Particle Swarm Optimization Applied to the Dynamic Allocation Problem

Jean L. Pierobom, Myriam R. Delgado, Celso A. A. Kaestner

Graduate Program on Applied Computing (PPGCA)
Federal University of Technology of Paraná (UTFPR)
Curitiba, PR, Brazil

Email: jean@pierobom.com, myriamdelg@utfpr.edu.br, celsokaestner@utfpr.edu.br

Abstract—Particle Swarm Optimization (PSO) is based on the analysis of emergent behavior of bird flocks. Though it was originally designed for continuous optimization, PSO has provided good results in some recent works when applied to static and discrete optimization problems. In this paper, the particle encoding scheme is based on permutations and the PSO algorithm is adapted to solve a real-world application (cabs-customers allocation) of the dynamic task assignment problem. In the proposed approach, as the optimal solution may change during the optimization process, different strategies to detect and react to changes are tested. The results show that combinations of traditional techniques achieve good solutions in tested instances defined with different sizes and scales of changes.

Keywords—Swarm Intelligence, Particle Swarm Optimization, Combinatorial Optimization, Task Assignment Problem, Dynamic Environments

I. INTRODUCTION

The first version of the discrete PSO algorithm was proposed in 1997 [1]. Since then, many PSO-based approaches for combinatorial optimization problems have been developed [3-10]. The Task Assignment Problem (TAP) is one of the best known problems in Combinatorial Optimization and may be used to model a large variety of practical applications including problems of scheduling in industry [3] [4], university course scheduling [8] and storage space allocation [11]. According to Salman [12], TAP is NP-hard, i.e., there is no algorithm able to find optimal solutions in polynomial time, therefore, heuristic search methods must be developed to solve it.

This paper proposes a modification of the PSO algorithm to solve a real world application of the Dynamic Task Assignment Problem (DTAP). In a previous work [13], non-dynamic instances of the TAP were solved using a PSO algorithm with particles encoded as permutations. The algorithm was called Particle Swarm Optimization with Permutation (PSO-P). In this paper, the algorithm is improved by adding some methods that enable it to deal with dynamic fitness landscapes. This new approach is called Dynamic Particle Swarm Optimization with Permutations (DPSO-P).

The rest of the paper is organized as follows. Section II presents the addressed dynamic problem and Section III discusses some related approaches. Section IV presents the proposed approach describing its encoding scheme and algorithmic details. Sections V present the experiments and results. Finally, Section VI summarizes conclusions and indicates some future works.

II. THE DYNAMIC ADDRESSED PROBLEM

The problem considered in this paper is the Cab-Customer Allocation Problem (CCAP), which can be categorized as a Dynamic TAP. The problem consists in allocating \( N \) cabs (service offer agents) to \( M \) customers (demand service agents) in a way that the total distance traveled by cabs to get customers is minimal.

In [13], CCAP was described as follows. Let \( A \) be the allocation function that maps a set \( V \) of offer service agents to a set \( P \) of demand service agents \( A : V \rightarrow P \), where \( A(i) = j \), if an offer agent \( i \) is allocated to a demand agent \( j \). Let \( C(A) \) be the cost function of a solution \( A \). Then, \( C(A) = \sum_{i=1}^{n} distance(i, A(i)) \), where \( distance(i, j) \) is the geographic distance between two points in the city; in this case, the distance between the service offer agent \( i \) and the service demand agent \( j \), for \( j = A(i) \). In [13], we use the Euclidean distance between the geographical locations.

In this paper we employ the minimal paths over the Open Street Map [14] using the Dijkstra algorithm [15]. The aim is to find the optimal solution \( A^o \) with minimal cost in the set \( \Omega \) of feasible solutions, \( A^o = \arg\min C(A) \forall A \in \Omega \).

In a real scenario as the one considered in this paper, changes occur:

- (a) a new customer comes (the customer is “waiting”);
- (b) the cab positions change because they are moving to the allocated customer;
- (c) when the cab arrives to the customer position a pair (cab, customer) is created, and these elements must be eliminated from the optimization scenario (the cab is “occupied”);
- (d) when the pair (cab, customer) arrives to its destination the service ends, and a new service offer agent appears (the cab becomes “free”).

Those transitions characterize CCAP/TAP as a dynamic optimization problem, as fitness landscape can change during the evolutionary process.

1522-4899/12 $26.00 © 2012 IEEE
DOI 10.1109/SBRN.2012.35
III. PSO IN DYNAMIC ENVIRONMENTS

In dynamic optimization problems, algorithms must be able to track new optimal solutions whenever they change due to environment modifications [16]. Depending on the dimension of the changes and swarm convergence level, particles may have difficulties to explore other regions and identify a new optimal solution. Thus, the PSO algorithm must be adapted for applications in dynamic environments. Some previous PSO-based approaches, developed specifically for those environments are summarized below.

In [17] the authors proposed a method that monitors a randomly chosen position (particle) of the search space to determine the occurrence of changes based on the particle fitness value. If a change in the fitness value of the particle is not accompanied by a change in its location, the environment has changed. Hu and Eberhart [18] proposed a similar method that was named “changed-gbest-value” and re-evaluates the gbest solution at each iteration of the algorithm. The main difference of [17] and [18] regards the particle chosen, as in the second case it is always fixed as the gbest.

According to the [19], using the old swarm may be the better approach when the changes are smooth. For major variations, the more appropriate treatment should be completely restart (in a random way) the whole swarm. Another variation of these methods can be obtained through the combination of both i.e., initialize the new population with half the particles of the original population, and the remaining particles randomly positioned on the search space.

Hu and Eberhart [20] proposed a new strategy to respond to changes. The authors explain that if changes do not “disturb” so much the optimal solution, the optimal solution is still inside the convergence area, the PSO will detect the new optimum automatically, without any modification in the algorithm. For other situations, they use a method of reallocating some particles. Among many tested strategies the best one was to diversify 10% of the swarm.

Esquivel and Coello [21] introduced a mutation operator to maintain diversity in the swarm. The mutation does not interrupt the search process as the method uses an additional memory that stores best solutions obtained and is not destroyed when the environment changes. This memory is only updated when a particle achieves a fitness value better than that stored in its memory.

In this paper, PSO is adapted and tested in a real-world application (cabs-customers allocation) of the dynamic task assignment problem. In the proposed approach, as the optimal solution may change during the optimization process, different strategies and parameters based on those PSO algorithms are tested to detect and react to changes.

IV. THE PROPOSED APPROACH

In this paper the allocation function $A : V \rightarrow P$ considers $|V| = |P| = N$ and dynamic instances of TAP. Two FIFO queues are then used; one for service offer agents (O) and other for service demand agents (D). This approach minimizes the agent waiting time as it avoids undesirable conditions (e.g., customers far from a fleet of cabs would take too long to be served or even be served because large distances decrease the solution’s quality). Additionally, as every cab has a customer to serve, this strategy allows a better work distribution among the cabs. As the offer agents can meet only one demand agent at a time, and a demand agent must be served by one (and only one) offer agent, $N$ is calculated after each detected change (in the episode transition as will be discussed later), as follows:

$$N = \min\{|O^e|, |D^e|\},$$

where $|O^e|$ and $|D^e|$ are the total of elements at episode $e$ in queues $O^e$ and $D^e$, respectively.

The first $N$ offer and demand agents are thus selected from their respective queues according to the order of inclusion to form the set of offer agents ($V^e$) and demand agents ($P^e$) at episode $e$. $V^e = \{o_1^e, ..., o_N^e\}$, $P^e = \{d_1^e, ..., d_N^e\}$.

Sections IV-A and IV-B describe the encoding scheme and algorithm of the Dynamic Particle Swarm Optimization with Permutation (DPSO-P) being proposed.

A. DPSO-P Principles

The DPSO-P encodes particles as position permutations as described in [2]. The particle’s position is represented by a vector $x_k^t = (x_1, x_2, ..., x_n)$ of integer numbers whose indexes identify the service offer agents and whose values represent the demand agents. In the example of Table I, cab 1 would be allocated to customer 4, cab 2 would be allocated to customer 3, and so on.

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

The velocity vector $v_k^t$ is calculated in the usual way as shown in Equation 2.

$$v_k^t = wv_k^{t-1} + c_1r_1(pbest_k - x_k^{t-1}) + c_2r_2(gbest - x_k^{t-1})$$

In the DPSO-P algorithm, the inertia also suffers a decay given by Equation 3.

$$w = \frac{(T - it) \ast (w_{final} - w_{init}) + w_{init}}{T}, \quad it = 1, ..., T,$$

where $T$ is the average number of iterations between detected changes. At the first step of simulation, $T$ is fixed according to the problem size, because there are no changes yet; after this, $T$ is adjusted at each occurred change. The inertia decay thus occurs along time windows.
The particle velocity is then normalized into $[0, 1]$, with each element indicating the probability of a swap operation. The particle movement is illustrated in Figure 1. If a swap is supposed to happen in position $i$ of $x_i^k$ (e.g., the position associated with value 2 in the vector $x$ depicted in Figure 1), $gbest(i)$ is identified (8 in our case). The position $j$ to be swapped with $i$ in $x_i^k$ is that which stores $gbest(j)$. The values stored in positions $i$ and $j$ of $x_i^k$ are then swapped and the new particle position $x_i^{k+1} = x_i^k + v_i^k$ is derived.

\[
\begin{array}{cccccc}
\text{v} & 0.15 & 0.22 & 0.63 & 0.94 & 0.51 \\
\text{x} & 8 & 1 & 3 & 4 & 2 \\
\text{gbest} & 7 & 2 & 3 & 1 & 8 \\
\text{x + v} & 2 & 1 & 3 & 4 & 8
\end{array}
\]

Figure 1. DPSO-P Particle Movement

B. DPSO-P Algorithm

The DPSO-P algorithm (Algorithm 1) addresses problem dynamics by including different techniques to detect and respond to changes in dynamic environment.

The algorithm starts by creating random solutions to the problem i.e initializing the swarm. In the sequence, a random monitoredParticle solution is created to support a change detection. The method used to detect changes is that suggested in [17], it chooses a random particle and checks if its fitness has changed even if the particle position has not changed. Experiments have shown that this method is able to detect any change to the addressed problem, including those considered smooth.

The proposed algorithm tries to detect changes and apply the approaches $A$ and $B$ to react to them. Additionally, the approach $C$ is applied to maintain diversity in the swarm. These approaches are explained below:

- Approach $A$: restart to random positions a portion of the swarm, whenever a change is detected [19] [20]. The particles chosen to be reset are those with the worst fitness value. We tested different rates (see Section V);
- Approach $B$: reset $pbest$ memory of all particles in the swarm, whenever a change is detected [19];
- Approach $C$: uses a perturbation operator on particles position [21]. At each iteration there is a chance of applying randomly swap operations. We tested different rates, that are also described in the next section.

Algorithm 1 DPSO-P algorithm (for a maximization problem)

1: for $k = 1$ to $NParticles$ do
2: $x_i^k$ ← a random solution
3: $v_i^k$ ← a random velocity
4: $pbest_i$ ← $x_i^k$
5: end for
6: $gbest$ ← the better solution $∈ \{x_i, x_{NParticles}\}$
7: monitoredParticle ← a random solution $∉ \{x_i, x_{NParticles}\}$
8: monitoredFitness ← fitness(monitoredParticle)
9: while not reach the simulation time do
10: if fitness($monitoredParticle$) $<>$ monitoredFitness then
11: monitoredFitness ← fitness($monitoredParticle$)
12: $t = 1$
13: if approach $A$ then
14: randomWorstParticles($x_i^1$)
15: end if
16: if approach $B$ then
17: resetParticlesMemory($pbest_i$)
18: end if
19: if $k = 1$ to $NParticles$ do
20: $w = (1-\alpha(w_{end} - w_{begin})) + w_{begin}$
21: $v_i^k = w v_i^{k-1} + c1 r1 (pbest_i - x_i^k) + c2 r2 (gbest - x_i^{k-1})$
22: $x_i^k = x_i^{k-1} + N(v_i^k)$
23: if approach $C$ then
24: $x_i^k$ ← perturbation($x_i^k$)
25: end if
26: if fitness($x_i^k$) $>$ fitness($pbest_i$) then
27: $pbest_i$ ← $x_i^k$
28: end if
29: if fitness($x_i^k$) $>$ fitness($gbest$) then
30: $gbest$ ← $x_i^k$
31: end if
32: end for
33: $t = t + 1$
34: end while

Besides the inclusion of previously described techniques, other modification concerns the stop condition. As the goal for optimization in dynamic environments is not only to find the optimal solution, but also follow a new optimal solution whenever the environment changes, in the algorithm DPSO-P, the stop condition is achieved only when the simulation finishes.

V. EXPERIMENTS AND RESULTS

In this section, experiments are conducted to simulate CCAP/TAP dynamics. The simulation process considers a series of episode transitions (each episode presents a percentage of change in the environment when compared with a previous one). For reference, the program used in the experiments is coded in Java and run on a computer with an Intel Core 2 Duo 2.40GHz and 3GB of RAM.

A. Approach and Simulation Parameters

PSO parameters are described in Table II. Some values are based on two specific works described in literature [24], [25] while others are values usually adopted in many PSO applications.

As shown in Table III, the test bed encompasses four different instances produced to simulate a real scenario of the
is the worst fitness value found at episode $e$. This is the optimal value for a reference solution (provided by Exhaustive Search or robust PSO), all of them found at the end of time window:

$$\text{error} = \frac{1}{E} \sum_{e=1}^{E} (F(\text{best}_e) - F(\text{refer}_e)), \quad (4)$$

where $E$ is the number of episodes, $F(\text{best}_e)$ is the fitness of best solution and $F(\text{refer}_e)$ is the optimal (or reference) value of episode $e$, provided by the comparison approach (ES or robust PSO).

- Average accuracy [23] that indicates the average difference between the best and worst fitness, divided by the difference between reference and worst fitness values:

$$\text{accuracy} = \frac{1}{E} \sum_{e=1}^{E} \frac{F(\text{best}_e) - F(\text{worst}_e)}{F(\text{refer}_e) - F(\text{worst}_e)}, \quad (5)$$

where $F(\text{worst}_t)$ is the worst fitness value found at episode $t$.

- Average stability [23] measures the algorithm ability to track the optimal movement, and is calculated as the average difference of accuracy among episodes (Equation 6).

$$\text{stability} = \frac{1}{E} \sum_{e=1}^{E} \max\{0, \text{accuracy}_{e-1} - \text{accuracy}_e\} \quad (6)$$

As previously mentioned, the fitness function is different from [13]. Thus, the DPSO-P average execution time is higher, as shown in Table IV.

Each approach is executed only once, however it runs for a considerable number of episodes and the performance indicators represent averages of values obtained at the end of each time window. Tested approaches and results are shown in Table IV.

### Table IV

<table>
<thead>
<tr>
<th>Approach</th>
<th>N</th>
<th>Scale</th>
<th>E</th>
<th>Time Window / Episode</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>100</td>
<td>10</td>
<td>3.5</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>3.5</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>315</td>
</tr>
</tbody>
</table>

The robust PSO differs from DPSO-P as it has a larger swarm and runs for a higher number of iterations. A larger swarm provides more particles looking for solutions in the search space, improving this way the final solution qualities. Thus, the robust version is supposed to obtain better solutions than DPSO-P, though with higher computational cost. The robust PSO uses the same encoding scheme of DPSO-P, evolves along 10,000 iterations and has 10,000 particles in the swarm.

### C. Obtained Results

To compare all approaches based on the DPSO-P algorithm with ES and robust PSO we use the following performance measures:

- Fitness Error [22] that indicates the average difference between the fitness of the best solution, and the fitness of a reference solution (provided by Exhaustive Search or robust PSO), all of them found at the end of time window:

$$\text{error} = \frac{1}{E} \sum_{e=1}^{E} (F(\text{best}_e) - F(\text{refer}_e)), \quad (4)$$

where $E$ is the number of episodes, $F(\text{best}_e)$ is the fitness of best solution and $F(\text{refer}_e)$ is the optimal (or reference) value of episode $e$, provided by the comparison approach (ES or robust PSO).

- Average accuracy [23] that indicates the average difference between the best and worst fitness, divided by the difference between reference and worst fitness values:

$$\text{accuracy} = \frac{1}{E} \sum_{e=1}^{E} \frac{F(\text{best}_e) - F(\text{worst}_e)}{F(\text{refer}_e) - F(\text{worst}_e)}, \quad (5)$$

where $F(\text{worst}_t)$ is the worst fitness value found at episode $t$.

- Average stability [23] measures the algorithm ability to track the optimal movement, and is calculated as the average difference of accuracy among episodes (Equation 6).

$$\text{stability} = \frac{1}{E} \sum_{e=1}^{E} \max\{0, \text{accuracy}_{e-1} - \text{accuracy}_e\} \quad (6)$$

As previously mentioned, the fitness function is different from [13]. Thus, the DPSO-P average execution time is higher, as shown in Table IV.

Each approach is executed only once, however it runs for a considerable number of episodes and the performance indicators represent averages of values obtained at the end of each time window. Tested approaches and results are shown in Table IV.

### B. Comparison Approaches

During experiments, the DPSO-P algorithm was implemented in different versions (A, B, and so on...) that are compared with two alternative approaches: an Exhaustive Search (ES) described in [13] and a robust PSO proposed in this paper for comparison purposes.

The robust PSO differs from DPSO-P as it has a larger swarm and runs for a higher number of iterations. A larger swarm provides more particles looking for solutions in the search space, improving this way the final solution qualities. Thus, the robust version is supposed to obtain better solutions than DPSO-P, though with higher computational cost. The robust PSO uses the same encoding scheme of DPSO-P, evolves along 10,000 iterations and has 10,000 particles in the swarm.

### Table III

<table>
<thead>
<tr>
<th>Instance</th>
<th>N</th>
<th>Scale</th>
<th>E</th>
<th>Time Window / Episode</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10M1</td>
<td>10</td>
<td>100</td>
<td>10</td>
<td>3.5</td>
</tr>
<tr>
<td>N10M5</td>
<td>10</td>
<td>100</td>
<td>5</td>
<td>3.5</td>
</tr>
<tr>
<td>N100M10</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>315</td>
</tr>
<tr>
<td>N100M50</td>
<td>100</td>
<td>50</td>
<td>10</td>
<td>315</td>
</tr>
</tbody>
</table>

The developed simulator applies changes to the initial episode, while the DPSO-P algorithm runs in the background tracking those changes. Each approach runs along transitions of all successor episodes for each instance considered. Though some changes are artificially generated by the simulator, observe that allocations resulting from the optimization process also produce changes. Additionally, geographic movements of cabs toward allocated customers are also simulated, producing other changes.

### 3.5 Results

Tests were performed in different versions (A, B, and so on...) that are considered. Though some changes are artificially generated by the simulator, observe that allocations resulting from the optimization process also produce changes. Additionally, geographic movements of cabs toward allocated customers are also simulated, producing other changes.
in Tables V, and VI. Note that column Error presents the difference between each approach of the DPSO algorithm and the result provided by the comparison approach (ES or robust PSO).

### Table V

**RESULTS FOR INSTANCES WITH N = 10**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Approach</th>
<th>Error</th>
<th>Accuracy</th>
<th>Stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Diversity 100% of the swarm</td>
<td>0.00172</td>
<td>0.53722</td>
<td>0.10910</td>
</tr>
<tr>
<td>A2</td>
<td>Diversity 50% of the swarm</td>
<td>0.00071</td>
<td>0.93910</td>
<td>0.01840</td>
</tr>
<tr>
<td>A3</td>
<td>Diversity 100% of the swarm</td>
<td>0.00091</td>
<td>0.59386</td>
<td>0.00537</td>
</tr>
<tr>
<td>B</td>
<td>New memory of particles</td>
<td>0.00071</td>
<td>0.53722</td>
<td>0.10910</td>
</tr>
</tbody>
</table>

### Table VI

**RESULTS FOR INSTANCES WITH N = 100**

<table>
<thead>
<tr>
<th>Instance</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0.00195</td>
<td>0.00936</td>
<td>0.00372</td>
<td>0.00172</td>
<td>0.00071</td>
<td>0.00091</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.49895</td>
<td>0.49533</td>
<td>0.50739</td>
<td>0.49895</td>
<td>0.49533</td>
<td>0.49895</td>
</tr>
<tr>
<td>Stability</td>
<td>0.50105</td>
<td>0.50467</td>
<td>0.49261</td>
<td>0.50105</td>
<td>0.50467</td>
<td>0.49261</td>
</tr>
</tbody>
</table>

Due to the evolutionary aspect of values that compose each average reported in column Accuracy, no statistical test is used to compare the different approaches. Observe that approach C1 presents the best results for N = 10, independently of the scale, though we cannot affirm that differences are statistically significant. For N = 100, approach D1 obtained the best results in the case of smooth changes. For hard changes, approach H1 presented the best results.

Results show that, for smaller instances, the best approach reuses information from the previous episodes and with a small perturbation rate reacts to changes better than with higher rates. On the other hand, for more complex instances, part of information about previous episodes must be disregarded, resetting a portion of swarm. If the change is smooth, we can reset a small portion of the swarm; but if the change is significant, the whole swarm must be restarted.

**VI. CONCLUSIONS AND FUTURE WORKS**

In this paper we proposed the use of PSO for a dynamic application of the Task Assignment, the Cab-Customer Allocation Problem.

Dynamic aspects of the problem were considered. The proposed DPSO-P algorithm used permutations to encode particles and adopted some approaches to respond to changes, namely: (A) randomizes a portion of the swarm whenever any change is detected; (B) resets all particles memory; and (C) applies a perturbation operator with a certain probability. These approaches were also combined, to improve the results.

The experiments showed that, in the case of smaller instances, diversifying a portion (10%) of the swarm can be efficient. In the case of larger instances, this approach is also efficient, but it can be improved using a perturbation operator on particle positions, to maintain diversity in the swarm. The obtained results allow us to conclude that the combination of traditional methods enables the PSO to track the optimal
solution after the occurrence of changes in decision variables of the problem; and the information about the problem size and change scale can be used to determine the appropriate approach to be used.

In future work, we intend to formally model dynamical aspects of the addressed problem. Moreover, the proposed approach can be extended to consider $|V| \neq |P|$ (different number of cabs and customers). Additionally, desirable statistical treatment of results will be included. Finally, other similar applications could be also handled, for example: optimization times of busses and subways; allocation of police vehicles for the care of occurrences; allocation of technical assistance for technical support.

REFERENCES


