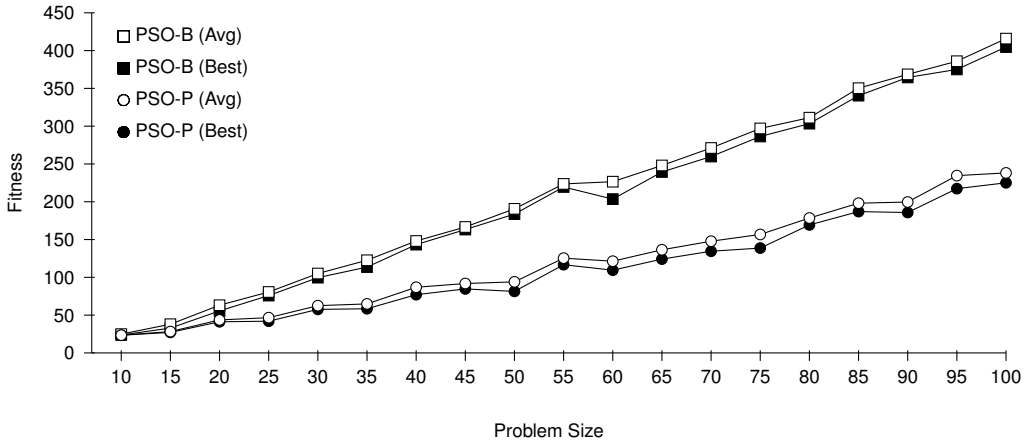

Figure 2: Evolution of the Fitness According to the Problem Size

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 1 to 13. For higher N values it wasn't possible to find the optimal value, since for $N = 13$, the computational time was longer than two hours and increases exponentially depending on the size of the problem. In spite of this, the solutions found by the ES are important to measure the quality of the solutions obtained by the other methods.

The PSO-P algorithm was capable of optimizing the CCAP/TAP problem in all the instances of the problem whose optimal value was found by ES. On the other hand, the PSO-B only got close to the optimal solutions in the instances of the problem with lower sizes. To illustrate, the PSO-B algorithm reached an average fitness $5,98\%$ worse then the PSO-P algorithm when $N = 10$, rising to a value of $74,60\%$ worse in the case of $N = 100$.

Therefore, the results obtained with PSO-P are promising: we argue that its best solutions can be compared with the ones obtained by ES, and the algorithm execution times are small and can be bounded, since it depends mainly on the size of the swarm and on the predefined number of iterations. So, this algorithm is a serious candidate to be used in real-time on-line applications.

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

- By considering the case $||V|| \neq ||P||$ (different number of cabs and costumers). 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 moment, where some free cabs must be allocated to some demanding costumers. 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

[1] J. Kennedy and R. C. Eberhart. "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, 1995*, pp. 1942–1948, 1995.

[2] R. Poli. "Analysis of the Publications on the Applications of Particle Swarm Optimisation". *Journal of Artificial Evolution and Applications*, vol. 2008, 2008.

[3] J. Kennedy and R. C. Eberhart. *A discrete binary version of the particle swarm algorithm*, volume 5, pp. 4–8. IEEE Press, Piscataway, NJ., 1997.

[4] A. Salman, I. Ahmad and S. Al-Madani. "Particle swarm optimization for task assignment problem". *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363 – 371, 2002.

[5] X. Hu, R. Eberhart and Y. Shi. "Swarm Intelligence for Permutation Optimization: A Case Study on N-Queens Problem". *In Proceedings of the IEEE Swarm Intelligence Symposium 2003*, pp. 243–246, 2003.

[6] D. Y. Sha and C.-Y. Hsu. "A hybrid particle swarm optimization for job shop scheduling problem". *Computers and Industrial Engineering*, vol. 51, pp. 791–808, December 2006.

[7] C. Liao, C. Tseng and P. Luarn. "A Discrete Version of Particle Swarm Optimization for Flowshop Scheduling Problems". *Comput. Oper. Res.*, vol. 34, no. 10, pp. 287–308, 2007.

[8] D. Y. Sha and C.-Y. Hsu. "A new particle swarm optimization for the open shop scheduling problem". *Computers and Operations Research*, vol. 35, pp. 3243–3261, October 2008.

[9] I.-H. Kuo, S.-J. Horng, T.-W. Kao, T.-L. Lin, C.-L. Lee, T. Terano and Y. Pan. "An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model". *Expert Syst. Appl.*, vol. 36, pp. 7027–7032, April 2009.

[10] M. Rosendo and A. Pozo. "A Hybrid Particle Swarm Optimization Algorithm for Combinatorial Optimization Problems". *In Proceedings of Congress on Evolutionary Computation - CEC 2010, Barcelona*, 2010.

[11] D. Y. Sha and H.-H. Lin. "A multi-objective PSO for job-shop scheduling problems". *Expert Syst. Appl.*, vol. 37, pp. 1065–1070, March 2010.

[12] H. Liu, L. Gao and Q. Pan. "A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem". *Expert Syst. Appl.*, vol. 38, pp. 4348–4360, April 2011.

[13] B. Jarboui, S. Ibrahim, P. Siarry and A. Rebai. "A combinatorial particle swarm optimisation for solving permutation flowshop problems". *Computers and Industrial Engineering*, vol. 54, pp. 526–538, April 2008.

[14] L. Fang, P. Chen and S. Liu. "Particle swarm optimization with simulated annealing for TSP". In *Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases - Volume 6*, pp. 206–210, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[15] D. Tank and J. Hopfield. "Collective Computation in Neuronlike Circuits". *Scientific American*, vol. 257, no. 6, pp. 104–114, 1987.

[16] B. L. C. Costa and F. C. C. Costa. "O sistema de táxis: mobilidade urbana e redução nas emissões de gases de efeito estufa no Rio de Janeiro". In *4º Congresso Luso-Brasileiro para o Planejamento Urbano, Regional, Integrado e Sustentável, 2010*, Universidade do Algarve, Faro, Portugal, 2010.

[17] W. T. Reeves. "Particle Systems - a Technique for Modeling a Class of Fuzzy Objects". *ACM Trans. Graph.*, vol. 2, pp. 91–108, April 1983.

[18] C. W. Reynolds. "Flocks, herds and schools: A distributed behavioral model". *SIGGRAPH Comput. Graph.*, vol. 21, pp. 25–34, August 1987.

[19] R. C. Eberhart and Y. Shi. "Comparison between Genetic Algorithms and Particle Swarm Optimization". In *Proceedings of the 7th International Conference on Evolutionary Programming VII*, EP '98, pp. 611–616, London, UK, 1998. Springer-Verlag.

[20] A. Banks, J. Vincent and C. Anyakoha. "A review of particle swarm optimization. Part I: background and development". vol. 6, pp. 467–484, December 2007.