Multi-objective Artificial Bee Colony Algorithm *with external memory* for the Optimization of Engineering Problems

Carlos A. Duchanoy[a] ⋆, Marco A. Moreno-Armendáriz[a], Carlos A. Cruz-Villar[b] and Hiram Calvo[a]

[a] *Instituto Politécnico Nacional, Centro de Investigación en Computación, Av. Juan de Dios Bátiz s/n, México D.F. 07738*
[b] *Sección de Mecatrónica, CINVESTAV, Av. Instituto Politécnico Nacional 2508, Apdo Postal 14740, México D.F., México*

Abstract. In this paper, a memory module for the Multi-Objective Artificial Bee Colony (MOABC) algorithm is proposed. The inclusion of a memory has the objective of reducing the number of evaluations required by the MOABC algorithm. The proposed memory stores the fitness values for every food source, thus avoiding to evaluate more than once these sources. As case study, we present a damper optimal design. With the objective of carrying out an in-depth analysis of the suspension system, the full behavior of the suspension is considered. This is accomplished using a full car model that includes the kinematics, dynamics and geometry of the suspension. However, simulation of the car model requires a lot of computing power which leads to long simulation times. The simulation times motivates the modifications made to the MOABC algorithm. As result of the modifications the number of evaluations needed by the MOABC is reduced to the half.

Keywords: Multi-Objective Artificial Bee Colony, Evolutionary optimization, Suspension System

1 Introduction

Several computational methods have been proposed to find the optimal solution to a problem. In [7] a comparison between methods such as Particle Swarm Optimization (PSO), Genetic Algorithms (GA), Differential Evolution (DE), Evolutionary Algorithms (EA) and Artificial Bee Colony algorithms (ABC) is presented. As conclusion the authors determine that the ABC algorithm

⋆ Corresponding author. Email: duchduchanoy@gmail.com

performs better than the mentioned algorithms and can be efficiently employed to solve multimodal engineering problems with high dimensionality. More studies on the ABC algorithm behavior are enlisted in [8], which is a survey of the ABC algorithm an its applications.

Optimization problems with more than one criterion to be considered are common in many engineering problems. The goal is to find a set of solutions that represents a tradeoff among these criteria. These type of problems are called multi-objective and some authors have proposed modifications to the artificial bee colony algorithm (ABC) in order to solve them. The original ABC algorithm was modified for solving multi-objective problems in [6]. This algorithm is Pareto based and was named as Multi-Objective Artificial Bee Colony (MOABC). A multi-objective ABC algorithm for scheduling problems is introduced in [2]. Finally an improved version of the MOABC algorithm is presented in [1].

Of all the above-mentioned algorithms, which has been applied more to solve the different problems of engineering is the MOABC algorithm [[3] to [7]]. Other example is [5], where the authors proposed the application of a MOABC to solve the motif discovery problem in DNA sequences. Also, in [3] a MOABC for optimization of power and heating system is introduced.

2 Multi-Objective Artificial Bee Colony Algorithm *with external memory*

In this section, the proposed Multi-Objective Artificial Bee Colony Algorithm *with external memory* (MOABCM) is explained in detail. It represents the foraging behavior of honey bees.The colony has three types of bees: employed, onlooker and scout. This algorithm assumes a fixed number of food sources; each food source represents a solution to the problem, these food sources have been found by employed bees and will be presented in the dance area. The onlooker bees wait on the dance area to choose a food source by its quality. When the food source could not be improved after some cycles, a scout bee will do a random search in order to find a new food source.

The MOABCM is a Pareto based algorithm with an external archive used to store non-dominated solutions and a memory that stores the aptitude values of each food source, in order to reduce the number of evaluations of the objective function. This algorithm requires as parameters: Population size, Max-trial, maximum number of iterations. Population size is the number of food sources, employed and onlooker bees, Max-trial is the value that the algorithm uses to find out which food sources should be abandoned. The maximum number of iterations indicates when the algorithm must be stopped (end condition).

2.1 Proposed memory module

The memory has the objective of eliminating the need of evaluating twice a food source. As a result, the number of evaluations of the objective function is considerably reduced. The novel memory stores: All the food parameters (vector ϕ for our case study), the fitness of all the objectives and the number of food sources dominated by each solution. In order to simplify the code this memory is divided into three smaller ones: The first one stores the parameters of all the food sources, it is named $food$, the second stores the fitness for all the objectives and is named *apt* and the last one stores the number of food sources dominated by each solution, this one is called *Dom*. A schematic of the memory is shown in figure 1.

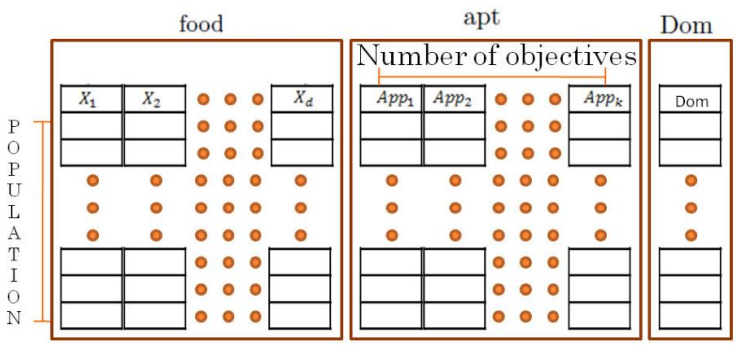


Fig. 1: Schematic of the proposed memory module for MOABCM.

2.2 Update archive

The MOABCM algorithm uses a different structure for the external archive. The new archive holds the best solutions ever found, saving the food parameters in a entity called Non-dom and fitness of the non dominated solutions in a entity called *arch*. The pseudocode for updating the archive is given in algorithm 1; the algorithm makes use of the Dominate function that calculates the Pareto dominance for the given data. This function uses the food memory and the aptitude matrix (apt) and it returns the non-dominated data matrix (Non−dom), their aptitude matrix (arch), and a domination matrix (Dom) in which the program indicates which solution dominates whom; then it saves this information into an external archive.

The update-archive function proposed in this paper also differs from the one proposed in [1] because in the original MOABC the update archive function

Algorithm 1 [arch, Non−dom, Dom]=update−archive(food, apt)

1: [arch, Non−dom, Dom]**Dominate**(apt, food)
2: **Save**(arch, Non−dom)

performs the evaluation of all the food sources while in the MOABCM, the update archive function does not perform evaluations of the fitness function. Instead, the fitness values stored in the memory are used.

2.3 Initialization

In the initialization phase, a new matrix named *food* is defined to preserve the possible solutions. The pseudocode of the initialization module is given in Algorithm 2. It uses the same parameters as the original MOABC. The initialization algorithm uses the population number which is equal to the number of food sources, it also needs the dimension of the problem which is the number of parameters that will be optimized. In this function Lev_{down} and Lev_{up} are the lower and upper bound matrices respectively.

Algorithm 2 [food, trial]=Initialization(dimension, population)

1: **For** i=1:dimension
2: **For** j=1:population
3: $Food(j,i) = Lev_{down}(i) + Rand()(Lev_{up}(i) - Lev_{down}(i))$
4: $trial(i) = 0$
5: **End For**
6: **End For**

2.4 Evaluate

The evaluate function calculates the aptitude of each food source using the fitness function of the problem. This function returns the aptitude value and stores it in the proposed memory, the food memory and the population number.

2.5 Employed−bee

The pseudocode of Employed−Bee function is given in Algorithm 3. The employed bee function improves food sources using the external archive since it contains the best solutions found so far. It is similar to the one proposed in the MOABC algorithm, but differs in line 8 because in order to make the comparison of the original food source with the updated one, it calculates the fitness of both

sources and compares them. The proposed MOABCM instead of evaluating the fitness of the original food source, uses the value stored in the *apt* memory and compares it with the fitness of the updated food source.

Algorithm 3 [food, apt, trial]=Employed−bee(food, apt, trial, population, dimension, Non−dom)

For i=1:population
  D =**Randi**($dimension$)
  N =**Randi**(Non−dom)
  $v = food(i, D) + w_1$(**Rand**()($food(i, D)$−Non−dom(N, D))
  $test = food(i, :)$
  $test(D) = v$
  [apd] =**Evaluate**($test, 1$)
  If apd is better than $apt(i)$
    $food(i, :) = test$
    $apt(i) = apd$
  Else
    $trial(i) = trial(i) + 1$
  End If
End For

The Employed−bee function uses the **Randi**(k) function, this returns a random integer between 0 and $k > 0$ and uses **Rand**() function which returns a random number between 0 and 1.

2.6 Onlooker−bee

After all employed bees optimized their food source by considering the information provided by the external archive, they will share their information with onlooker bees which optimize food sources advertised by employed bees. The pseudocode of the Onlooker−bee module is given in Algorithm 4.

In order to choose which food source advertised by the employed bees will be optimized, the algorithm uses a probabilistic method. For this purpose, the probability for each food source advertised by the corresponding Employed−bee will be calculated with equation (1).

$$P_k = \frac{fit(x_k)}{\sum_{m=1}^{population} fit(x_m)} \tag{1}$$

Where $fit(x_k)$ is the probability of the food source proposed by Employed−bee k which is proportional to the quality of food source. In MOABC, the quality

Algorithm 4 [food, apt, trial]=Onlooker−bee(food, apt, trial, population, dimension, Dom)

1: **For** i=1:population
2: D =**Randi**($dimension$)
3: N =**Roulette**(Dom)
4: $v = food(i, D) + w_1(\textbf{Rand}()(food(i, D) - food(N, D))$
5: $test = food(i, :)$
6: $test(D) = v$
7: $[apd]$ =**Evaluate**($test, 1$)
8: **If** apd is better than $apt(i)$
9: $food(i, :) = test$
10: $apt(i) = apd$
11: **Else**
12: $trial(i) = trial(i) + 1$
13: **End If**
14: **End For**

of a food source depends on the number of food sources dominated by the food source k. This can be formulated using the equation (2).

$$fit(x_k) = \frac{Dom(k)}{population} \tag{2}$$

Where $Dom(m)$ is the number of food sources dominated by food source m. After the probabilities are calculated, the onlooker bees use roulette wheel to chose the food source. After that, each Onlooker−bee optimizes its food source.

The pseudocode of Algorithm 4 is different from the one proposed in MOABC in two parts: the first one is that in line 3, the MOABC in order to calculate the probability of the food source, evaluates the fitness of all food sources and the MOABCM uses the values stored in the Dom memory. The second change is in line 8, the proposed MOABCM instead of evaluating the fitness of the original food source uses the value stored in the apt memory and compares it with the fitness of the updated food source.

2.7 Scout−bee

The pseudocode of Scout−bee module is given in Algorithm 5. This algorithm will find if an employed be should abandon his food source and replace it with a new one. In the MOABCM line 4 of the Algorithm 5, is added in order to save the fitness of the new food source.

Algorithm 5 [food, apt, trial]=Scout−bee(food, apt, trial, population, dimension)

1: **For** i=1:population
2: **If**$(trial(i) >= maxtrial)$
3: $[food(i), trial(i)]$ =**Initialization**$(1, dimension)$
4: $[apt(i)]$ =**Evaluate**$(food(i, 1))$
5: **End If**
6: **End For**

3 Comparison between algorithms

The MOABCM algorithm changes were proposed with the purpose of reducing the number of evaluations needed by the algorithm with very similar performance. Table 1 presents a comparison of the number of needed evaluations in the MOABC and in the MOABCM. This table describes the evaluations made by each module and in the iteration row it enlists the number of evaluations per iteration.

Module	MOABC [1]	MOABCM
update-archive	Population	0
Employed-bee	2 Population	Population
Onlooker-bee	3 Population	Population
Scout-bee	0	0≤x≤Population
Evaluations per iteration	6 Population	max 3 Population

Table 1: Number of evaluations of the fitness function

It is necessary to consider that the number of evaluations made in the scout-bee can be a number between zero and the population number because it only evaluates new food sources. As shown in the last row of Table 1 the MOABCM algorithm in the worst case scenario has to evaluate three times the population while the original MOABC needs six times population evaluations per iteration. This is a considerable improvement because the number of evaluations is reduced to a half.

4 Case study: Optimal Suspension Problem

The suspension system performs two main functions. The first one is to guarantee the correct contact of tires during a travel and its second function is to isolate the passengers from road disturbances. During the design process, these

objectives are in conflict. For this reason, the objective is to design a damper that provides the best compromise between these objectives. In order to perform the optimization process, a complex mathematical model of the vehicle is required. For our case, we select the full car model developed by our research group in [4]. This full car model consists of the front suspension, rear suspension, the tires model and their interaction with the chassis.

The mathematical model is programmed in Matlab Simulink© in order to simulate the behavior of the vehicle. A terrain in which the performance of the suspension system will be measured, also was simulated. The terrain selected for this experiment is conformed by a section of caps, see figure 2.

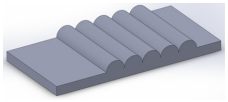


Fig. 2: Schematic diagram of the caps terrain.

The MOABCM algorithm uses the evaluate function to calculate the fitness of the possible solutions. Each time the function calls the simulator and waits for the result. As result the simulator generates a plot of the displacement of the cockpit (x_c) and four plots of contact area of the tires, one for each tire ($A_{ct}(fr), A_{ct}(fl), A_{ct}(rr), A_{ct}(rl)$). The results are used to calculate the fitness of the solutions using equations (3 and 4). The objective of this case study is to design a passive damper which improves the performance of the suspension in two factors: vehicle's safety and comfort. So, we define safety and comfort as follows: it should be understood by comfort C, a reduction in the amplitude of the cockpit displacement and as safety S, an increase of the contact area of the tires. The problem is formulated as the minimization of the chassis displacement and the maximization of the contact area of all tires. The problem is expressed in equations (3) to (9).

$$\min_{\text{under } \phi} C = \int_0^T L_1(x(t), z(t), \theta, \phi, t)dt \tag{3}$$

$$\max_{\text{under } \phi} S = \int_0^T L_2(x(t), z(t), \theta, \phi, t)dt \tag{4}$$

$$\phi = [K_d(fr), K_d(fl), K_d(rr), K_d(rl), B_d(fr), B_d(fl), B_d(rr), B_d(rl)] \tag{5}$$

Subject to:

$$\dot{x} = f(t, x(t), z(t), \theta, \phi) \tag{6}$$
$$0 = g(t, x(t), z(t), \theta, \phi) \tag{7}$$

Where:

$$L_1(x(t), z(t), \theta, \phi, t) = z_c \tag{8}$$
$$L_1(x(t), z(t), \theta, \phi, t) = A_{ct}(fr) + A_{ct}(fl) + A_{ct}(rr) + A_{ct}(rl) \tag{9}$$

We propose as design variables the stiffness of the front suspension damper $K_d(fr), K_d(fl)$, the damping of the front suspension damper $B_d(fr), B_d(fl)$, the stiffness of the rear suspension $K_d(rr), K_d(rl)$ and the damping of the rear suspension damper $B_d(rr), B_d(rl)$. The design variables vector ϕ is expressed in equation (5).

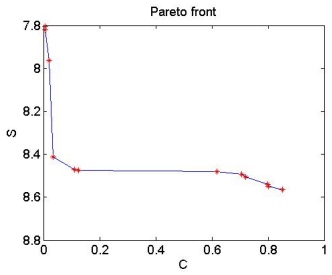
5 Experiments and results

The optimization of the passive dampers for the suspension was made trough the modification of the design variables vector ϕ (5), applying the algorithm described in section 2. The method was performed using the following parameters: Population=60, max-trial=10, max-iterations=40, and $w_1 = 0.8$ (an explanation about how to select this values is founded in [4]). The execution of the algorithm spent around of 30 hrs, with a total of 1420 evaluations of the fitness function and used half of time of the original MOABC (on average, obtained by running the experiment 10 times). The Pareto front obtained as solution for this problem is shown in figure 3a. The algorithm finds 14 possible solutions for the problem that for a numeric optimization are very few, but for our case of study are good enough since they provide a good set of different behaviours to test the optimized suspension.

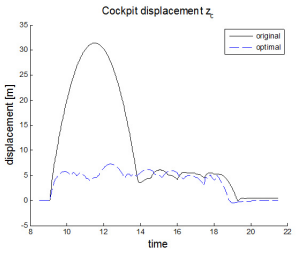
As an example, we select a solution on the middle of the Pareto front (best compromise between comfort C and safety S) and the results are shown in in figures 3b to 3d. Notice that the optimal solution improves the cockpit displacement and reduces the time where the tires do not have contact with the ground.

6 Conclusion and Future Work

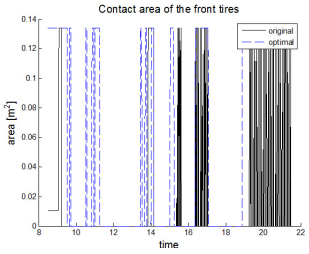
The model proposed for the vehicle is able to more accurately represent its behavior. It is important to highlight that more detailed models consume longer simulation times, which motivates the proposed modifications to the MOABC algorithm. To solve this problem, we present the MOABCM algorithm which is



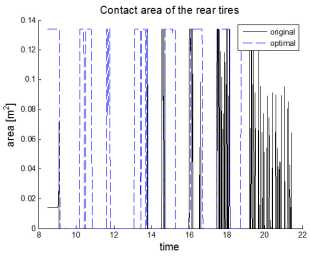
(a) Pareto front.



(b) Cockpit displacement.



(c) Contact area of the front tires.



(d) Contact area of the rear tires.

Fig. 3: Dynamic behavior of the Vehicle.

capable to reduce the number of evaluations of the fitness function. This in turn helps reducing the optimization time to less than half of the time needed by the original MOABC algorithm.
To test the performance of the algorithm a case study of a passive suspension optimization was presented. As shown in Section 5 we could appreciate that the behavior of the two parameters were optimized by the proposed methodology.
As future work remains the construction and testing of the suspension designed with this methodology.

Acknowledgments The authors would like to thank the team members of UPIITA SAE Baja 2009 who made the construction and design of the vehicle in which this work was based. The authors appreciate the support of Mexican Government (SNI, SIP-IPN, COFAA-IPN, BEIFI-IPN and CONACyT).

References

1. Akbari, R., Hedayatzadeh, R., Ziarati, K., Hassanizadeh, B.: A multi-objective artificial bee colony algorithm. Swarm and Evolutionary Computation 2, 39–52 (Feb

2011)
2. Arsuaga-Rios, M., Vega-Rodriguez, M.a., Prieto-Castrillo, F.: Multi-Objective Artificial Bee Colony for scheduling in Grid environments. In: 2011 IEEE Symposium on Swarm Intelligence. pp. 1–7. IEEE (Apr 2011)
3. Atashkari, K., NarimanZadeh, N., Ghavimi, A., Mahmoodabadi, M.J., Aghaienezhad, F.: Multi-objective Optimization of Power and Heating System Based on Artificial Bee Colony. In: International sympo- sium on innovations in intelligent systems and applications (INISTA). pp. 64–68 (2011)
4. Duchanoy, C.A., Moreno Armendaríz, M.A., Cruz-villar, C.A.: Nonlinear Full-car Model for Optimal Dynamic Design of an Automotive Damper. In: Multibody Mechatronic Systems, pp. 489–500. Springer International Publishing (2015)
5. González-Álvarez, D.L., Vega-Rodríguez, M.A., Gómez-Pulido, J.A., Sánchez-Pérez, J.M.: Finding Motifs in DNA Sequences Applying a Multiobjective Artificial Bee Colony (MOABC). Evolutionary computation, machine learning and data mining in bioinformatics. Lecture notes in computer science, 6623, 89–100 (2011)
6. Hedayatzadeh, R., Hasanizadeh, B., Akbari, R., Ziarati, K.: A multi-objective Artificial Bee Colony for optimizing multi-objective problems. In: 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE). vol. 2, pp. 277–281. Ieee (Aug 2010)
7. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8(1), 687–697 (Jan 2008)
8. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review pp. 1–37 (Mar 2012)