A Swarm Intelligence Method Applied to Resources Allocation Problem

Cédric Leboucher[1], Rachid Chelouah[1], Patrick Siarry[2], Stéphane Le Menec[3]

[1] L@ris, EISTI, Avenue du Parc,
95011 Cergy-Pontoise, France
cedric.leboucher@eisti.fr, rachid.chelouah@eisti.fr

[2] Université Paris-Est Créteil (UPEC)
61 av. du Général de Gaulle,
94010 Créteil, France
siarry@u-pec.fr

[3] MBDA France
1, rue Réaumur,
92350 Le Plessis-Robinson, France
stephane.le-menec@mbda-systems.com

Abstract

This paper presents a suggested method to solve an allocation problem with a swarm intelligence method. The application of swarm intelligence has to be discrete. This allocation problem can be modelled like a multi-objective optimization problem where we want to minimize the time and the distance of the total travel in a logistic context. To treat such a problem we are presenting a Discrete Particle Swarm Optimization (DPSO) method in which we adapt the movement of the particles according to the constraints of our application. To test this algorithm, we create a problem whose solution is already known. The aim of this study is to check whether this adapted DPSO method succeeds in providing an optimal solution for general allocation problems and to evaluate the efficiency of convergence towards the solution.

By the way, for comparison purpose, we also applied evolutionary game techniques on the same example. Tentative allocation plans are strategies. Evolutionary game theory studies the behavior of large populations of agents who repeatedly engage in strategic interactions. Changes in behavior in these populations are driven by natural selection via differences in birth and death rates. We focused on replicator dynamic which is a fundamental deterministic evolutionary dynamic for games.

Keywords

Resources Allocation, Discrete Particle Swarm Intelligence, Hard-constrained environment, Evolutionary Game Theory, Replicator dynamic

1 Introduction

In this paper, we propose the use of swarm intelligence to solve an allocation problem. We explore a well-known method of swarm intelligence: DPSO. For the last decade, the study of animal behaviour has been the subject of active research. Numerous scientists have been inspired by the modelling of social interactions to solve NP problems. Even though the communication among the different agents is limited to an exchange of basic information, it results in a very effective team work that is definitely worth modelling. For example, let's consider a fish shoal. Each fish shares information with a small number of neighbours, those at his left and right sides and ahead of it. Knowing the movement of the neighbours is the simple information upon which a shoal fish bases its strategy to avoid predators. As an isolated individual, a fish may, or may not, have the ability to escape a predator, depending on the context. As a member of a group, the fish makes up a collective intelligence that reduces its vulnerability. The aim of the creators of PSO method (Kennedy and Eberhart) was to reproduce this social interaction among agents in order to solve combinatorial problems. Even though the meta-heuristics methods prove efficient and give satisfactory solutions, the contribution of PSO enables to solve many problems with more accurate results than traditional methods like Genetic Algorithms (GA) [1] [4] [11] . Initially, the PSO worked in a continuous space of solutions. It was later adapted to solve problems in a discrete space of solutions [1] [5] [11] [12] [16] [22]. In many industrial contexts, allocation problem arouse a great interest in the operational research community. How to share available resources among a set of agents? If the resource allocation problem can be solved with a variety of techniques, it is still a challenging problem that requires further studies, especially when it diverts to unusual forms. Here we consider a resource allocation problem in hard-constrained environment.

Before looking into the details, let's consider the possible application fields of resource allocation. As it was mentioned before, allocation problems are widely explored by the researcher community, and there are many different real applications using that modelling to solve those problems, from air traffic management [7], enterprise contracting [8], water usage [9], economic applications [13], scheduling problems [17], job shop problem [20], tugboat allocation [21]... The variety of this list shows that there are many application fields to resource allocation

problems. In addition, the specific problem that we propose to solve matches a real industrial requirement: supply chain context. Being studied extensively by the scientific community and in a large field of applications, in this paper, we propose to focus on a side of the problem worth looking into : we will use a Discrete Particle Swarm Optimization method (DPSO) to solve this hard-constraint and combinatorial problems. DPSO is derived of the continuous PSO [2], and, nowadays, it is increasingly used to solve combinatorial problems. The basic principles of DPSO are well described in [10]. In the literature, the use of DPSO is encountered in a variety of areas such as systems of identification [1], budget allocation [5], scheduling problems [6] [11] [16] [22], but also the travelling salesman problem (TSP) [18].

To our knowledge, only a few researches are taking up with resources allocation problems in hard-constrained environment. That adapted method to classic DPSO enables to solve many resource allocation problems while adapting only function evaluations and constraints.

This document is organized as follows: in the first section, we introduce the adapted PSO in discrete context. In the second section, we describe the problem that we propose to solve. We give a mathematical modelling of that problem in the third section. The proposed method is described in section four. And the fifth section consists in an application and the results that we obtain. At last, we conclude and envision a few perspectives.

2 Background of DPSO

As in the PSO method, we have to define the particles, their velocities and moving rules. In this description we use the same notations as [11]. It is important to mention that, so as to match our discrete requirement, the particles have to be described with binary components and the velocity is modelled by a probability for one bit to become a 1.

Let's consider X_i^t a particle, where X_i^t is the i^{th} particle at the t^{th} iteration.

Let $X_i^t = [x_{i1}^t, x_{i2}^t, ..., x_{iD}^t]$, $x_{id}^t \in \{0, 1\}$ be a particle in a population of P particles, and composed by D bits. We denote the velocity by $V_i^t = [v_{i1}^t, v_{i2}^t, ..., v_{iD}^t]$, $v_{id}^t \in \mathbb{R}$. Then, as in the PSO method described in [2], the last step is to define the best position $P_i^t = [P_{i1}^t, P_{i2}^t, ..., P_{iD}^t]$ for the i^{th} particle, and the best position $P_g^t = [P_{g1}^t, P_{g2}^t, ..., P_{gD}^t]$ of the entire population at the iteration t.

As in continuous PSO, each particle adjusts its velocity according to social and selfish criteria. We can define the velocity of a particle i in the direction d with: $v_{id}^{t+1} = v_{id}^t + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (p_{gd}^t - x_{id}^t)$, where c_1 is called the cognition learning factor, and c_2 is called the social learning factor. r_1 and r_2 are random numbers uniformly distributed in $[0, 1]$. Both of the coefficients c_1 and c_2 are fixed and represent the weights of the two components. In order to determine the probability that a bit changes, the last step is the conversion of the velocity into a probability through applying a sigmoid function. Thus we have:

$$s(v_{id}^t) = \frac{1}{1 + e^{-v_{id}^t}}, (\forall v_{id}^t \in \mathbb{R})(s(v_{id}^t) \in [0, 1])$$

Here, $s(v_{id}^t)$ represents the probability of a bit x_{id}^t to become 1 at the next iteration. The process is well described in [11] by a pseudo-code. This pseudo-code is the one developed by Kennedy, Eberhart and Shi for DPSO.

3 Description of the problem at hand and proposed method

3.1 Problem description

In this section we extend the method proposed by Kennedy and Eberhart in [10] to solve a resource allocation problem with strong constraints and real-time components. We consider a supply chain context where we dispose of five delivery resources for three customers. The aim is to determine the best resource allocation to respond to several objectives.

The objective functions: we are considering a multi-objective problem, in which we want to minimize the global distance travelled by our delivery resources and minimize the time of delivery. The objective function is built as follows: we add the time and the distance from stock centre to customer, multiply each of them by a coefficient, then divide that cost by the priority rank of the customer.

The figure 1 shows an example of the problem that we want to solve by this DPSO method.

The available resources: we have delivery resources at our disposal, defined by several properties. First, a resource is associated to its own stock centre. Second, the resources can join the customers with their own average speed.

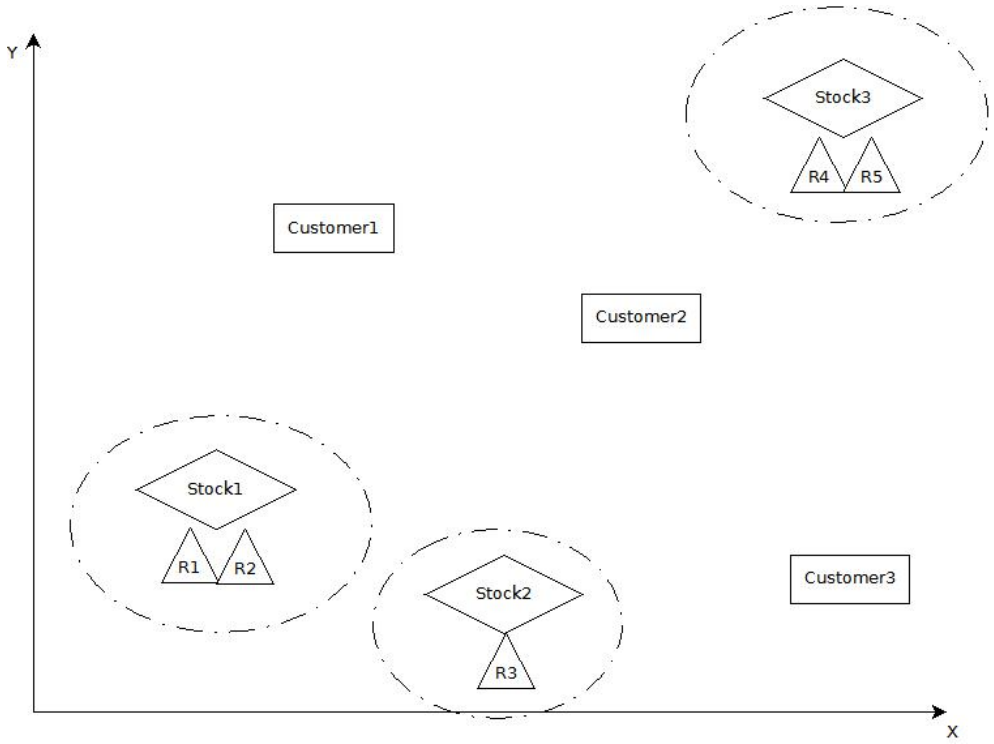

Figure 1: Example of scenario including 3 customers and 5 available resources in 3 different storage centres

The customers: the customers have requirements about the quantity of resources. We also suppose that we know their geographical positions. Besides, customers are not considered as equal and we have ranked them by importance of treatment. We define the priority vector P to express this importance of treatment. For example, if the priority vector is: $P = [2, 3, 1]$, it means the customer 1 is ranked second, the customer 2 is ranked third, and the customer 3 is ranked first. In this vector, the k^{th} element represents the k^{th} customer and the value of $P(k)$ is the priority rank of the k^{th} customer.

The constraints: a single resource cannot be assigned to several customers, and all the resources don't need to be used to solve this resource allocation problem.

Stop criterion: we consider that the algorithm has converged as soon as the number of iterations has been reached.

3.2 Proposed method

We choose to use a DPSO method to solve this problem. First, it is important to state clearly the definition of the particle: in our case, like it has been mentioned in the introduction of DPSO method, we consider that a particle is a binary matrix where: $X^t_{ijk} = 1$, if the resource j is assigned to the customer k, and $X^t_{ijk} = 0$ else.

A particle is a combinatorial solution respecting the system constraints. The aim is to define a set of particles 'moving' on the solution space and interacting together about the quality of the found solution. The proposed algorithm is developed on the same basis that the original work of Kennedy and Eberhart for the DPSO. The adaptation to our application stands in the particle movements. In fact, due to the context , we must respect hard constraints in order to define a move on the solution space. Those constraints that we need to respect are the following: first, a resource can be either unassigned, or assigned to one customer at most. Second, we consider that in the space solution, all the customer requests are covered. To put into practice those constraints, we cannot deal with common DPSO, and in order to adapt that algorithm we proceed as follows: when the sigmoid function provides the probability matrix of move toward a position, we respect the customer priority first. To illustrate this point, let's give an example where we have five resources and three customers:

Initialization of the problem: we initialize a set of particles randomly generated on the space solution, respecting the operational constraints. We have to determine the number of particles, the selfish and the social coefficients.

Evaluation of initial population: we compute the cost of each particle to establish which of them is the best one.

Move of the particles: for each particle, we compute the velocity, the sigmoid, then the new statement of the particle.

Principle: let S be the probability matrix for a 0 to become a 1.

$$S = \begin{pmatrix} 0.2448 & 0.3988 & 0.3577 \\ 0.2057 & 0.8959 & 0.4289 \\ 0.3034 & 0.2993 & 0.9033 \\ 0.8783 & 0.4261 & 0.3701 \\ 0.7912 & 0.3559 & 0.3036 \end{pmatrix}$$

If we consider the priority rank: first, customer 2 (2^{th} column), second, customer 3 (3^{rd} column), third, customer 1 (1^{st} column). The algorithm generates a random number and compares it to the first element of the column. In our example, we have:

Initialization of the particles:

$$X_{init} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

First Step: selection of the most important customer (here, the 2^{th} column).

We generate a random number and compare it to the first element of the column 2. If the random number is lesser than the probability of change, we convert 0 into 1. We repeat that process until the customer request is satisfied.

If we consider that the customer 2 wants only one resource, a possible statement of the new particle could be:

$$X_{init} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Second Step: when the most important customer has his demand satisfied, we take care of the second one. Of course, we have to remove the resources chosen by the first customer.

If we consider that the customer 3 wants two resources, a possible statement of the new particle could be:

$$X_{init} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Third Step: the last step is to match the requirement of the last customer that we consider the least important of the three. We draw a new sequence by the same process as the other customers.

$$X_{init} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Update: we update the best individual and global performance.

Stop criterion: we continue that way until we reach the limit of the maximum iteration defined in the initialization of the simulation.

4 Mathematical modelling

In this section, we will describe the mathematical modelling we use to solve the problem. First, it appears that we have a combinatorial problem where we want to assign resources to a task. The first point of analysis leads to model our problem as a task allocation problem. Thus we can define the starting point of our method. Before looking into the details, let C be the set of customers and R the set of resources. All the elements in R are assigned to a stock centre S_i, where $i \in \mathbb{N}$. We consider that we know the geographical position of each customer and our stock centre in a two dimensional plan.

A particle is defined by a $N \times P$ matrix, where N represents the total number of resources, and P the number of customers.

$$X^t_{i\ (n,p)} = (a^t_{np})_{n=1,...,N,p=1,...,P}, a^t_{np} \in \{0,1\},$$

With the constraints:

$$\begin{cases} \sum_{k=1}^{P} a_{nk} \in \{0,1\}, n \in \{1,2,...,N\} \\ \sum_{k=1}^{N} a_{kp} = d_p, p \in \{1,2,...,P\} \end{cases}$$

The first constraint means a resource can be allocated to only one customer, but also be unassigned. The second one means that all the customer requests have to be covered. Let $c = (c_p)_{p=1..P}, c_p \in \mathbb{N}$ be a vector of requirement by the customers. We denote by $U = (u_p)_{p=1..P}, r_p \in \mathbb{N}$ the vector of priority of treatment. The subscript of the vector represents the number of the customer, and the value of the p^{th} element represents the rank of that customer. All the elements in U are natural numbers and 1 represents the first customer to which we want to provide a resource. Let consider D and T two $N \times P$ matrices. D represent the distance between the n^{th} resource and the p^{th} customer, and T represents the travel time of the n^{th} resource to the p^{th} customer. Those matrices are built as follows:

$$D = (d(r_i, p_j))_{i=1..N,j=1..P},$$

where $d(A, B)$ represents the euclidean distance of the point A to B.

4.1 Objective function

Now, let's describe the objective function: we want to minimize the cost of a solution evaluated by:

$$F(f_1, f_2) = \alpha_1 f_1 + \alpha_2 f_2, (\alpha_1, \alpha_2) \in \mathbb{R}$$

Thus we have a multi-objective problem, and the aim is to minimize F. Now let $v^{t+1}_{id} = v^t_{id} + c_1 r_1 (p^t_{id} - x^t_{id}) + c_2 r_2 (p^t_{gd} - x^t_{id})$ be the velocity of the particle. We use the same notation that Kennedy and Eberhart mentioned in [11]. Thus, we obtain a matrix V^t_i having the same dimension as any particle X^t_i.

$$V^t_{i\ (n,p)} = (v^t_{np})_{n=1,...,N,p=1,...,P}, v^t_{np} \in \mathbb{R}$$

Then, we want to express those velocities as a probability of change for each element in the matrix X^t_i. To convert a velocity matrix into a probability matrix, we apply a sigmoid function. Using a sigmoid function, the probability to change an assignment will be all the higher as the distance with the optimal solution is important. Otherwise, it enables to keep an assignment if it is close to the best solution. The sigmoid function has the form:

$$s(t) = \frac{1}{1 + e^{-\lambda t}}, \lambda \in \mathbb{R}$$

The important point is that the sigmoid function has its value in $[0, 1]$. Therefore we can consider that it defines a probability. This leads to one of the last questions: how are the particles moving? First, we define the matrix of probability generated by applying the sigmoid function to the V^t_i matrix.

$$s(v^t_{inp}) = \frac{1}{1 + e^{-v^t_{inp}}}$$

i represents the i^{th} particle, n the n^{th} resource, and p the p^{th} customer. We can build the matrix $s(V^t_i)$:

$$s(V^t_{i\ (n,p)}) = \left(s(v^t_{np})\right)_{n=1,...,N,p=1,...,P}, v^t_{np} \in \mathbb{R}$$

The coefficients in that matrix represent the probability that a bit changes (0 into 1). The last step of the method consists in generating a random number in uniform distribution. Then we compare this number to the matrix coefficient to find out if the particle will move towards another direction at the next iteration. Since we must comply to special operator requirements, we will check if the resource quantity has been reached after each change of a bit.

5 Algorithm proposition

In this section, we describe the pseudo-code used to solve the problem. In fact, the method we use is very similar to the one elaborated by Liaoa C.J., Tsengb C.T. and Luarnb P in [11], but in the case of our problem requirements some constraints have been added, and the particle sequence construction is quite different, as described in the algorithm 1.

Algorithm 1 DPSO(nb_exp, nb_iterations, r1, r2, nb_particules)

```
Initialisation
for 1 to NbMax_iteration do
  if Stop_Criterion then
    break;
  else
    for All_Particles do
      V(Particle)
      S(V)
      X ← gener_Constrained_newParticle
      Cost(X)
      if Cost(X) > Best_IndivFitness then
        Best_IndivFitness ← X
      end if
      if Cost(X) > Best_GlobalFitness then
        Best_GlobalFitness ← X
      end if
    end for
  end if
end for
```

6 Application

6.1 Assumption

Now, using the same notations as those introduced in the mathematical modelling section, let's describe an example in which the initial conditions are the following : there are $P = 3$ customers who are located in $C_1(2.5, 12)$, $C_2(4.1, 8.3)$ and $C_3(18.3, 4.9)$. The stock centres are located in: $S_1(1.1, 0.5)$, $S_2(10, 2.2)$, and $S_3(8.9, 15)$ (all the coordinates are given in the (x, y) plan). There are $N = 5$ delivery resources. $(r_1, r_2) \in S_1$, $(r_3) \in S_2$, and $(r_4, r_5) \in S_3$. The priority vector is defined as follows: $U = [2, 3, 1]$.

Thus we can compute the distance matrix D:

$$D = \begin{pmatrix} 11.5849 & 8.3570 & 17.7539 \\ 11.5849 & 8.3570 & 17.7539 \\ 12.3406 & 8.4865 & 8.7281 \\ 7.0682 & 8.2420 & 13.7975 \\ 7.0682 & 8.2420 & 13.7975 \end{pmatrix}$$

Then, if we consider the average speed of each delivery resource: $w = [5, 7, 9, 5, 9]$, we obtain the following time matrix: T:

$$T = \begin{pmatrix} 2.3170 & 1.6714 & 3.5508 \\ 1.6550 & 1.1939 & 2.5363 \\ 1.3712 & 0.9429 & 0.9698 \\ 1.4136 & 1.6484 & 2.7595 \\ 0.7854 & 0.9158 & 1.5331 \end{pmatrix}$$

After initializing the parameters, we can define the cost function as:

$$F(X_i^t) = \alpha_1 \sum_{i=1}^{5} \sum_{j=1}^{3} \left(\frac{T(i,j)}{U(j)} \right) X_i^t(i,j) + \alpha_2 \sum_{i=1}^{5} \sum_{j=1}^{3} \left(\frac{D(i,j)}{U(j)} \right) X_i^t(i,j), \alpha_1 = 0.4, \alpha_2 = 0.6,$$

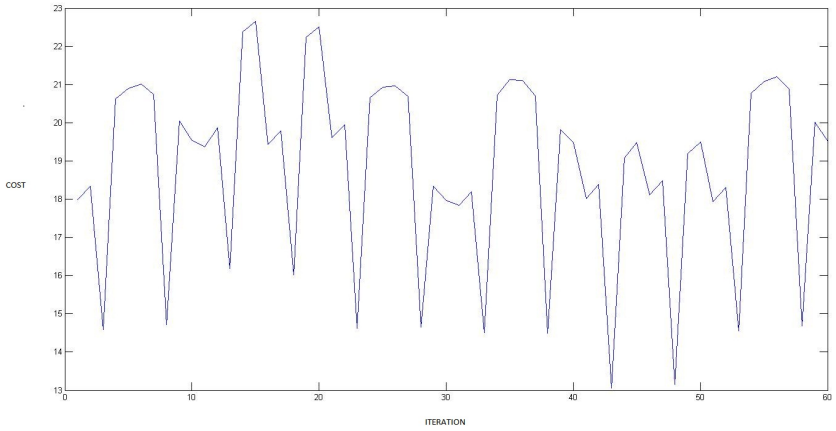

Figure 2: Evolution of the cost of the solution for all the possible permutations.

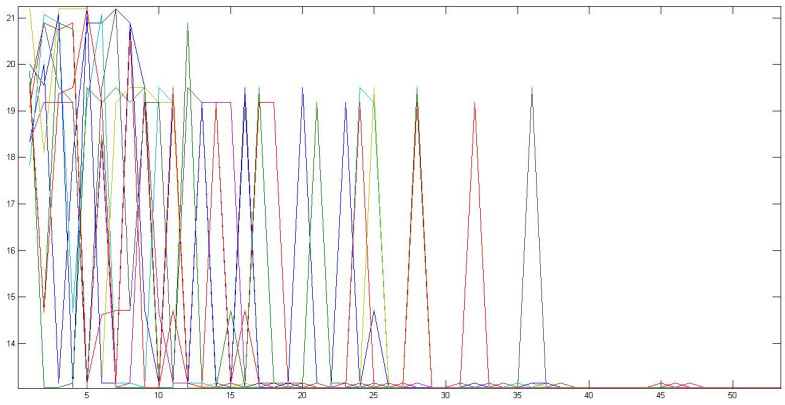

Figure 3: Evolution of the cost of particles. Each curve represents a particle. After 50 iterations, the swarm is stabilised.

6.2 Results and comments

We want to make sure that the results that we obtain match the expected solution. If we compute all the space solutions and evaluate each of them, we obtain the graph 2. Thus, we reckon that the optimal solution for the particle is: $X = [0, 2, 3, 1, 1]$ and $F([0, 2, 3, 1, 1]) = 13.05141$. First, let's use our algorithm to obtain the result of the application. The coefficients c_1 and c_2 are respectively 0.01 and 0.05. We remember that c_1 represents the selfishness of the particle to follow its own best way and c_2 represents the social coefficient. In our case, we have chosen $c_2 = 5c_1$ because it appeared to be the best solution after a set of numerous trials. We decided to generate 10 particles and we chose a number of iterations equal to 500 so as to be sure that all the particles would converge towards the optimal solution.

As we can see on the graph of the figure 3, all the particles are effectively converging towards the optimal solution and they are stabilising in those positions. In order to study the global convergence of the swarm, we can compute the mean of the solution found by each particle. Thus we obtain the graph represented on the figure 4. This graph reveals a fast convergence. In addition, it explores a wide range of the solution space, as it appears clearly in the amplitude of the oscillations. To make the reading clearer, we draw on the same graph the average move on ten values (on the figure 4). Now, the aim is to repeat the simulation many times, check the results each time, and make sure that we can consider the algorithm reliable for that kind of problem. Let's see what the results are if we launch the algorithm 500 times: the figure 5 shows two graphs: the first illustrates the value of the solution obtained in each simulation. The second tests the convergence of the solution. In fact, we reckon the number of the different solutions obtained after 500 iterations. The regularity of the results proves that the method is reliable. It is also very fast, since the time of computing doesn't exceed 100 ms. Thus, we can conclude that the results are satisfying and the adapted DPSO works out properly to solve allocation problems.

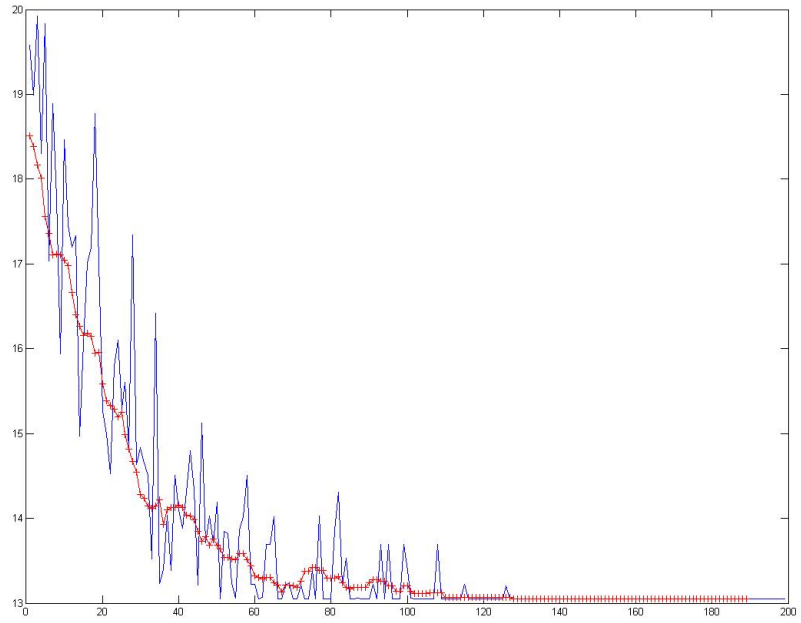

Figure 4: Evolution of the mean for each iteration and evolution of the average move

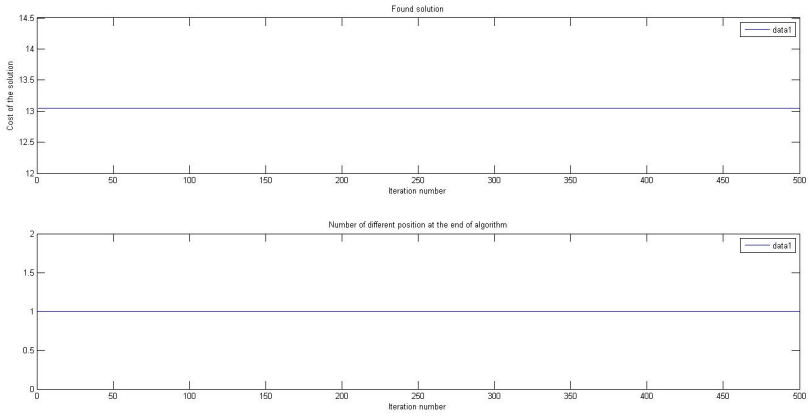

Figure 5: The upper graphic shows the value of the found solution. The lower graphic shows the number of convergence points after 500 iterations

7 Resource allocation optimization using evolutionary game theory

7.1 Evolutionary stable strategies

An Evolutionary Stable Strategy (ESS) is a strategy such that, if all members of a population adopt it, then no mutant strategy could invade the population under the influence of natural selection [3, 15]. Assume we have a mixed population consisting of mostly p^* individuals (agents playing optimal strategy p^*) with a few individuals using strategy p. That is, the strategy distribution in the population is:

$$(1-\varepsilon)p^* + \varepsilon p$$

where $\varepsilon > 0$ is the small frequency of p users in the population. Let the fitness, i.e. payoff of an individual using strategy q in this mixed population, be

$$\pi(q, (1-\varepsilon)p^* + \varepsilon p).$$

Then, an interpretation of Maynard Smith's requirement [14] for p^* to be an ESS is that, for all $p \neq p^*$,

$$\pi(p, (1-\varepsilon)p^* + \varepsilon p) > \pi(p^*, (1-\varepsilon)p^* + \varepsilon p)$$

for all $\varepsilon > 0$ sufficiently small; for agents minimizing their fitness.

7.2 Replicator dynamic

A common way to describe strategy interactions is using matrix games. Matrix games are described using notations as follows. e_i is the i^{th} unit vector of $i = 1, ..., m$. $A_{ij} = \pi(e_i, e_j)$ is the $m \times m$ payoff matrix. $\Delta^m \equiv \{p = (p_1, ..., p_m) \mid p_1 + ... + p_m = 1,\ 0 \leq p_i \leq 1\}$ is the set of mixed strategies (probability distributions over the pure strategies e_i). Then, $\pi(p, q) = p \cdot Aq$ is the payoff of agents playing strategy p facing agents playing strategy q. Another interpretation is $\pi(p, q)$ being the fitness of a large population of agents playing pure strategies (p describing the agent proportion in each behavior inside a population) respect to a large q population.

The replicator equation (RE) is an Ordinary Differential Equation expressing the difference between the fitness of a strategy and the average fitness in the population. Lower payoffs (agents are minimizers) bring faster reproduction in accordance with Darwinian natural selection process.

$$\dot{p}_i = -p_i(e_i \cdot A_p - p \cdot Ap)$$

RE for $i = 1, ..., m$ describes the evolution of strategy frequencies p_i. Moreover, for every initial strategy distribution $p(0) \in \delta^m$, there is an unique solution $p(t) \in \delta^m$ for all $t \geq 0$ that satisfies the replicator equation. The replicator equation is the most widely used evolutionary dynamics. It was introduced for matrix games by Taylor and Jonker [19].

8 Results applying replicator dynamic

We formed an evolutionary game based on the resource allocation problem described previously. We aim finding a solution that minimizes the objective function F (see section 4.1). Pure strategies are all the allocation plans satisfying the problem constraints. We build a $m \times m$ symmetric matrix game with payoff as follows:

$$\pi(e_i, e_j) = \frac{F(e_i) - F(e_j)}{2}$$

Therefore, the payoffs on the matrix diagonal are equal to zero and are solution candidates. We run the RE starting from different initial conditions around the uniform distribution ($p_i = \frac{1}{m}; 1 \leq i \leq m;$). Figure 6 shows results we obtained; trajectories explain the population evolution (probability distribution p and corresponding payoff). The e_i pure strategies (cross marks) are spread all along a circle in the horizontal plan. The vertical axis is payoff $p \cdot Ap$ of the population. We obtain convergence to the best pure optimal allocation plan highlighted by a circle mark in figure 6 whatever the initial conditions are. In Figure 7 we draw the value of objective function F corresponding to all the resource allocation plans satisfying the problem constraints. The minimum value also found by RD is pointed by a vertical line.

Perspectives could be to better understand what are the similarities between particle swarm optimization (PSO) techniques and evolutionary game theory (EGT) and better understand how we could merge both approaches. From the EGT point of view it could be interesting to test such algorithms on larger problems; maybe switch to parallel computation and test other revision protocols for evolutionary dynamics as the best response or imitation protocols.

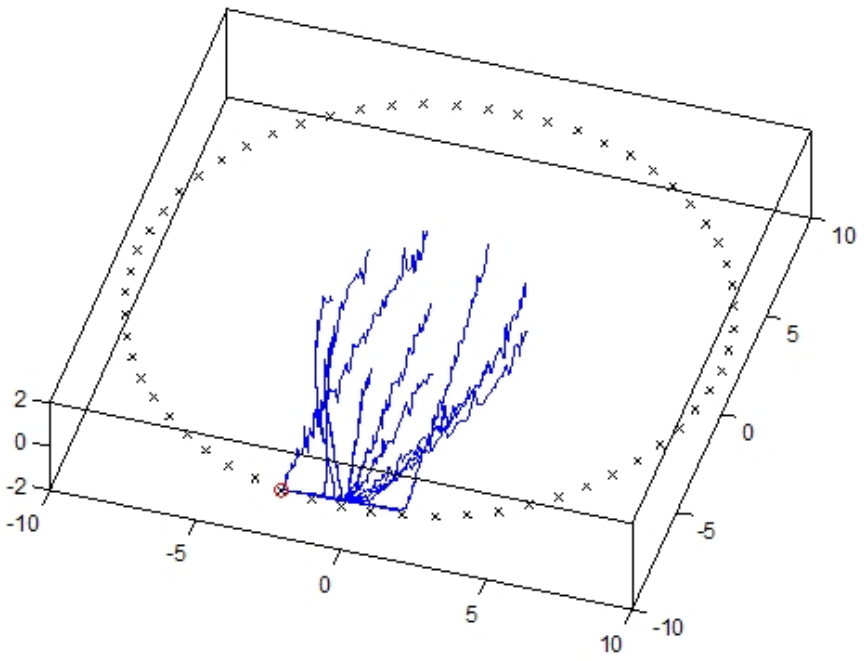

Figure 6: Population evolution according to Replicator Dynamic; vertical axis is proportional to matrix payoff; pure strategies have been drawn on a circle in the horizontal plan; population horizontal plan coordinates have been computed in accordance to p describing the ratio of pure strategies in the population

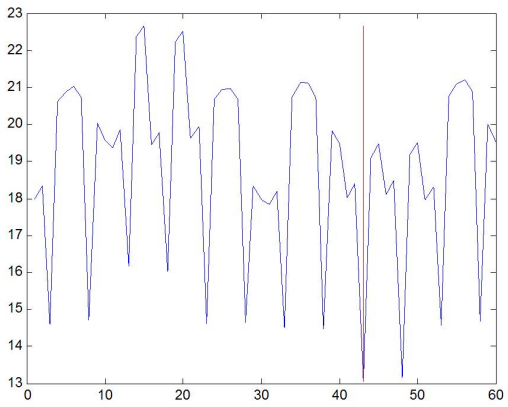

Figure 7: Cost of resource allocation plans; optimal solution tagged by a vertical mark

9 Conclusion and perspectives

In this study we focused on the resource allocation problem through a swarm intelligence method. The adapted DPSO to a hard-constrained environment shows very promising results. The numerous and successful trials show that the method is reliable and provides valuable solutions to the resource allocation problems. The difference between the new algorithm and those that were developed before lies in the movement of the particles through the solution space. Not only do we include the priority rank of the customers, but also the customer requirements. Therefore, we take into account a hard-constrained environment which is closer to a real industrial application. To check the efficiency of the method, we test the algorithm in a supervised environment. The analysis of the results proves that this method is very efficient to solve resource allocation problems. Nevertheless, adapting the selfish and social coefficients according to the problem may prove a tricky task and requires further thinking. The coefficients that we chose enable a good exploration of the solution space and a fast convergence towards the solution. For a real-time requirement we could increase the social coefficient leading to a quicker convergence, but this would present the risk of falling in a local trap. Depending on the application and the experiment, it will take a compromise to find the optimal choice of coefficients.

In the future, it will be interesting to test this algorithm in the case of more sophisticated problems. For example, we could add a time component. If we envision the possibility that the customers have time requirements, we will mix this adapted DPSO and a scheduling problem. It will also be interesting to add a Pareto approach, considering an expert can choose among a set of solutions.

References

[1] M.A. Badamchizadeh and K. Madani. Applying modified discrete particle swarm optimization algorithm and genetic algorithm for system identification. In *Computer and Automation Engineering (ICCAE), 2010*, volume 5, pages 354–358, Singapore (Republic of Singapore), February 2010.

[2] M. Clerc and P. Siarry. Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires. *J3eA*, 3-7:1–13, 2004.

[3] R. Cressman. *Evolutionary Dynamics and Extensive Form Games*. MIT Press, 2003.

[4] R. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming*, pages 611–616, San Diego (USA), March 1998.

[5] T-F Ho, S. J. Jian, and Y-L W. Lin. Discrete particle swarm optimization for materials budget allocation in academic libraries. In *Computational Science and Engineering (CSE), 2010*, pages 196–203, Hong Kong (China), December 2010.

[6] W.B. Hu, J.X. Song, and W.J. Li. A new PSO scheduling simulation algorithm based on an intelligent compensation particle position rounding off. In *ICNC '08 Proceedings of the 2008 Fourth International Conference on Natural Computation*, volume 1, pages 145–149, Jinan (China), October 2008.

[7] U. Junker. Air traffic flow management with heuristic repair. *The Knowledge Engineering Review*, 0:1–24, 2004.

[8] T. Kaihara and S. Fujii. A study on multi-agent based resource allocation mechanism for automated enterprise contracting. In *Emerging Technologies and Factory Automation, 2006. ETFA '06*, pages 567–573, Prague (Czech Republic), September 2006.

[9] D. Karimanzira and M. Jacobi. A feasible and adaptive water-usage prediction and allocation based on a machine learning method. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 324–329, Cambridge (UK), April 2008.

[10] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *The 1997 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4104–4108, Orlando (USA), October 1997.

[11] C.J. Liaoa, C.T. Tsengb, and P. Luarnb. A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, 34:3099–3111, 2007.

[12] P. Liu, L. Wang, X. Ding, and X. Gao. Scheduling of dispatching ready mixed concrete trucks trough discrete particle swarm optimization. In *The 2010 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4086–4090, Istanbul (Turkey), October 2010.

[13] B. Lu and J Ma. A strategy for resource allocation and pricing in grid environment based on economic model. In *Computational Intelligence and Natural Computing, 2009. CINC '09*, volume 1, pages 221–224, Wuhan (China), June 2009.

[14] J. Maynard-Smith. *Evolution and the theory of games*. Cambridge University Press, 1982.

[15] W. Sandholm. *Population Games and Evolutionary Dynamics*. MIT Press, 2010.

[16] D.Y. Shaa and Lin H.H. A multi-objective PSO for job-shop scheduling problems. *Expert Systems with Applications*, 37:1065–1070, 2010.

[17] F. Shaheen and M. B. Bader-El-Den. Co-evolutionary hyper-heuristic method for auction based scheduling. In *Congress on Evolutionary Computation (CEC), 2010*, pages 1–8, Barcelona, July 2010.

[18] X.H. Shi, Y.C. Lianga, H.P. Leeb, C. Lub, and Q.X. Wanga. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103:169–176, 2007.

[19] P. Taylor and Jonker L. Evolutionary stable strategies and game dynamics. *Mathematical Bioscience*, 40:145–156, 1978.

[20] V. V. Toporkov. Optimization of resource allocation in hard-real-time environment. *Journal of Computer Systems Sciences*, 43-3:383–393, 2004.

[21] S. Wang and B. Meng. Chaos particle swarm optimization for resource allocation problem. In *Automation and Logistics, 2007*, pages 464–467, Jinan (China), August 2007.

[22] Y. Yare and G. K. Venayagamoorthy. Optimal scheduling of generator maintenance using modified discrete particle swarm optimization. *Bulk Power System Dynamics and Control - VII. Revitalizing Operational Reliability, 2007 iREP Symposium*, 7:1–7, 2007.