Hybrid Ant Colony-Genetic Algorithm (GAAPI) for Global Continuous Optimization

Irina Ciornei, *Student Member, IEEE*, and Elias Kyriakides, *Senior Member, IEEE*

***Abstract*—Many real-life optimization problems often face an increased rank of nonsmoothness (many local minima) which could prevent a search algorithm from moving toward the global solution. Evolution-based algorithms try to deal with this issue. The algorithm proposed in this paper is called GAAPI and is a hybridization between two optimization techniques: a special class of ant colony optimization for continuous domains entitled API and a genetic algorithm (GA). The algorithm adopts the downhill behavior of API (a key characteristic of optimization algorithms) and the good spreading in the solution space of the GA. A probabilistic approach and an empirical comparison study are presented to prove the convergence of the proposed method in solving different classes of complex global continuous optimization problems. Numerical results are reported and compared to the existing results in the literature to validate the feasibility and the effectiveness of the proposed method. The proposed algorithm is shown to be effective and efficient for most of the test functions.**

***Index Terms*—Ant colony optimization (ACO), genetic algorithm (GA), global continuous optimization.**

I. Introduction

GLOBAL optimization in operations research and computer science refers to the procedure of finding approximate solutions, which are considered the best possible solutions, to objective functions [1]. Ideally, the approximation is optimal up to a small constant error, for which the solution is considered to be satisfactory. In general, there can be solutions that are locally optimal, but not globally optimal; this situation appears more frequently when the dimension of the problem is high and when the function has many local optima [2]. Consequently, global optimization problems are typically quite difficult to be solved exactly, particularly in the context of nonlinear problems or combinatorial problems. Global optimization problems fall within the broader class of nonlinear programming. It should be noted that approximation algorithms are increasingly being used for problems where exact polynomial algorithms are known but are computationally expensive due to the dimensionality of these problems.

This paper focuses on the general global optimization problems in the continuous domain, having a nonlinear objective function that is either unconstrained or that has simple bound constraints. A variety of strategies, such as deterministic methods and probabilistic/heuristic methods, have been proposed to solve these problems. In deterministic methods, a clear relation exists between the characteristics of the possible solutions for a given problem. Probabilistic/heuristic methods solve optimization problems approximately in the case that the aforementioned relationship is not obvious, or it is complicated, or the dimensionality of the search space is very high [3].

In the last three decades, a significant research effort was focused on the development of effective and efficient stochastic methods that could reach the nearest global optimal solution. In this class of methods, evolutionary computation (EC) is one of the favorite methodologies, using techniques that exploit a set of potential solutions (called a population) to detect the optimal solution through cooperation and competition among the individuals of the population [4]. These techniques often find the optima for difficult optimization problems faster than traditional adaptive stochastic search algorithms. The most frequently used population-based EC methods include evolutionary strategies [4]–[6], genetic algorithms (GAs) [7]–[9], evolutionary programming (EP) [10], clustering methods [11], ant colony optimization (ACO/API) [12]–[14], and particle swarm optimization (PSO) [15], [16].

One of the issues that probabilistic optimization algorithms might face in solving global, highly nonconvex optimization problems is premature convergence. One of the causes of premature convergence of evolutionary-based algorithms is the lack of diversity. In nature, the diversity is ensured by the variety and abundance of organisms at a given place and time. The same principle (different type of solutions at one moment in the iterative search process) is used in computational intelligence for optimization algorithms [17].

Another issue of probabilistic approaches in optimization is related to their lack of advanced search capability around the global solution. Several studies have shown that incorporating some knowledge about the search space can improve the search capability of evolutionary algorithms (EAs) significantly [18]. In particular, the hybridization of EAs with local searches has proven to be very promising [19], [20].

A. ACO for Continuous Search Domains

This paper adopts the hybridization of a special class of ACO called API [14] and a real-coded genetic algorithm (RCGA) similar to the approach of the evolutionary strategy with float representation. Compared to its ACO surrogates, which were mainly applied to discrete optimization problems, API was particularly designed for continuous optimization problems. The proposed ant-based algorithm differs in terms of search

Manuscript received August 9, 2010; revised March 14, 2011 and July 5, 2011; accepted July 24, 2011. Date of current version December 7, 2011. This paper was recommended by Associate Editor S. Hu.

The authors are with the KIOS Research Center for Intelligent Systems and Networks and the Department of Electrical and Computer Engineering, University of Cyprus, 1678 Nicosia, Cyprus (e-mail: ciornei.irina@ucy.ac.cy; elias@ucy.ac.cy).

Digital Object Identifier 10.1109/TSMCB.2011.2164245

strategy from the basic ACO in that the ants have to navigate by using their memory of the visual landmarks encountered along with the familiar routes, instead of using pheromones. Further, API aims at maximizing the prey instead of minimizing the path. API has also a good "downhill" (gradient descending) search behavior. One of its drawbacks though, is that it may end up quickly in a local minimum due to a constant movement of the nest only in the best position found by the ants (the search agents).

ACO started to be analyzed from the continuous optimization perspective in the analytical way [21]–[23] only recently, in spite of the fact that the first proposal to adapt ACO for continuous optimization dates back to 1995 [24]. Bilchev and Parmee's proposal, entitled continuous ant colony optimization, initializes a nest at a given point of the search space. Then, it generates random vectors corresponding to the directions that will be followed by each ant in its attempt to improve the solution. If an ant is successful in such a pursuit, the direction vector chosen is updated. The continuous pheromone model proposed later on [23] is a more complex approach which uses a Gaussian probabilistic approach for the pheromone update. This model consists of a Gaussian pdf centered on the best solution found so far in the search process (up to the current iteration). The variance vector of this pdf starts with a value three times greater than the range of each variable of the problem (e.g., each ant takes a proportion from the solution space to explore). Then, this variance value is modified according to a weighted average of the distance between each individual (ant) in the population and the best solution found so far. The variance vector depends only on the number of ants. The main drawback of this model is the fact that "it only investigates one promising region of the problem at a time" [21], and it may therefore suffer from premature convergence. In [22], an ACO algorithm for the continuous domain is proposed; this algorithm can avoid premature convergence, and therefore local optima trapping. The proposed algorithm uses an archive that holds a predefined number of the best solutions found so far. Each solution corresponds to the center of a different Gaussian pdf.

B. Hybrid ACO-GA

In recent years, there were proposals to hybridize ACO and GA in a number of optimization applications [25]–[28]. The approaches adopted in [25] and [27] are similar and consist of using both ACO and GA algorithms to search in parallel for a better solution. Both of them refer to the combinatorial optimization model of ACO. Migration of solutions from one algorithm to the other occurs whenever any of them finds an improved potential solution after an iteration. Thus, a percentage of the best solutions of ACO (say K%) are added to the GA population pool which will follow the breeding process proportional to their fitness. Then, another percentage of the best individuals in the GA (L% of the GA population) are used to add "fitness-proportional pheromone" in the ACO search process and a new percentage of the worst fitted individuals of GA (M%) are used to "evaporate a constant amount of pheromones" in the ACO search. When both algorithms find an improvement, or both algorithms do not improve after an iteration, then no migration takes place. The application of the hybrid ACO-GA to continuous optimization problems was adopted in [26] and [28].

This work proposes a novel hybrid stochastic algorithm which intends to solve global unconstrained continuous optimization problems of medium and large dimensions. The proposed algorithm is a hybridization of two classes of evolutionary optimization algorithms: a special class of ACO for continuous domain and a RCGA. The proposed algorithm, entitled GAAPI, keeps the "downhill" search ability of API, while avoiding local minimum trapping by using the diversity in the solution given by the RCGA. The main advantages of the optimization tool proposed are its reduced computational time, and the robustness and consistency of high quality approximate solutions.

The paper is organized as follows: Section II provides background information and necessary concepts for the type of optimization problems tackled in this paper. Section III provides a description of the proposed algorithm and its basic steps. Sections IV and V provide an analysis of the algorithm's performance. More specifically, Section IV focuses on parameter settings and benchmark functions used for comparison analysis with other heuristic methodologies for continuous global optimization. Section V provides an empirical proof of convergence behavior of the proposed algorithm using 20 test functions typically used in the literature for the analysis of heuristic optimization techniques. Section VI is allocated to discussion and conclusions.

II. Background Information and Concepts

The algorithm proposed in this paper addresses optimization problems in the continuous domain of the form

$$\min_{x \in H} f(x) \tag{1}$$

where $x \in R^n$, and $H = \{x \in R^n | l_i \leq x_i \leq u_i\} \subseteq R^n$, $i = 1, \ldots, n$, with l_i and u_i being the lower and upper bounds of x_i, respectively. We are particularly interested in unconstrained optimization problems, and thus it is assumed that the set H is wide enough such that

$$H \supseteq H_o = \{x \in R^n | f(x) \leq c\} \tag{2}$$

for a sufficiently large real number c.

Further, it is assumed that the function $f(x)$ is continuous on H, and that $H \cap H_0$ is a nonempty and compact set for a real number c. An interpretation of the H_0 and c for the algorithm proposed in this paper is given in following section of the paper.

Suppose that $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is a globally optimal solution and $\varepsilon > 0$ is a sufficiently small number. If a candidate solution $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$ satisfies

$$|\tilde{x}_i - x_i^*| \leq \varepsilon, \quad i = 1, \ldots, n \tag{3}$$

or

$$|f(\tilde{x}) - f(x^*)| \leq \varepsilon, \quad i = 1, \ldots, n \tag{4}$$

then, $\tilde{x}$ is called an ε-optimal solution of the problem defined in (1).

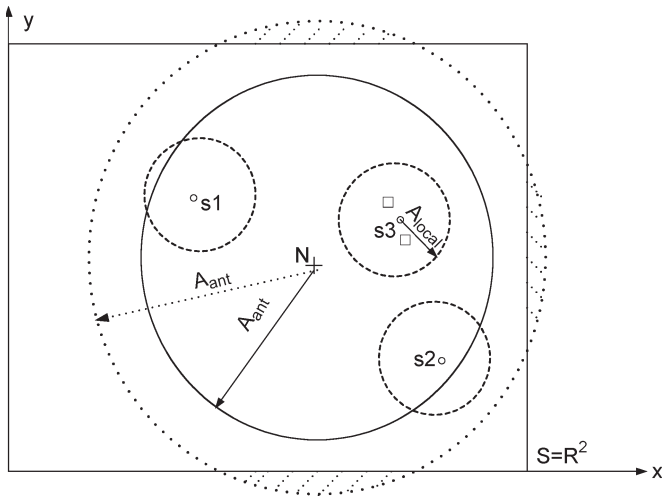

Fig. 1. Search space division according to API strategy. $S = R^2$ denotes a bi-dimensional solution space; s_1, s_2, s_3 are sites randomly generated around the $nest$, their maximum distance from the $nest$ being given by A_{ant}. The small squares denote local exploration of the site (points situated at a maximum distance of A_{site} from the site center s) [14].

In the case of the algorithm proposed in this paper, to find ε-optimal solutions, the feasible solution space $[l, u]$ is divided into smaller solution spaces with different amplitudes (defined as a percentage of the search space) from the initial domain, where overlapping is allowed. Fig. 1 shows how the initial solution space is divided into smaller search spaces. The example in Fig. 1 is given for a 2-D search space. This approach is the one adopted by Monmarche in his thesis when proposing the API algorithm [29]. The approach is quite similar to the adaptation of ACO for continuous domains proposed in [24].

The nest N initially takes a random position in the feasible search space $[l, u]$, where $l = (l_1, l_2, \ldots, l_n)$, and $u = (u_1, u_2, \ldots, u_n)$ are the lower and upper bound vectors for each dimension, respectively, delimitating the feasible solution space in $\boldsymbol{R}_n$ (n is the dimension of the problem). Therefore, $N = (N_1, N_2, \ldots, N_n)$ is the initial position of the nest in the feasible solution space.

The amplitudes for search space division change dynamically. The formula used to determine the search amplitude of each agent (ant) is given by

$$A_{ant} = \left(1 - \frac{k}{N_{ants}}\right) G_{ant_i} \tag{5}$$

where A_{ant} is the radius from the *nest*, delimitating the solution space ant i can cover; k is the current index (iteration of the search loop) of ant i, N_{ants} is the total number of search agents, and G_{ant_i} is the age of the ant and it is a parameter that increases as ant i performs its tasks with time, and is computed by

$$G_{ant_i} = \frac{T_i}{T_{ant_i}}. \tag{6}$$

This parameter was inspired from the real behavior of pachycondyla apicalis ants described in [30]. T_i is the current number of iterations after the movement of the ant ant_i, and T_{ant_i} is the maximum number of iterations between two movements of the ant ant_i.

III. GAAPI Algorithm

This section of the paper describes the proposed GAAPI algorithm. First, an introduction to the two main components of the GAAPI algorithm is provided. The first component, the API algorithm, is the core of the proposed method; the second component is the RCGA with emphasis on GA operators modified such as to maintain an improved balance between exploration and exploitation in the search procedure. In essence, the proposed GAAPI algorithm is mainly the API algorithm, but the ants selected for recruitment, instead of exchanging information from their memorized sites, obtain new hunting sites through a RCGA that uses a population that includes as individuals values from old forgotten sites.

A. API Algorithm

The API algorithm was inspired by the behavior of a type of ants (pachycondyla apicalis ants) which live in the Mexican tropical forest near the Guatemalan border. Colonies of these ants comprise around 20 to 100 ants. The foraging strategy of the pachycondyla apicalis ants can be characterized by the following description. First, these ants create their hunting sites which are distributed relatively uniformly within a radius of approximately 10 m around their nest. In this way, using a small mosaic of areas, the ants cover a rather large region around the nest. Second, the ants intensify their search around some selected sites for prey. Pachycondyla apicalis ants use a recruitment mechanism called tandem running, in which pairs of ants, one leading and one following, move toward a resource. In this foraging process, these ants use their memory of the visual landmark (map of the terrain encountered in their previous search) rather than pheromone trails (chemical signals) encountered in other ant species. After capturing their prey, the ants will move to a new nest via the tandem running recruitment mechanism to begin a new cycle of foraging. Based on the natural behavior of pachycondyla apicalis ants described in [30], Monmarché *et al.* proposed an API algorithm (short for apicalis) for the solution of optimization problems [29]. Despite the good performance of the algorithm, further research shows that API has poor use of the memory that generally characterizes ant colony systems [31]. Monmarché suggested that a hybridization of API with simulated annealing could improve its performance [29].

A short step-by-step description of API, the main core of the GAAPI algorithm proposed in this paper, is given below:

1. ***Initialization***: set the algorithm parameters
2. ***Generation of new nest*** (exploration)
3. ***Exploitation***
    3.1. ***Intensification search***:
        For each ant
            if the number of hunting sites in its memory is less than a predefined number
            then create a new site in its neighborhood and exploit the new site
            elseif the previous site exploitation was successful
            then exploit the same site again

else exploit a probabilistically selected site (among the sites in its memory)
end
end

3.2. ***Erase sites***: from the memory of ants erase all sites that have been explored unsuccessfully more than a predefined consecutive number of times

3.3. ***Information sharing***:
Choose two ants randomly and exchange information between them. The information exchanged is the best site in their memory at the current iteration

3.4. ***Nest movement***:
if the condition for nest movements is satisfied, go to step (4)
else, go to step (3.1)
end

4. ***Termination test***:
if the test is successful, STOP
else, empty the memory of all ants and go to step (2)
end.

B. RCGA Algorithm

The RCGA is inspired from the float representation of the evolutionary strategy approach. RCGAs work with real number representation; therefore, there is no other structure of the chromosomes, but floating vectors whose size is the number of variables of the optimization problem to be solved. This form of GA has the advantage of eliminating coding and decoding procedures needed in the binary representation of GA, thus using real-value object representation. For most applications of GAs in constrained optimization problems, the real coding is used to represent a solution to a given problem. This is one of the reasons that it has been adopted for hybridization with API in this work.

GAs start searching the solution space by initializing a population of random candidates for the solution. Every individual in the population undergoes genetic evolution through crossover and mutation. The selection procedure is conducted based on the fitness of each individual. By using elitist strategy, the best individual in each generation is ensured to be passed to the next generation. The elitist selection operator creates a new population by selecting individuals from the old populations, biased toward the best individuals. The chromosomes, which produce the best optimal fitness, are selected for the next generations. Crossover is the main genetic operator, which swaps chromosome parts between individuals. Crossover is not performed on every pair of individuals, its frequency being controlled by a crossover probability (P_c). This probability should have a large value for a higher chance of creating offspring with genome appropriate to the parents. The blend crossover (denoted as BLX-alpha) is the operator adopted in this work, due to its interesting property: the location of the child solution depends on the difference in the parent solutions. In other words, if the difference between the parent solutions is small, the difference between the child and parent solutions is also small. This property is essential for a search algorithm to exhibit self-adaptation. Thus, the BLX-alpha proceeds by blending two float vectors (x^t, y^t) using a parameter $\alpha = [0, 1]$, where t denotes the index of the generation. The resulting children (x^{t+1}, y^{t+1}) are equal to $x_i^{t+1} = (1 - \alpha_i)x_i^t + \alpha_i y_i^t$ and $y_i^{t+1} = \alpha_i x_i^t + (1 - \alpha_i)y_i^t$, respectively.

The last operator is mutation and its action is to change a random part of the string representing the individual. Mutation probability should be quite low, relative to the crossover probability, so that only a few elements in the solution vector undergo the mutation process. If the probability of mutation is high, then the offspring may lose too many of the characteristics of the parents and may lead to divergence in the solution. Uniform mutation was adopted in this work. The algorithm is repeated for several generations until one of the individuals of the population converges to an optimal value or the required number of generations is reached.

A short step-by-step mathematical description of the RCGA is given below:

1. ***Initialize the population***:
$s_i = U(l_k, u_k)^k$, where s^i is the individual i of the population, with $i = 1, \ldots, m$, and U is a uniform distribution in the range $[l_k, u_k]$, bounded in each k dimension, and m is the number of potential parents (or population size).
2. ***Determine the fitness score of the population and select parents according to their fitness score (the individuals with the highest fitness are selected as parents)***:
$\Theta(s_i) = G(f(s_i))$, where $\Theta(s_i)$ is the fitness score of the individual s_i, G denotes the fitness score function, and f is the real fitness or optimization function. The fitness score function adopted in this paper is the inverse of the fitness function to be optimized. In case of minimization problems, the individual is considered to be the most fitted if it has the smallest value of the optimization function. In case of maximization problems, the fitness score is given by the fitness function.
3. ***Variance assignment***:
3.1. Apply blend ***crossover***, with probability P_c

$$s_i = s_{i+m}$$

3.2. Apply ***mutation*** operator, with probability P_m

$$s_{i+m,j} = S_{i,j} + N(0, \beta_j * \theta(s_i) + z_j), \quad j = 1, \ldots, k$$

where $S_{i,j}$ is the element j of the individual i, $N(\mu, \sigma^2)$ is the Gaussian random variation with mean μ and variance σ, β_j is a constant of proportionality to scale $\Theta(s_i)$, and z_j is the offset that guarantees the minimum amount of variance.
4. ***Determine the fitness score of each variance***:
Each variance s_{i+m} is assigned a fitness score $\Theta(s_{i+m}) = G(f(s_{i+m}))$.
5. ***Rank the solution*** in descending order of $\Theta(s_i)$
6. ***Repeat***: Go to step 3 until an acceptable solution has been found or the available execution time is exhausted.

The equation from item 3.1 above $(s_i = s_{i+m})$ shall be read as follows: after crossover, a new individual is formed (s_{i+m}), which is added at the end of the current population (whose dimension is m). If a randomly generated number is higher than

the probability of crossover (P_c) of the ith individual in the current population, then the newly formed individual (s_{i+m}) replaces the ith individual in the next generation. The same applies to item 3.2, in which the mutation operator is applied with a probability of mutation P_m to each gene j of each individual i.

C. GAAPI Algorithm

To eliminate the shortcomings and the insufficient robustness of the global search ability of the API algorithm, a GAAPI algorithm that incorporates some favorable features of GA and API algorithms is proposed in this paper. The idea in GAAPI is to keep the algorithm focused toward continuously improving in the solution, while avoiding trapping in local optima. Therefore, the API algorithm was intended to be the core of the GAAPI, keeping the search tracked toward improvement in the solution, while RCGA was intended to provide the escape mechanism from local optima when API is trapped. Thus, when API is at the search level of sites (the lowest search level) and continuously improves the solution, RCGA is in a passive mode. In this passive mode, the population of RCGA is formed by all the best solutions generated by API at the ant level only (there are no sites to be forgotten). When API is slow in improving the solution (there are sites to be forgotten due to failure in improving the solution), RGCA switches to an active role. This time, its population uses the information of forgotten sites as well (the population is more heterogeneous than in the former case), and thus the solution generated by the RCGA has more chances to be far from the local optimum in which the API was trapped.

The key modifications in API to form the new GAAPI algorithm are summarized below.

1) ***Generation of New Nest***: After initialization, only the best solution found since the last nest move has the opportunity to be selected as a new nest to start the next iteration. The "hill climb" property is not very strong in this case, so the trapping in local minima is avoided.
2) ***Exploitation with API***: Initially, each ant checks its memory. If the number of hunting sites in its memory is less than a predefined number, it will generate a new site in the small neighborhood of the center of its current hunting site, save it to its memory, and use it as the next hunting site. Otherwise, one of the sites in its memory is selected as the hunting site. The ant then performs a local search around the neighborhood of this hunting site. If this local exploitation is successful, the ant will repeat its exploration around the site until an unsuccessful search occurs; otherwise (if the previous exploration was unsuccessful), the ant will select an alternative site among its memorized sites. This process will be repeated until a termination criterion is reached. The termination criterion used in this phase is that the procedure will stop automatically once the number of successive unsuccessful explorations reaches a predefined value, or there is no improvement after a number of iterations. A schematic representation of the search mechanism of API is given in Fig. 2.

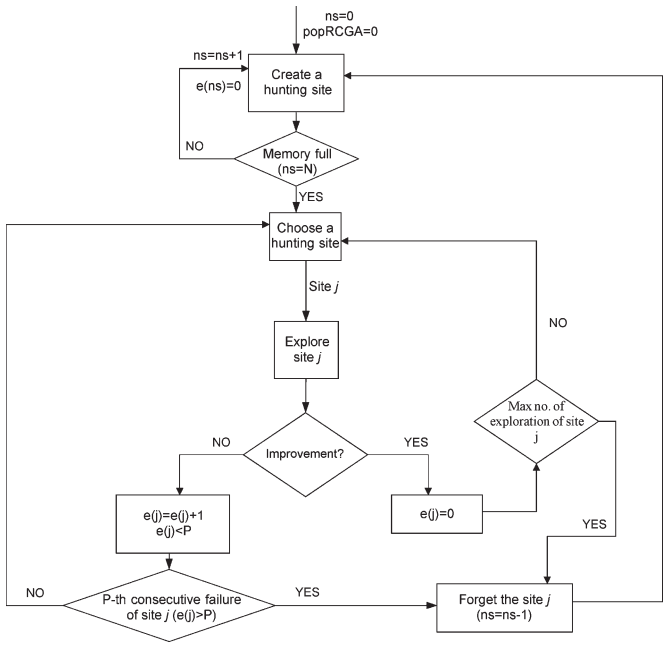

Fig. 2. API search mechanism as used in the GAAPI algorithm. ns represents the counter for the number of sites memorized by each ant; $e(ns)$ is the counter for consecutive failure in site search; Ns is the total number of sites one ant can memorize; $popRCGA$ is the counter for the number of individuals added into the population of the RCGA algorithm; P is the predefined number of allowed consecutive search failures in one site before it is deleted from the memory of the ant.

2.1. ***Information sharing with RCGA***: To keep diversity in the solution space, information sharing is performed using a simple RCGA method. A random site is chosen in the memory of a randomly chosen ant, and it is replaced by the new RCGA solution. This can be seen as a form of communication. The RCGA procedure involves a population formed by the currently best hunting sites in the memory of all ants as well as the forgotten (erased) sites. The best solution obtained after one set of GA operations (selection, crossover, mutation) replaces the chosen site in the memory of the selected ants. This technique is applied before moving the nest to the best position so far. The RCGA contains the forgotten sites to keep diversity in the population.

The RCGA operators are set as follows: (1) *Blend crossover* operator (BLX-α) [7] with a probability of 0.3, and a value of α set to 0.366 [30]; a *uniform mutation* with a mutation probability set to 0.35; ***Elitism***: the two best individuals are retained with no modifications in the population of the next generation, such that the strongest genes up to this point are retained. Fig. 3 shows the GAAPI algorithm in the form of a flowchart, demonstrating the key steps of the process.

For the algorithm proposed here, H_0 and c defined in (2) have the following interpretation: H_0 is the solution space of all better solutions than the current nest position $(nest)$ found by the ant colony during the current search, and $c = f(nest)$ is the evaluation of the objective function in the current position of the nest.

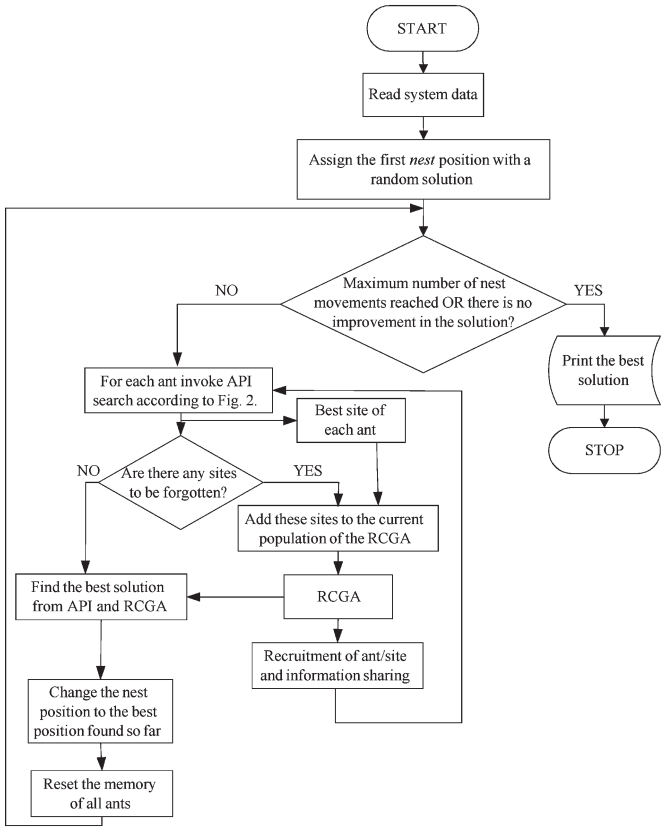

Fig. 3. GAAPI flowchart.

Under the assumptions made in the second section of the paper and according to the lemmas defined in [32], the probability to end up in an ε-optimal solution tends to 1 as the number of iterations (nest movements) tends to infinity.

GAAPI has a well-established balance between exploration (with API and RCGA) and exploitation (API). API keeps the algorithm focused toward the global optimum, moving the nest position (the point where exploitation starts) only in the best solution found so far, while RCGA helps the ants to use useful information of less explored regions (forgotten sites) The strong influence of API with its "down-hill" (gradient descending) behavior may increase the speed of convergence toward the global when compared to other powerful global search techniques such as PSO, EAs, or GAs, where the exploration behavior may play a strong role.

IV. Parameter Settings and Test Functions

In this section of the paper, the performance of the proposed algorithm is investigated, considering a set of 20 benchmark test functions. These test functions are widely used in the scientific literature to test optimization algorithms. Note that most of the test functions have many local minima so that they are challenging enough for performance evaluation.

A. Test Functions

Twenty widely used functions have been chosen from [20], [29], [32] as test functions, and the proposed algorithm in this paper was tested for all of them. These functions are shown in the Appendix of the paper. A few descriptive characteristics of a class of six very popular test functions (out of the 20 functions) are provided in Table I in the Appendix. The basic parameters of all 20 test functions are listed in Table II, including search space limits, their dimension, and their global minimum.

TABLE I
Characteristics of Six Benchmark Functions

Test function	Search space	Global minimum	Dimension (n)
F1	$[-500, 500]^n$	-12569.5	30
F2	$[-5.12, 5.12]^n$	0	30
F3	$[-32, 32]^n$	0	30
F4	$[-600, 600]^n$	0	30
F5	$[-50, 50]^n$	0	30
F6	$[-50, 50]^n$	0	30
F7	$[0, pi]^n$	-99.2784	100
F8	$[-pi, pi]^n$	0	100
F9	$[-5, 5]^n$	-78.3324	100
F10	$[-5, 10]^n$	0	100
F11	$[-100, 100]^n$	0	30
F12	$[-1.28, 1.28]^n$	0	30
F13	$[-10, 10]^n$	0	30
F14	$[-100, 100]^n$	0	30
F15	$[-100, 100]^n$	0	30
F16	$[-5, 5]^n$	-1.03163	2
F17	[-5, 10]x[0, 15]	0.398	2
F18	$[-2, 2]^n$	3	2
F19	$[-5, 5]^n$	0.000308	4
F20	$[0, 1]^n$	-3.32	6

TABLE II
Characteristics of the Test Functions

Function	MIN	MAX	MEAN	STD	CPU (s)
F1	-12569.5	-12568.3	-12569.4	0.2618	30.5867
F2	1.02E-06	0.028603	0.005046	0.0074	27.0671
F3	0.000163	0.187231	0.038933	0.0545	18.2644
F4	2.2E-05	1.027464	0.077839	0.2250	37.2827
F5	3.65E-10	1.72E-08	2.73E-06	1.1E-06	22.4786
F6	2.23E-07	0.33526	0.068774	0.1102	42.2726
F7	-39.7847	-24.5752	-37.4486	4.7103	24.2658
F8	1.4E-07	1.441302	0.149302	0.3299	401.7522
F9	-78.3323	-78.331	-78.3322	0.0003	37.9272
F10	4.18E-05	0.257584	0.040124	0.0707	35.5866
F11	6.67E-09	0.063028	0.010741	0.0176	35.6439
F12	1.28E-05	0.0132	0.0037	0.0030	17.0035
F13	0.001297	0.244238	0.055812	0.0546	37.5640
F14	0.000537	9.408785	2.292226	3.2218	39.5760
F15	0.000298	0.047323	0.01266	0.0134	30.7023
F16	6.94E-10	1.24E-05	1.36E-06	3.05E-06	23.8269
F17	10.22525	10.22532	10.22526	1.47E-05	27.8655
F18	3.002442	7.805611	3.703591	1.2472	27.2073
F19	0.051743	0.051869	0.051756	3.08E-05	27.9693
F20	-22.231	-22.231	-22.231	1.45E-05	29.0590

TABLE III
PERFORMANCE OF GAAPI OVER THE 20 TEST FUNCTIONS

Function	Algorithm used and CPU time (s)			
F1	HTGA	CPSO-H6	LEA	GAAPI
	689.30	658.70	656.30	30.59
F2	HTGA	CPSO-H6	LEA	GAAPI
	607.50	557.70	557.20	27.07
F3	ALEP	CPSO-H6	LEA	GAAPI
	359.30	326.80	326.10	18.26
F4	HTGA	CPSO-H6	LEA	GAAPI
	373.80	368.10	365.60	37.28
F5	HTGA	CPSO-H6	LEA	GAAPI
	378.60	369.80	368.50	22.48
F6	HTGA	CPSO-H6	LEA	GAAPI
	381.20	363.70	359.10	42.27
F7	HTGA	CPSO-H6	LEA	GAAPI
	765.20	719.60	660.80	24.27
F8	ALEP	CPSO-H6	LEA	GAAPI
	689.40	503.40	493.40	401.75
F9	ALEP	CPSO-H6	LEA	GAAPI
	782.70	685.80	612.30	37.93
F10	HPSO-TVAC	CPSO-H6	LEA	GAAPI
	594.50	501.10	443.80	35.59
F11	HTGA	CPSO-H6	LEA	GAAPI
	312.50	242.60	240.20	35.64
F12	HTGA	CPSO-H6	LEA	GAAPI
	318.40	243.70	242.40	17.00
F13	HTGA	CPSO-H6	LEA	GAAPI
	322.60	243.00	240.80	37.56
F14	HTGA	CPSO-H6	LEA	GAAPI
	328.40	244.60	241.30	39.58
F15	HTGA	CPSO-H6	LEA	GAAPI
	334.20	243.10	242.20	30.70
F16	HTGA	ALEP	LEA	GAAPI
	31.60	31.10	30.80	23.83
F17	HTGA	ALEP	LEA	GAAPI
	31.10	31.10	30.60	27.87
F18	HTGA	ALEP	LEA	GAAPI
	35.40	34.00	33.50	27.20
F19	HTGA	ALEP	LEA	GAAPI
	121.20	102.00	101.30	27.97
F20	CPSO-H6	ALEP	LEA	GAAPI
	67.70	66.60	66.20	29.06

For all 20 test functions, the results obtained by GAAPI are compared to other well-known evolutionary based optimization methods. The results are presented in Tables III–VIII. These tables are based on the comparison tables given in [32], which the authors found to be very well documented.

B. Parameter Values for GAAPI

The values of the parameters of GAAPI that have been used for the global optimization of the 20 test functions are given below.

TABLE IV
COMPARISON TO OTHER HEURISTIC METHODS WITH RESPECT TO CPU TIME

Function	Algorithm	M-num-fun	M-best	Std	Opt-F
F1	ALEP	150000	-11469.2	58.2	-12569.5
	FEP	900000	-12554.5	52.6	
	OGA/Q	302116	-12569.45	6.44E-4	
	HTGA	163468	-12569.46	0	
	EDA/L	52216	-12569.48	N/A	
	M-L	655895	-5461.826	275.15	
	LEA	287365	-12569.45	4.83E-4	
	ACAGA	N/A	-12569.48	4.12E-3	
	GAAPI	26510	-12569.5	5.77E-7	
F2	ALEP	150000	5.85	2.07	0
	FEP	500000	0.046	0.0012	
	CEP	250000	4.73	N/A	
	OGA/Q	224710	0	0	
	HTGA	16267	0	0	
	HPSO-TVAC	200000	0.044	0.19	
	CPSO-H6	200000	0.778	N/A	
	EDA/L	75014	0	N/A	
	M-L	305899	121.7575	7.75	
	LEA	223803	2.10E-8	3.3E-18	
	API	N/A	2.32	N/A	
	ACAGA	N/A	2.53E-6	3.62E-6	
	GAAPI	24714	1.02E-6	4.58E-9	
F3	ALEP	150000	0.019	0.001	0
	FEP	150000	0.018	0.021	
	CEP	250000	7.49E-4	N/A	
	OGA/Q	112421	4.4E-16	3.9E-17	
	HTGA	16632	0	0	
	CPSO-H6	200000	2.7E-12	N/A	
	EDA/L	106061	4.1E-15	N/A	
	M-L	121435	2.5993	0.094	
	LEA	105926	3.2E-16	3.0E-17	
	API	N/A	2.22E-3	N/A	
	ACAGA	N/A	4.69E-6	7.12E-5	
	GAAPI	19592	0.000163	9.1E-7	
F4	ALEP	150000	0.024	0.028	0
	FEP	200000	0.016	0.022	
	CEP	250000	2.52E-7	N/A	
	OGA/Q	134000	0	0	
	HTGA	20999	0	0	
	HPSO-TVAC	200000	0.01	0.001	
	CPSO-H6	200000	0.0524	N/A	
	EDA/L	79096	0	N/A	
	M-L	151281	0.11894	0.0104	
	LEA	130498	6.10E-6	2.5E-17	
	GAAPI	29647	2.2E-05	2.05E-8	
F5	ALEP	150000	6.0E-6	1.0E-6	0
	FEP	150000	9.2E-6	3.6E-6	
	OGA/Q	134556	6.01E-6	1.15E-6	
	HTGA	66457	1.0E-6	0	
	EDA/L	89925	3.6E-21	N/A	
	M-L	146209	0.2105	0.0360	
	LEA	132642	2.42E-6	2.27E-6	
	GAAPI	60297	3.65E-10	1.82E-11	

- The *population size* of RCGA is variable and depends on the current iteration and the number of unsuccessful sites memorized until the recruitment process. In the case of the initial iteration, the population has five individuals: the first and second best up to the first call of RCGA and three other individuals chosen randomly from all the sites of all

TABLE V
COMPARISON TO OTHER HEURISTIC METHODS FOR F1 TO F5

Function	Algorithm	M-num-fun	M-best	Std	Opt-F
F6	ALEP	150000	9.80E-05	1.20E-05	0
	FEP	150000	1.60E-04	7.30E-05	
	OGA/Q	134143	1.87E-04	2.62E-05	
	HTGA	59003	0.0001	0	
	EDA/L	114570	3.49E-21	N/A	
	M-L	147928	1.51E+00	2.25564	
	LEA	130213	1.73E-04	1.21E-04	
	GAAPI	26895	2.23E-07	3.06E-03	
F7	OGA/Q	302773	-92.83	0.02626	-99.3
	HTGA	265693	-92.80	0	
	EDA/L	169887	-94.3757	N/A	
	M-L	329087	-23.9754	0.62875	
	LEA	289863	-93.01	0.02314	
	GAAPI	21235	-39.7847	0.100521	
F8	OGA/Q	190031	4.67E-07	1.29E-07	0
	HTGA	186816	5.87E-05	8.33E-05	
	EDA/L	124417	3.29E-08	N/A	
	M-L	221547	25877.8	1739.75	
	LEA	189427	1.63E-06	6.527E-07	
	GAAPI	28778	1.40E-07	2.33E-02	
F9	OGA/Q	245930	-7.83E01	6.29E-03	-78.3
	HTGA	216535	-78.303	0	
	EDA/L	153116	-78.3107	NA	
	M-L	251199	-35.8099	0.89146	
	LEA	243895	-78310	6.13E-03	
	GAAPI	28701	-78.3323	1.38E-05	
F10	OGA/Q	167863	0.752	0.114	0
	HTGA	60737	0.7	0	
	HPSO-TVAC	200000	9.855	6.725	
	EDA/L	128140	4.32E-03	N/A	
	CPSO-H6	200000	0.194	N/A	
	M-L	137100	2935.93	134.8186	
	LEA	168910	0.5609	0.1078	
	GAAPI	29171	4.18E-05	5.03E-03	

ants. In the case of subsequent iterations, the population composition is like the first iteration only if no forgotten sites appear up to that point.

- ***Blend crossover*** operator (BLX-α) with a probability $P_c = 0.3$; the value of α was set to 0.366.
- *Uniform mutation* with a mutation probability $P_m = 0.35$.
- *The number of ants* in the API colony was set to 100.
- The *number of sites* each ant can search and memorize was set to 3.

The maximum number of explorations of the same site was set to 30. For a number of functions with many local minima very near to each other (F5, F7, F16, F17, and F20), the maximum number of explorations was set to 500. The number of consecutive unsuccessful visits at one site before being deleted from the memory of the ant was set to 5 (or 40 for the functions cited above).

V. EMPIRICAL PROOF OF CONVERGENCE: RESULTS AND ANALYSIS

The proposed algorithm was executed in 50 independent runs for each test function, to keep the same base of comparison. The algorithm was implemented in MATLAB 7.a on a Pentium IV personal computer with a 3.6 GHz processor. The following data were recorded: the minimum function value denoted by MIN, the maximum function value denoted by MAX, the average function value denoted by MEAN, the standard deviation denoted by STD, the average CPU time of 50 independent runs denoted by CPU, and the mean number of function evaluations denoted by M-num-fun. The last two analysis components give a fair indication about the effectiveness of the

TABLE VI
COMPARISON TO OTHER HEURISTIC METHODS FOR F6 TO F10

Function	Algorithm	M-num-fun	M-best	Std	Opt-F
F11	ALEP	150000	6.32E-04	7.60E-05	0
	FEP	150000	5.70E-04	1.30E-04	
	CEP	250000	3.09E-07	N/A	
	OGA/Q	112559	0	0	
	HTGA	20844	0	0	
	HPSO-TVAC	120000	0.01	N/A	
	M-L	162010	3.19123	0.29463	
	LEA	110674	4.73E-16	6.22E-17	
	API	N/A	6.65E-06	N/A	
	GAAPI	29199	6.67E-09	1.89E-04	
F12	CEP	250000	9.42	N/A	0
	OGA/Q	112652	6.30E-03	4.07E-04	
	HTGA	20065	0.001	0	
	M-L	124982	1.703986	0.52155	
	LEA	111093	5.14E-03	4.43E-04	
	GAAPI	4149	1.28E-05	7.21E-07	
F13	FEP	200000	0.0081	7.70E-04	0
	CEP	250000	1.99E-03	N/A	
	OGA/Q	112612	0	0	
	HTGA	14285	0	0	
	M-L	120176	9.7416	0.463769	
	LEA	110031	4.25E-19	4.24E-19	
	ACAGA	N/A	2.58E-05	4.17E-05	
	GAAPI	30714	0.001297	2.01E-02	
F14	ALEP	150000	0.04185	5.97E-02	0
	FEP	500000	0.016	0.014	
	CEP	250000	0.612	N/A	
	OGA/Q	112576	0	0	
	HTGA	26469	0	0	
	CPSO-H6	200000	2.63E-66	N/A	
	M-L	155783	2.21994	0.50449	
	LEA	110604	6.78E-18	5.43E-18	
	ACAGA	NA	2.26E-07	1.75E-6	
	GAAPI	31792	0.000537	3.1932	
F15	FEP	500000	0.3	0.5	0
	CEP	250000	0.323	N/A	
	OGA/Q	112893	0	0	
	HTGA	21261	0	0	
	M-L	125439	0.55755	4.00E-02	
	LEA	111105	2.68E-16	6.26E-17	
	GAAPI	31040	0.000298	4.01E-03	

TABLE VII
COMPARISON TO OTHER HEURISTIC METHODS FOR F11 TO F15

Function	Algorithm	M-num-fun	M-best	Std	Opt-F
F16	ALEP	3000	-1.031	0	-1.031
	FEP	10000	-1.03	4.90E-07	
	M-L	13592	-1.02662	5.27E-03	
	LEA	10823	-1.03108	3.36E-07	
	GAAPI	27241	6.94E-10	1.40E-07	
F17	FEP	10000	0.398	1.50E-07	0.398
	M-L	12703	0.403297	8.83E-03	
	LEA	10538	0.398	2.65E-05	
	GAAPI	29625	10.22525	5.43E-07	
F18	ALEP	3000	3	0	3
	FEP	10000	3.02	0.11	
	M-L	16325	3.048855	0.0603749	
	LEA	11721	3.00003	6.25E-05	
	GAAPI	29625	3.002442	2.85E-05	
F19	FEP	400000	5.00E-04	3.20E-04	3.08E-4
	M-L	186768	1.34E-03	2.98E-04	
	LEA	55714	3.51E-04	7.36E-05	
	GAAPI	28654	0.051743	8.45E-07	
F20	FEP	20000	-3.27	0.059	-3.32
	M-L	92516	-3.12696	0.067397	
	LEA	28428	-3.301	7.83E-03	
	GAAPI	29302	-22.231	3.87E-07	

TABLE VIII
COMPARISON TO OTHER HEURISTIC METHODS FOR F16 TO F20

Function no.	Function name	Description
F2	Rastrigin	A highly multimodal function. The location of the deep local minima is regularly distributed. The global minimum is in $x^* = 0$ and $f(x^*) = 0$.
F3	Ackley	A multimodal function with deep local minima. The global minimum of this function is $x^* = 0$ and $f(x^*) = 0$. The variables of this function are independent.
F4	Grienwangk	A multi-modal function, having the global minimum in $x^* = 0$ and $f(x^*) = 0$. There is significant interaction between its variables due to the product term.
F10	Rosenbrock	A unimodal function, that has its global minimum at $x^* = (1, 1, \ldots, 1)$ and $f(x^*) = 0$. This function has many interactions between some of its variables.
F11	Spherical	A very simple unimodal function, that has its global minimum at $x^* = 0$ and $f(x^*) = 0$. This function has no interaction between its variables.
F14	Quadratic	It is a variation of the spherical function but with many interactions between its variables. The global minimum is located at $x^*=0$ and $f(x^*) = 0$.

algorithm in real problems. The aforementioned parameters are generally accepted indicators of performance when referring to heuristic global optimization algorithms. Note that CPU time, together with the PC platform on which the algorithm was executed, is only provided for comparison reasons to other works which used this indicator. However, this parameter is subject to hardware platform where the algorithm is run and may not be the best choice of comparison of computational performance. The use of M-num-fun is emphasized in this paper instead.

Table III gives the quantitative performance results for the 20 test functions. In most of the benchmark functions, GAAPI proved its consistency, having the lowest standard deviation among the other methods and the lowest mean number of function evaluations and CPU time. The mean number of function evaluations (M-num-fun) is the average of the total number of function evaluations during a predefined number of independent runs of the algorithm. In other words, if we denote with nF_i the number of function evaluations in the ith independent run of the proposed algorithm and we have a total of M runs which we take into account in our evaluation process, then the mean number of function evaluations is

$$\mathrm{M-num-fun} = \frac{\sum nF_i}{M}. \tag{7}$$

In most of the cases, the number of function evaluations to reach a solution very near to the global solution is 10 to 50 times less than the other methods used for comparison. For seven of the most popular and difficult functions, GAAPI obtained the best global solution fast and accurately (F1, F5, F8–F12).

GAAPI responds very well, particularly for complex functions with higher dimensionality ($N = 100$ or 30, such as in F1–F7, F9–F15, and F18). However, the algorithm did not perform satisfactorily for test functions F16 (Fig. 4), F17 (Fig. 5), F19, and F20, which have flat minima (many local minima at the same level). This may be due to the termination criterion to stop when no improvement occurs after a number of consecutive nest movements. For the analysis of this paper the GAAPI algorithm was executed for all 20 functions using the same termination criterion: the algorithm stops if no improvement occurs after 20 consecutive nest movements.

Table IV provides a comparison of the computational time required for GAAPI and other heuristic methods for determining the global optimal solution. Results on other methods are obtained from [32]. It is shown that GAAPI is faster compared to the other methods; in some cases, it is considerably faster. As the computational effort is very important, particularly to actual problems that need to be solved in real time, GAAPI may be considered as a useful optimization tool based on the computational time required determining the global optimum.

The initials of the algorithms referenced in this paper are presented in Table IX of the Appendix. A brief description of some of these algorithms is presented in [32].

Tables V–VIII show a comparison between the performance of GAAPI and the performance of other heuristic algorithms, for all 20 test functions used for the present analysis. Each table compares the performance of each algorithm (if the results are available, and only for the functions tested by the authors) and provides the mean number of function evaluations (M-num-fun), the best value determined by each algorithm (M-best), and the standard deviation for 50 independent runs of each algorithm. Further, the optimal value of each test function is provided. It should be noted that in the literature selected for comparison for the purposes of this work, the same number of function evaluations for each algorithm was not available. Thus, the comparison below gives this measure only to sustain a quasi-comparison on the speed of convergence of different

TABLE IX
NOTATIONS OF THE ALGORITHMS USED FOR COMPARISON

Notation	Description
ALEP	Evolutionary programming with adaptive Levy mutation
FEP	Fast evolutionary programming Cauchy mutation
OGA/Q	Orthogonal genetic algorithm with quantization
HTGA	Hybrid Taguchi – genetic algorithm
EDA/L	Hybrid estimation of distribution algorithm
M-L	Modified mean-level-set method proposed in [32]
LEA	Level-set evolution and Latin squares algorithm
CEP	Conventional evolutionary programming
HPSO-TVAC	Hierarchical particle swarm optimization with time-varying acceleration coefficients
CPSO-H6	Hybrid cooperative particle swarm optimization, API – special class of continuous domain ant colony optimization search based on the Monmarche approach [14]
ACAGA	Hybrid algorithm combining ant colony algorithm with genetic algorithm for continuous domain optimization problems [28]

heuristic algorithms toward a near global solution denoted by the authors as the best-mean solution over a number of independent runs.

For the first five functions GAAPI found the near global solution in much less computational time and/or mean number of function evaluations (up to 20 times less) for all of the functions under analysis in this table. Also, GAAPI found the best solution among all algorithms for two of the functions (F1 and F5), while for the other three functions, GAAPI practically found the global optimum (the error was less than 10^{-4}). It should be noted that the values for the API and ACAGA algorithms given in Table V and VII were obtained from [29] and [28], respectively.

For test functions F6 to F10, in three of the cases, GAAPI had the best performance in terms of the global optimum solution, standard deviation and computational effort (F8, F9, and F10); for F6, the best solution found by GAAPI is the second best among all algorithms and very near to the global optimum solution. However, for F7, GAAPI did not succeed in finding the global optimum solution.

For the next group of functions (F11 to F15), for the first two functions, GAAPI obtained the best solution reported so far; for the next three functions (F13 to F15), GAAPI obtained a near global optimum solution. For F13, GAAPI obtained better solutions than the FEP, CEP and M-L algorithms, and better CPU time/ mean number of function evaluations than all the other algorithms. However, OGA/G, HTGA, ACAGA, and LEA had a better minimum solution. For F14, GAAPI outperformed ALEP, FEP, CEP, and M-L, but OGA/G, HTGA, CPSO-H6, ACAGA, and LEA performed better than GAAPI. For F15, LEA, OGA/Q, HTGA, and LEA outperformed GAAPI in terms of the best solution found so far; however, the GAAPI solution was near the global optimum in faster computational time.

For the last group of five test functions, GAAPI obtained a good solution for F18. However, its performance for the other four test functions was not satisfactory.

VI. CONCLUSION

In this paper, a new algorithm, called GAAPI, was proposed to solve global unconstrained continuous optimization problems. This algorithm is appropriate for optimization problems whose decision variables take values from the real-number domain. The hybrid meta-heuristic algorithm proposed in this paper, was created by combining some unique characteristics of two other robust meta-heuristic algorithms: RCGA and API.

It was proven that in most of the cases presented in this paper (15 out of 20 benchmark functions), GAAPI provided satisfactory or optimum solutions, with very little computational effort. The algorithm is recommended for large, complex problems with a dimensionality greater than 30. For seven benchmark functions, GAAPI gave the best solution reported so far in the literature, with less number of function evaluations (10 to 50 times less than other powerful methods). The best solution was found for complex functions with high dimensionality ($n = 30$ or $n = 100$) (seven test functions). For eight other test functions with high dimensionality ($n = 30$), GAAPI gave near global solutions with much less computational effort. However, for a small class of functions (five benchmark functions), having mainly small dimensionality ($n = 2$, $n = 4$ or $n = 6$), GAAPI failed to find the global optimum solution. The main reason for this failure is the flatness of the objective function around the global minimum.

There are at least two main reasons why GAAPI performs better than other powerful heuristic techniques. First, the balance in exploration and exploitation given by the two chosen algorithms API and GA is one of the reasons. API has a strong influence targeting the search toward a continuously improved solution (the nest is moved only in the best solution found at each iteration by its ants), while GA has an active role in the solution search, only when API reduces its speed of convergence (the solution does not improve much from one iteration to another, or there are many failures in exploiting different sites). This balance in exploration and exploitation increases the chances of a faster convergence toward the global optimum, while other methods such as PSO, EAs, or GAs have a strong exploration component. The second reason is the choice of crossover and mutation functions in RCGA. These influence the activeness or passiveness of GA in the GAAPI search. A different crossover (for example an arithmetic real-coded crossover) would maintain GA active at each nest movement, which may lead to solution divergence. The same may happen if the mutation probability is higher than the crossover probability.

Other hybridization techniques of API with variances of EAs may further improve the quality of the solution in difficult global optimization problems, but a difficulty in implementation could appear due to the complicated forms of the operators to be used. There may be value in comparing analytically the search behavior of GAAPI and other search models for ACO-GA hybridization techniques or in the association of API with other EAs used for some applications in continuous global optimization. There may also be value in concentrating on comparisons of GAAPI to other hybridization schemes which relate to GA and local search mechanisms. This study focused mainly on continuous domain optimization problems, so further work can be addressed to see the applicability of the proposed algorithm to discrete as well as constrained optimization problems.

APPENDIX

The functions used for testing the proposed algorithm are provided below. These are taken from [28], [29], and [32].

$$F_1 = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right)$$

$$F_2 = \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos(2\pi i) + 10\right)$$

$$F_3 = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi i)\right) + 20 + \exp(1)$$

$$F_4 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$F_5 = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$$

where

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

and

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_x < -a \end{cases}$$

$$F_6 = \frac{1}{10}\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \times \left[1 + \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2 \times \left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$$

$$F_7 = -\sum_{i=1}^{n}\sin(x_i)\cdot\sin^{20}\left(\frac{ix^2}{\pi}\right)$$

$$F_8 = \sum_{i=1}^{n}\left(\sum_{j=1}^{n}(\chi_{ij}\cdot\sin\omega_j + \psi_{ij}\cdot\cos\omega_j) - \sum_{j=1}^{n}(\chi_{ij}\cdot\sin x_j + \psi_{ij}\cdot\cos x_j)\right)^2$$

where, χ_{ij} and ψ_{ij} are random numbers in $[-100, 100]$, and ω_i is a random number in $[-\pi, \pi]$.

$$F_9 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^4 - 16x_i^2 + 5x_i\right)$$

$$F_{10} = \sum_{i=1}^{n}\left[100\left(x_i^2 + x_{i+1}\right)^2 + (x_i - 1)^2\right]$$

$$F_{11} = \sum_{i=1}^{n} x_i^2$$

$$F_{12} = \sum_{i=1}^{n} -ix_i^4 + rand[0, 1)$$

$$F_{13} = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i|$$

$$F_{14} = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)$$

$$F_{15} = \max\{|x_i|, i = 1, 2, \ldots, n\}$$

$$F_{16} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

$$F_{17} = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

$$F_{18} = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \times \left[30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 42x_2 - 36x_1x_2 + 27x_2^2\right)\right]$$

$$F_{19} = \sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2 + b_ix_2\right)}{b_i^2 + b_ix_3 + x_4}\right]$$

where

$$[a_1, \ldots, a_11] = [0.1957\ 0.1947\ 0.1735\ 0.16\ 0.0844\ 0.0627\ 0.0456\ 0.0342\ 0.0323\ 0.0235\ 0.0246];$$

$$[b_1, \ldots, b_{11}] = [4\ 2\ 1\ 0.5\ 1/4\ 1/6\ 1/8\ 1/10\ 1/12\ 1/14\ 1/16]$$

$$F_{20} = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$$

where

$$[c_1, \ldots, c_4] = [1\ 1.2\ 3\ 3.2];$$

$$[a_{ij}]_{4\times 6} = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$

$$[p_{ij}]_{4\times 6} = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1415 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}$$

REFERENCES

[1] Y.-W. Leung and Y. Wang, "An orthogonal genetic algorithm with quantization for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 5, no. 1, pp. 41–53, Feb. 2001.

[2] T. Weise, Global Optimization Algorithms: Theory and Applications2009.

[3] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*. New York: Springer-Verlag, 2004.

[4] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies: A comprehensive introduction," *Natural Comput.: Int. J.*, vol. 1, no. 1, pp. 3–52, May 2002.

[5] P. Guturu and R. Dantu, "An impatient evolutionary algorithm with probabilistic Tabu search for unified solution of some NP-hard problems in graph and set theory via clique finding," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 3, pp. 645–666, Jun. 2008.

[6] G. Y. Yang, Z. Y. Dong, and K. P. Wong, "A modified differential evolution algorithm with fitness sharing for power system planning," *IEEE Trans. Power Syst.*, vol. 23, no. 2, pp. 514–522, May 2008.

[7] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. New York: Springer-Verlag, 1994.

[8] Z. Tu and Y. Lu, "A robust stochastic genetic algorithm (StGA) for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 456–470, Oct. 2004.

[9] C. Perales-Gravan and R. Lahoz-Beltra, "An AM radio receiver designed with a genetic algorithm based on a bacterial conjugation genetic operator," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 129–142, Apr. 2008.

[10] C. C. A. Rajan and M. R. Mohan, "An evolutionary programming-based tabu search method for solving the unit commitment problem," *IEEE Trans. Power Syst.*, vol. 19, no. 1, pp. 577–585, Feb. 2004.

[11] P. C. H. Ma, K. C. C. Chan, Y. Xin, and D. K. Y. Chiu, "An evolutionary clustering algorithm for gene expression microarray data analysis," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 296–314, Jun. 2006.

[12] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA: Bradford Company, 2004.

[13] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[14] N. Monmarche, G. Venturini, and M. Slimane, "On how Pachycondyla apicalis ants suggest a new search algorithm," *Future Gener. Comput. Syst.*, vol. 16, no. 8, pp. 937–946, Jun. 2000.

[15] K. E. Parsopoulos and M. N. Vrahatis, "On the computation of all global minimizers through particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 211–224, Jun. 2004.

[16] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.

[17] I. Paenke, J. Branke, and Y. Jin, "On the influence of phenotype plasticity on genotype diversity," in *Proc. IEEE Symp. FOCI*, Honolulu, HI, Apr. 2007, pp. 33–40.

[18] Y. Jin, "Guest editorial: Special issue on knowledge extraction and incorporation in evolutionary computation," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 35, no. 2, pp. 129–130, May 2005.

[19] M. Lozano, F. Herrera, N. Krasnogor, and D. Molina, "Real-coded memetic algorithms with crossover hill-climbing," *Evol. Comput.*, vol. 12, no. 3, pp. 273–302, Sep. 2004.

[20] N. Noman and H. Iba, "Differential evolution for economic load dispatch problems," *Elect. Power Syst. Res.*, vol. 78, no. 8, pp. 1322–1331, Aug. 2008.

[21] F. Olivetti de Franca, G. P. Coelho, F. J. Von Zuben, and R. R. de Faissol Attux, "Multivariate ant colony optimization in continuous search spaces," in *Proc. 10th Annu. Conf. Genetic Evol. Comput.*, Atlanta, GA, 2008, pp. 9–16.

[22] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1155–1173, Mar. 2008.

[23] S. H. Pourtakdoust and H. Nobahari, "An extension of ant colony system to continuous optimization problems," in *Proc. ANTS Workshop*, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Brussels, Belgium, Sep. 2004, pp. 294–301.

[24] G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," in *Proc. Sel. Papers From AISB Workshop Evol. Comput.*, 1995, pp. 25–39.

[25] A. Acan, "GAACO: A GA + ACO hybrid for faster and better search capability," in *Proc. 3rd Int. Workshop Ant Algorithms*, Brussels, Belgium, Sep. 2002, pp. 15–26.

[26] G. Li, Z. Lv, and H. Sun, "Study of available transfer capability based on hybrid continuous ant colony optimization," in *Proc. 3rd Int. Conf. Elect. Utility Deregulation Restruct. Power Technol. (DRPT)*, Nanjing, China, Apr. 2008, pp. 984–989.

[27] S. Nemati, M. E. Basiri, N. Ghasem-Aghaee, and M. H. Aghdam, "A novel ACO-GA hybrid algorithm for feature selection in protein function prediction," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 12 086–12 094, Dec. 2009.

[28] B. Liu and P. Meng, "Hybrid algorithm combining ant colony algorithm with genetic algorithm for continuous domain," in *Proc. 9th ICYCS*, Hunan, China, Nov. 2008, pp. 1819–1824.

[29] N. Monmarché, "Algorithmes de fourmis artificielles: Applications a la classification et a l'optimisation," Ph.D. dissertation, Laboratories d'Informatique, Université de Tours, Tours, France, 2000.

[30] D. Fresneau, "Biologie et comportement social d'une fourmi ponérine néotropicale," Ph.D. dissertation, Laboratoire d'Ethologie Expérimentale et Comparée, Université de Paris XIII, Paris, France, 1994.

[31] Q. Lv and X. Xia, "Towards termination criteria of ant colony optimization," in *Proc. 3rd ICNC*, Haikou, China, Aug. 2007, pp. 276–282.

[32] Y. Wang and C. Dang, "An evolutionary algorithm for global optimization based on level-set evolution and Latin squares," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 579–595, Oct. 2007.

Irina Ciornei (S'07) received the Bachelor's degree in power and electrical engineering from the Technical University of Iasi, Iasi, Romania, in 2002 and the M.Sc. degree in energy and environment engineering from the same university and from the "Ecole Superieur Politechnique," Poitiers, France, in 2003. She is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus.

She is a Researcher with the KIOS Research Center for Intelligent Systems and Networks, Department of Electrical and Computer Engineering, University of Cyprus. Between 2003–2006, she worked as a Research Assistant at the Technical University of Iasi, where she worked on a number of feasibility studies for energy producers. Her research interests include heuristic methods for optimization (GA, API, SA, and PSO), economic dispatch in electrical power systems, integration of wind energy into the main power grid, and power system analysis.

Elias Kyriakides (S'00–M'04–SM'09) received the B.Sc. degree from the Illinois Institute of Technology, Chicago, in 2000, and the M.Sc. and Ph.D. degrees from Arizona State University, Tempe, in 2001 and 2003, respectively, all in electrical engineering.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus, and a founding member of the KIOS Research Center for Intelligent Systems and Networks. He is the Action Chair of the ESF-COST Action IC0806 "Intelligent Monitoring, Control, and Security of Critical Infrastructure Systems" (IntelliCIS) (2009–2013). His research interests include synchronized measurements in power systems, security and reliability of the power system network, optimization of power system operation techniques, modeling of electric machines, and renewable energy sources.