Incremental Social Learning in Swarm Intelligence Algorithms for Continuous Optimization

Marco A. Montes de Oca

Department of Mathematical Sciences, University of Delaware, Newark, DE, U.S.A.
mmontes@math.udel.edu

Abstract. Swarm intelligence is the collective problem-solving behavior of groups of animals and artificial agents. Often, swarm intelligence is the result of self-organization, which emerges from the agents' local interactions with one another and with their environment. Such local interactions can be positive, negative, or neutral. Positive interactions help a swarm of agents solve a problem. Negative interactions are those that block or hinder the agents' task-performing behavior. Neutral interactions do not affect the swarm's performance. Reducing the effects of negative interactions is one of the main tasks of a designer of effective swarm intelligence systems. Traditionally, this has been done through the complexification of the behavior and/or the characteristics of the agents that comprise the system, which limits scalability and increases the difficulty of the design task. In collaboration with colleagues, I have proposed a framework, called incremental social learning (ISL), as a means to reduce the effects of negative interactions without complexifying the agents' behavior or characteristics. In this paper, I describe the ISL framework and three instantiations of it, which demonstrate the framework's effectiveness. The swarm intelligence systems used as case studies are the particle swarm optimization algorithm, ant colony optimization algorithm for continuous domains, and the artificial bee colony optimization algorithm.

1 Introduction

Some animals form large groups that behave so coherently and purposefully that they truly seem to be superorganisms with a mind of their own [5]. These groups are often called *swarms* because the individuals that comprise them are usually of the same kind and are so numerous that they resemble true insect swarms. If the behavior of a swarm allows it to solve problems beyond the capabilities of any of its members, then we say that the swarm exhibits *swarm intelligence* [3]. One of the best known examples of swarm intelligence is the ability of ant colonies to discover the shortest path between their nest and a food source [11]. The members of a swarm usually cannot perceive or interact with all the other members of the swarm at the same time. Instead, swarm members interact with one another and with their environment only locally. As a result, a swarm member cannot possibly supervise or dictate the actions of all the other swarm members. This restriction implies that swarm intelligence is often the result of self-organization, which is a process through which patterns at the collective level of a system emerge as a result of local interactions among its lower level components [4]. Other mechanisms through which swarm intelligence may be obtained are leadership, blueprints, recipes, templates, or threshold-based responses [4,9].

K. Madani et al. (Eds.): *Computational Intelligence*, SCI 465, pp. 31–45.
DOI: 10.1007/978-3-642-35638-4_3 © Springer-Verlag Berlin Heidelberg 2013

Through the study of natural swarm intelligence systems, scientists have identified a number of principles and mechanisms that make swarm intelligence possible [9]. The existence of these principles and mechanisms makes the design of artificial swarm intelligence systems possible because we can make robots or software agents use the same or similar rules to the ones animals use. The first efforts toward the development of artificial swarm intelligence systems began in the 1990s with pioneering works in robotics, data mining, and optimization [6]. In this paper, I focus on swarm intelligence systems for optimization, which have been very successful in practice [31,35].

A swarm intelligence algorithm for optimization consists of a set of agents, called swarm, or colony, that either generates candidate solutions or represents the actual set of candidate solutions. For example, in particle swarm optimization (PSO) algorithms [18], the swarm is composed of "particles" whose positions in the search space represent candidate solutions (see Section 4.1). In ant colony optimization (ACO) algorithms [7], the colony is made of "ants" that generate solutions in an incremental way guided by "pheromones" (see Section 4.2). In any case, the size of the swarm or colony is a parameter that determines the number of candidate solutions generated at each iteration of the algorithm: the larger the swarm, the more candidate solutions are generated and tried per iteration. The effect of the swarm size on the algorithms' performance depends on the amount of time allocated to the optimization task [7,27]: If a long time is available, large swarms usually return better results than small swarms. On the contrary, if only a short amount time is allocated, small swarms return better results than large swarms. In Section 2, I provide an explanation of this phenomenon in terms of positive and negative interactions among agents. For the moment, it is enough to say that the discovery of good solutions to an optimization problem typically occurs when swarms are near a convergence state. Thus, since small swarms reach a convergence state sooner than large swarms, it follows that small swarms discover good solutions before large swarms. However, small swarms converge before the allocated time runs out, which causes search stagnation.

In the context of optimization, the incremental social learning (ISL) framework [26,29,24] exploits the faster convergence of small swarms while avoiding search stagnation. This is accomplished by varying the population size over time. An optimization algorithm instantiating the ISL framework starts with a small population in order to find good quality solutions early in the optimization process. As time moves forward, new individuals are added to the population in order to avoid search stagnation. The newly added individuals are not generated at random. They are initialized using information already present in the population through a process that simulates social learning, that is, the transmission of knowledge from one individual to another. These two elements, an incremental deployment of individuals and the social learning-based initialization of new individuals, are the core of the ISL framework. The actual implementation of these elements may vary from system to system but the goals of each element remain the same. ISL has not only been used in optimization but also in swarm robotics [28,24]. In both cases, an important improvement of the system's performance has been obtained. With this paper, I hope to spark interest in the application and theoretical study of the ISL framework.

The rest of the paper is structured as follows. In Section 2, I explain the three kinds of interactions that occur in multiagent systems, including swarm intelligence systems. This explanation motivates the introduction of the ISL framework, which is described in detail in Section 3. In Section 4, I describe the three case studies that are used to show the effectiveness of the ISL framework in the context of optimization. I close this paper with some conclusions in Section 5.

2 Interactions in Multiagent and Swarm Intelligence Systems

In multiagent systems, including swarm intelligence systems, individual agents interact with one another and with their environment in order to perform a task. It is possible to classify all inter-agent and agent-environment interactions as "positive", "negative", or "neutral" based on whether they help the system achieve its goals or not [10]. Interactions that facilitate the accomplishment of the agents' assigned task are called positive. For example, a positive interaction would be one in which agents cooperate to perform a task that agents could not if they acted individually (see e.g., [19]). Negative interactions, also called *interference* [23], *friction* [10], or *repulsive and competitive interactions* [14], are those that block or hinder the ability of the system's constituent agents to perform the assigned task. Since negative interactions are an obstacle toward the efficient completion of a task, they decrease the performance of the system. For instance, in swarm intelligence algorithms for data clustering [13], agents can undo the actions of other agents, which increases the time needed to find a satisfactory final clustering. An interaction that does not benefit or harm progress toward the completion of a task is called neutral. An example of a neutral interaction could be a message exchange between two agents that just confirms information they already have and thus do not have to change plans.

Three difficulties arise when trying to directly measure the effects of interactions in a multiagent system. First, in many systems agent interactions are not predictable, that is, it is impossible to know in advance whether any two agents will interact and whether they will do so positively, negatively, or neutrally. Consequently, one can determine whether the effects of an interaction are beneficial or not only after the interaction has occurred. Second, an interaction may be positive, negative or neutral, depending on the time scale used to measure its effect. For example, an interaction that involves two robots performing collision avoidance can be labeled as a negative interaction in the short term because time is spent unproductively. However, if the time horizon of the task the robots are performing is significantly longer than the time frame of a collision avoidance maneuver, then the overall effect of such an interaction may be negligible. In this case, such interaction may be labeled as neutral. Third, the nature of the interactions themselves poses a challenge. In some systems, agents interact directly on a one-to-one or one-to-many basis. In other systems, agents interact stigmergically [12], that is, indirectly through the environment. Stigmergy makes the classification of interactions difficult because there may be extended periods of time between the moment an agent acts and the moment another agent (or even the acting agent itself) is affected by those actions. With these difficulties, the only practical way to measure the effects of interactions is to do it indirectly through the observation of the system's performance.

This is the approach my colleagues and I have taken to measure the effects of ISL (see Section 4).

Swarm intelligence systems are special kinds of multiagent systems. In contrast with traditional multiagent systems in which agents usually play a very specific role, swarm intelligence systems are usually composed of identical individuals. This feature has profound effects on the difficulty of the design task. The main problem is that the designer of a swarm intelligence system has to devise individual-level behaviors that foster positive interactions and, at the same time, minimize the number of negative interactions. Unfortunately, it is not always possible to achieve both goals simultaneously. For example, Kennedy and Eberhart [18], the designers of the first PSO algorithm, pondered different candidate particle interaction rules before proposing the rules that we now know (see Section 4.1). Their ultimate goal was to design rules that promoted positive interactions between particles. In the final design, particles cooperate, that is, they engage in positive interactions, by exchanging information with one another about the best solution to an optimization problem that each particle finds during its lifetime. It is hoped that this information exchange helps the algorithm improve the quality of the solutions by making particles move toward promising regions in the search space. At the same time, however, such an exchange of information may make particles evaluate regions of the search space that may in fact not contain the optimal solution or improve their current best estimate. When this happens, objective function evaluations are spent unproductively. The trade-off between solution quality and speed that many optimization algorithms exhibit is the result if these opposite-effect processes. As I said earlier, it is not possible to know in advance which particle interactions will be positive, or negative and thus a balance between these two kinds of interactions is always sought, usually through appropriate parameter settings [21].

Despite the aforementioned difficulties, swarm intelligence systems often exhibit the following two properties that make the management of negative interactions possible:

1. The number of negative interactions increases with the number of agents in the system. This effect is the result of the increased number of interactions within the system. The larger the number of agents that comprise the system, the more frequently negative interactions occur.
2. The number of negative interactions tends to decrease over time. At one extreme of the spectrum, one can find a system in which interactions between agents are completely random or not purposeful. In such a case, it is expected that agents cannot coordinate and thus, cannot perform useful work. As a result, the number of negative interactions remains constant over time. At the other extreme of the spectrum, one finds well-behaved systems consisting of a number of agents whose interaction rules are designed in order to make agents coordinate with each other. Initially, it is expected that many negative interactions occur because agents would not have enough knowledge about their current environment. However, over time, the behavioral rules of these agents would exploit any gained knowledge in order to make progress toward the completion of the assigned task. Thus, in cases like these, the number of negative interactions decreases over time.

The incremental social learning framework, which will be described next, exploits the two aforementioned properties in order to control, up to a certain extent, the number of negative interactions in a swarm intelligence system.

3 Incremental Social Learning

A framework called incremental social learning (ISL) was proposed by the author and colleagues [26,29,24] to reduce the effects of negative interactions in swarm intelligence systems. As a framework, ISL offers a conceptual algorithmic structure that does not prescribe a specific implementation of the ideas on which it relies. Each instantiation of ISL will benefit from knowledge about the specific application domain, and therefore, specific properties of the framework should be analyzed in an application-dependent context.

The ISL framework consists of two elements that exploit the two properties mentioned in Section 2. The first element of the framework directly reduces the number of negative interactions within a system by manipulating the number of agents. The strategy for controlling the size of the agent population exploits the second property, that is, that the number of negative interactions tends to decrease over time. Under the control of ISL, a system starts with a small population. Over time, the population grows at a rate determined by a user-defined agent addition criterion. Two phenomena with opposite effects occur while the system is under the control of the ISL framework. On the one hand, the number of negative interactions increases as a result of adding new agents to the swarm (first property described in Section 2). On the other hand, the number of negative interactions decreases because the system naturally tends toward a state in which fewer negative interactions occur (second property described in Section 2). The second element of the framework is social learning. This element is present before a new agent freely interacts with its peers. Social learning is used so that the new agent does not disrupt the system's operation due to its lack of knowledge about the environment or the task. Leadership, a swarm intelligence mechanism [4,9], is present in the framework in the process of selecting a subset of agents from which the new agent learns. The best strategy to select such a set depends on the specific application. However, even in the case in which a random agent is chosen as a "model" to learn from, knowledge transfer occurs because the selected agent will have more experience than the new agent that is about to be added.

The two elements that compose ISL are executed iteratively as shown in Algorithm 1. In a typical implementation of the ISL framework, an initial population of agents is created and initialized (line 4). The size of the initial population depends on the specific application. In any case, the size of this initial population should be small in order to reduce interference to the lowest possible level. A loop allows the interspersed execution of the underlying system and the creation and initialization of new agents (line 7). This loop is executed until some user-specified stopping criteria are met. Stopping criteria can be specific to the application or related to the ISL framework. For example, the framework may stop when the task assigned to the swarm intelligence system is completed or when a maximum number of agents are reached. While executing the main loop, agent addition criteria, which are also supplied by the user, are repeatedly evaluated (line 8). The criteria can range from a predefined schedule to conditions based on

Algorithm 1. Incremental social learning framework

```
Input: Agent addition criteria, stopping criteria
 1: /* Initialization */
 2: $t \leftarrow 0$
 3: Initialize environment $\mathbf{E}^t$
 4: Initialize population of agents $\mathbf{X}^t$
 5:
 6: /* Main loop */
 7: while Stopping criteria not met do
 8:    if Agent addition criteria is not met then
 9:       default($\mathbf{X}^t, \mathbf{E}^t$) /* Default system */
10:    else
11:       Create new agent $a_{new}$
12:       slearn$(a_{new}, \mathbf{X}^t)$ /* Social learning */
13:       $\mathbf{X}^{t+1} \leftarrow \mathbf{X}^t \cup \{a_{new}\}$
14:    end if
15:    $\mathbf{E}^{t+1} \leftarrow$ update($\mathbf{E}^t$) /* Update environment */
16:    $t \leftarrow t + 1$
17: end while
```

statistics of the system's progress. If the agent addition criteria are not met, the set of agents work normally, that is, the underlying swarm intelligence system is executed. In line 9, such an event is denoted by a call to the procedure default$(\mathbf{X}^t, \mathbf{E}^t)$. If the agent addition criteria are satisfied, a new agent is created (line 11). In contrast to a default initialization such as the one in line 4, this new agent is initialized with information extracted from a subset of the currently active population (line 12). Such an initialization is denoted by a call to the procedure slearn$(a_{new}, \mathbf{X}^t)$. This procedure is responsible for the selection of the agents from which the new agent will learn, and for the actual implementation of the social learning mechanism. Once the new agent is properly initialized, it becomes part of the system (line 13). In line 15, we explicitly update the environment. However, in a real implementation, the environment may be continuously updated as a result of the system's operation.

In most swarm intelligence systems, the population of agents is large and homogeneous, that is, it is composed of agents that follow exactly the same behavioral rules. Thus, any knowledge acquired by an agent is likely to be useful for another one. The social learning mechanism used in an instantiation of the ISL framework should allow the transfer of knowledge from one agent to the other. In some cases, it is possible to have access to the full state of the agent that serves as a "model" to be imitated, and thus, the social learning mechanism is simple. In other cases, access to the model agent's state may be limited and a more sophisticated mechanism is required. In most cases, the result of the social learning mechanism will not be simply a copy of the model agent's state, but a biased initialization toward it. Copying is not always a good idea because what may work very well for an agent in a system composed of n agents may not work well in a system of $n + 1$ agents.

4 Case Studies

In this section, I will briefly describe the three case studies that colleagues and I used in order to measure the effectiveness of ISL in the context of optimization. The swarm intelligence algorithms used were the particle swarm optimization (PSO) algorithm [18], the ant colony optimization algorithm for continuous domains (ACO$_\mathbb{R}$) [33], and the artificial bee colony (ABC) algorithm [17].

4.1 Case Study 1: Particle Swarm Optimization

The Basic Algorithm. In PSO algorithms [18], very simple agents, called *particles*, form a *swarm* and move in an optimization problem's search space. Each particle's position represents a candidate solution to the optimization problem. The position and velocity of the i-th particle along the j-th coordinate of the problem's search space at iteration t are represented by $x^t_{i,j}$ and $v^t_{i,j}$, respectively. The core of the PSO algorithm is the set of rules that are used to update these two quantities. These rules are:

$$v^{t+1}_{i,j} = wv^t_{i,j} + \mathrm{U}(0,\varphi_1)(p^t_{i,j} - x^t_{i,j}) + \mathrm{U}(0,\varphi_2)(l^t_{i,j} - x^t_{i,j})\,, \tag{1}$$

$$x^{t+1}_{i,j} = x^t_{i,j} + v^{t+1}_{i,j}\,, \tag{2}$$

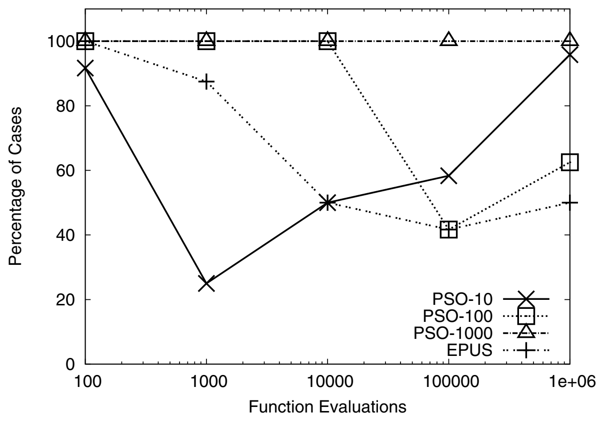
where w, φ_1 and φ_2 are parameters of the algorithm, $\mathrm{U}(a,b)$ represents a call to a random number generator that returns a uniformly distributed random number in the range $[a,b)$, $p^t_{i,j}$ represents the j-th component of the best solution ever visited by the i-th particle, and $l^t_{i,j}$ represents the j-th component of the best solution ever visited by a subset of the swarm referred to as the i-th particle's *neighborhood*. The definition of each particle's neighborhood is usually parametric, fixed, and set before the algorithm is run.

Integration with ISL. The ISL framework can be instantiated in different ways in the context of PSO algorithms. Here, I present the most basic variant, which was first described in [26] and benchmarked in [29]. A more sophisticated variant that exhibits a much better performance is presented in [25].

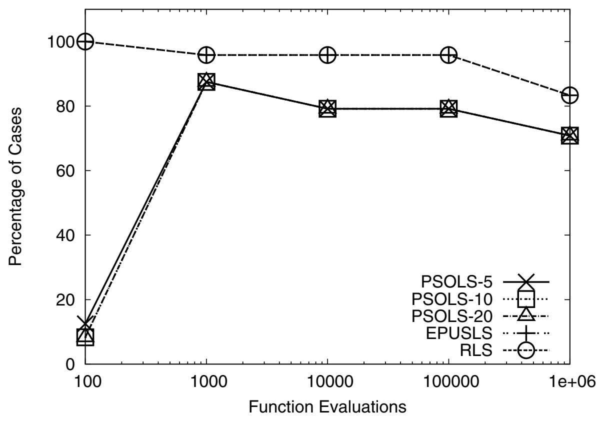
The most basic instantiation of the ISL framework in the context of PSO algorithms is a PSO algorithm with a growing population size called incremental particle swarm optimizer (IPSO). In IPSO, every time a new particle is added, it is initialized using the following rule:

$$x'_{new,j} = x_{new,j} + U(p_{model,j} - x_{new,j}), \tag{3}$$

where $x'_{new,j}$ is the new particle's updated position, $x_{new,j}$ is the new particle's original random position, $p_{model,j}$ is the model particle's previous best position, and U is a uniformly distributed random number in the range $[0,1)$. This rule moves a new particle from an initial randomly generated position in the problem's search space to one that is closer to the position of a model particle. Once the rule is applied for each dimension, the new particle's previous best position is initialized to the point $\boldsymbol{x}'_{new}$ and its velocity is set to zero. The random number U is the same for all dimensions in order to ensure



a) PSO – No local search



b) IPSOLS – With local search

Fig. 1. Percentage of test cases in which the performance of IPSO (a) and IPSOLS (b) is better or no worse (according to a Wilcoxon test at a significance level of 0.05) than the performance of other comparable algorithms

that the new particle's updated previous best position will lie somewhere along the direct attraction vector $\boldsymbol{p}_{model} - \boldsymbol{x}_{new}$. Finally, the new particle's neighborhood, that is, the set of particles from which it will receive information in subsequent iterations, is generated at random using the same parameters used to generate the rest of the particles' neighborhoods.

A better performing variant of IPSO, called IPSOLS, uses a local search procedure. In the context of the ISL framework, a call to a local search procedure may be interpreted as a particle's "individual learning" ability since it allows a particle to improve its solution in the absence of any social influence. In experiments with IPSOLS and other algorithms, we used Powell's conjugate directions set method [32] as local search.

Results. A condensed view of the results obtained with IPSO and IPSOLS in [29] is shown in Fig. 1. The figure shows the percentage of test cases (a total of 24 cases: 12 benchmark functions and two neighborhood types) in which the performance of IPSO and IPSOLS is better than or indistinguishable from the performance of reference algorithms (in a statistical sense). The reference algorithms are a PSO algorithm with three different population sizes and a PSO algorithm with a sophisticated mechanism for changing the population size over time (EPUS) [16]. The algorithms used in the comparison with IPSOLS are also a PSO algorithm with local search and three different population sizes, EPUS with local search, and a randomly restarted local search. Details about the experimental setup can be found in [29].

Using IPSO is advantageous when the optimal population size for a particular budget in terms function evaluations is not known in advance. IPSO is advantageous in these cases because a specific population size will produce acceptable results only for runs of a particular length. For example, in our experiments, a PSO algorithm with 10 particles returned good results only for runs of 1000 function evaluations. However, if more time is available, 10 particles return poor results in comparison with larger swarms. In contrast, IPSO has a competitive performance for runs of different length as can be seen in Fig. 1. In any case, the absolute quality of the results obtained with the use of a local search procedure is much better than without local search. Thus, the results in subfigure (b) are more interesting. Here, IPSOLS's performance is clearly better than that of the other algorithms. The reason is that the combination of ISL, PSO and a local search procedure makes particles in IPSOLS move from one local optimum to another [30], producing high quality solutions in a few iterations of the algorithm.

4.2 Case Study 2: Ant Colony Optimization

The Basic Algorithm. ACO$_{\mathbb{R}}$ [33] maintains a solution archive of size k that is used to keep track of the most promising solutions and their distribution over the search space. Initially, the solution archive is filled with randomly generated solutions. The archive is then updated as follows. At each iteration, m new solutions are generated and from the $k+m$ solutions that become available, only the best k solutions are kept. The mechanism responsible for the generation of new solutions samples values around the solutions s_i with $i \in \{1,\ldots,k\}$ in the archive. This is done on a coordinate-per-coordinate basis using Gaussian kernels defined as sums of weighted Gaussian functions. The Gaussian kernel for coordinate j is

$$G_j(x) = \sum_{i=1}^{k} \omega_i \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ij})^2}{2{\sigma_{ij}}^2}}, \tag{4}$$

where $j \in \{1,\ldots,D\}$ and D is the problem's dimensionality. The mean and variance of these Gaussian functions are set as follows: $\mu_{ij} = s_{ij}$, and

$$\sigma_{ij} = \xi \sum_{r=1}^{k} \frac{|s_{rj} - s_{ij}|}{k-1}, \tag{5}$$

which is the average distance between the j-th component of the solution s_i and the j-th component of the other solutions in the archive, multiplied by a parameter ξ.

The weight ω_i associated with solution s_i depends on its quality, represented by its ranking in the archive, rank(i) (the best solution is ranked first and the worst solution is ranked last). This weight is calculated using also a Gaussian function:

$$\omega_i = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(\text{rank}(i)-1)^2}{2q^2k^2}} , \tag{6}$$

where q is a parameter of the algorithm. During the solution generation process, each coordinate is treated independently. For generating the j-th component of a new solution, the algorithm chooses first an archive solution with a probability proportional to its weight. Then, the algorithm generates a normally-distributed random number with mean and variance equal to μ_{ij} and σ_{ij} as defined above. This number is the j-th component of the new solution. This process repeated m times for each dimension $j \in \{1, ..., D\}$ in order to generate m new candidate solutions.

Integration with ISL. The instantiation of the ISL framework with the ACO$_\mathbb{R}$ algorithm requires increasing the number of solutions handled per iteration and the biased initialization of new solutions. The resulting algorithm is called IACO$_\mathbb{R}$ if no local search is used, and IACO$_\mathbb{R}$-LS if it is. These algorithms were first proposed in [20].

In IACO$_\mathbb{R}$ the initial size of the solution archive is small. As the optimization process proceeds, new solutions are added to the solution archive at a rate determined by a user-specified criterion. New solutions are initialized using information from a subset of the solutions in the archive (usually the best solution). The rule used to bias the initialization of new solutions is the same as in IPSO (see Eq. 3).

IACO$_\mathbb{R}$ differs from the original ACO$_\mathbb{R}$ algorithm in the way the solution archive is updated. In IACO$_\mathbb{R}$, once a guiding solution is selected, and a new one is generated (in exactly the same way as in ACO$_\mathbb{R}$), they are compared. If the newly generated solution is better than the guiding solution, it replaces it in the archive. In contrast, in ACO$_\mathbb{R}$ all solutions, new and old, compete at the same time for a slot in the solution archive. Another difference is the mechanism for selecting the guiding solution in the archive. In IACO$_\mathbb{R}$, the best solution in the archive is used as guiding solution with probability p. With a probability $1 - p$, all the solutions in the archive are used to generate new solutions. Finally, IACO$_\mathbb{R}$ is restarted (keeping the best-so-far solution) if the best solution is improved less than a certain threshold for a number of consecutive iterations.

As in the PSO case, the quality of the solutions found with IACO$_\mathbb{R}$ typically improve if a local search method is used. In our experiments, we measured the performance of IACO$_\mathbb{R}$-LS with Powell's conjugate directions set [32] and Lin-Yu Tseng's mtsls1 [36] methods as local search procedures.

Results. To benchmark IACO$_\mathbb{R}$-LS, colleagues and I followed the protocol proposed by Lozano *et al.* for a special issue on large-scale optimization in the Soft Computing Journal [22]. We compared the results obtained by IACO$_\mathbb{R}$-LS with those obtained with IPSOLS (the version described in [25]) and other 15 algorithms. The results are shown in Fig. 2.

IACO$_\mathbb{R}$-LS using mtsls1 as a local search is among the best performing algorithms. In at least eight benchmark functions, IACO$_\mathbb{R}$-mtsls1 found an average solution quality at least equal to 10^{-14}. Some of these functions are the well-known Rosenbrock and Rastrigin functions. These results are thus remarkable considering the fact that these

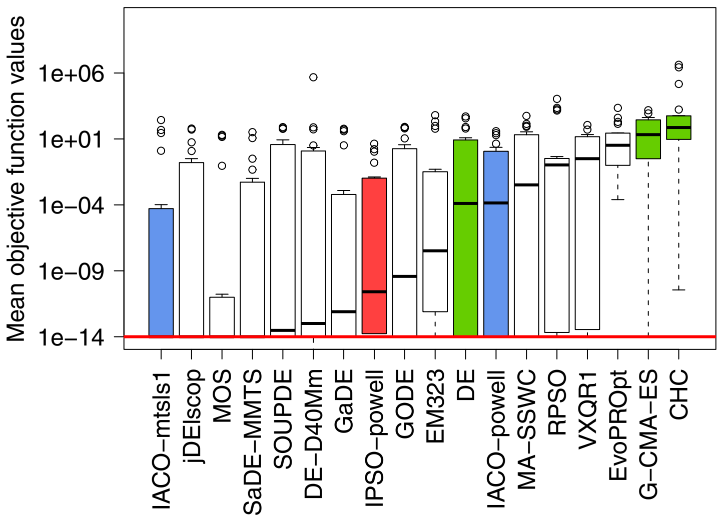


Fig. 2. Comparison of IACO$_{\mathbb{R}}$-LS with other algorithms: Distribution of average values (over 25 runs) obtained across 19 functions in 100 dimensions after up to 500,000 function evaluations. Values at or below the threshold 10^{-14} are considered "zero" for numerical reasons. DE [34], G-CMA-ES [1], and CHC [8] were proposed by Lozano *et al.* [22] as reference algorithms. The benchmark functions used are described in [15].

functions cannot usually be solved to such a precision level. An interesting result of this comparison comes from the fact that G-CMA-ES [1], which many still consider a state-of-the-art optimization algorithm, is among the worst performing algorithm. This result does not mean that G-CMA-ES is not a good algorithm, but that it does not scale well with the problem's size. Therefore, for large-scale problems, IACO$_{\mathbb{R}}$-mtsls1 can be considered a representative algorithm of the state of the art.

4.3 Case Study 3: Artificial Bee Colony Optimization

The Basic Algorithm. The design of the artificial bee colony (ABC) algorithm [17] is inspired by the foraging behavior of honeybee swarms, in particular, the recruitment of honeybees to good food sources. The first step of this algorithm is to randomly place a number SN of candidate solutions, called *food sources*, in the problem's search space. The algorithm's goal is to discover better food sources (improve the quality of candidate solutions). This is done as follows: First, simple agents called *employed bees* select uniformly at random a food source and explore another location using the following rule:

$$v_{i,j} = x_{i,j} + \mathrm{U}(-1,1)(x_{i,j} - x_{k,j}),\ i \neq k\,, \tag{7}$$

where i, $k \in \{1, 2, \ldots, SN\}$, $j \in \{1, 2, \ldots, D\}$, x_{ij} and x_{kj} are the position of the reference food source i and a randomly selected food source k in dimension j, respectively. The better food source between the new and the reference food sources is kept by the algorithm. The next step is performed by another kind of agent called *onlooker* *bee*, which looks for better food sources around other food sources based on their quality. This is done by first selecting a reference food source with a probability based on its quality so that better food sources are more attractive. This step is responsible for the intensification behavior of the algorithm since information about good solutions is exploited. The third step is performed by so-called *scout bees*. In this step, a number of food sources that have not been improved for a predetermined number of iterations (controlled by a parameter *limit*), are detected and abandoned. Then, scout bees search for a new food source randomly in the whole search space.

Integration with ISL. IABC and IABC-LS were proposed in [2]. From these two algorithms, IABC-LS is the better performing. In ABC-LS, the number of food sources increases over time according to a predefined schedule. Initially, only a few sources are used. New food sources are placed using Eq. 3. Scout bees in IABC-LS use a similar rule when exploring the search space. This rule is

$$x'_{\text{new},j} = x_{\text{best},j} + R_{\text{factor}}(x_{\text{best},j} - x_{\text{new},j}) \,, \tag{8}$$

where R_{factor} is a parameter that controls how close to the best-so-far food source the new food source will be. IABC-LS also differs from the original ABC algorithm in the way employed bees select the food source around which they explore. In IABC-LS, employed bees search around the best food source instead of around a randomly chosen one in order to enhance the search intensification. IABC-LS is a hybrid algorithm that calls a local search procedure at each iteration. The best-so-far food source location is usually used as the initial solution from which the local search is called. The result

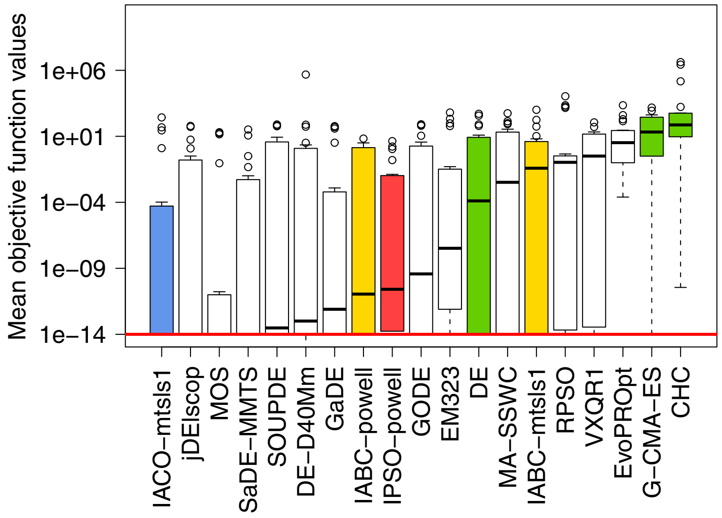


Fig. 3. Comparison of IABC-LS with other algorithms: Distribution of average values (over 25 runs) obtained across 19 functions in 100 dimensions after up to 500,000 function evaluations. Values at or below the threshold 10^{-14} are considered “zero” for numerical reasons. DE [34], G-CMA-ES [1], and CHC [8] were proposed by Lozano *et al.* [22] as reference algorithms. The benchmark functions used are described in [15].

of the local search replaces the best-so-far solution if there is an improvement on the initial solution. To fight stagnation, the local search procedure may be applied from a randomly chosen solution if the best-so-far solution cannot be improved any further.

Results. To measure the performance of IABC-LS, the same protocol used to benchmark IACO$_{\mathbb{R}}$-LS was used. The results are shown in Fig. 3. IABC-LS using Powell's conjugate directions set method as the local search component exhibits practically the same performance as IPSO-LS with the same local search. IABC-LS with mtsls1 as the local search method does not perform as well. These results together with the results obtained with IACO$_{\mathbb{R}}$-LS, suggest that the observed performance does not depend only on the local search method used, but on the interaction between the the incremental algorithm and the local search used. In any case, the instantiation of the ISL framework in the context of ABC algorithms also improves the performance of the original algorithm. ISL transformed an algorithm not known for being state of the art (ABC) into a highly competitive algorithm.

5 Conclusions

Engineered swarm intelligence systems are composed of agents that interact with one another and with their environment in order to accomplish a certain task. Usually, these systems are composed of agents that use the same behavioral rules; therefore, these rules must allow agents to engage in positive interactions (those that help the system accomplish the assigned task) and avoid negative interactions (those that block or hinder the agents' task-performing behavior). Typically, it is impossible to predict when any two agents will interact or whether they will do so positively. As a consequence, designers often complexify the behavioral rules of the agents, or the agents' characteristics. Both of these strategies limit the systems' scalability potential and make the design task more challenging.

The incremental social learning (ISL) framework was proposed to reduce the effects of negative interactions in swarm intelligence systems without requiring the complexification of the agents' behavioral rules or characteristics. Three case studies in the context of optimization were carried out in order to assess the effectiveness of the ISL framework. The three algorithms that served this purpose were particle swarm optimization, ant colony optimization for continuous domains ACO$_{\mathbb{R}}$, and artificial bee colony optimization. In each of these cases, the ISL framework improved the performance of the underlying algorithms, a sign of the reduced effect of negative interactions. The instantiation of the ISL framework with ACO$_{\mathbb{R}}$ resulted in a new state-of-the-art optimization algorithm for problems whose dimensionality makes them unsuitable to be dealt with other high performance algorithms such as G-CMA-ES.

Acknowledgements. I thank Agostinho Rosa and Joaquim Filipe for inviting me as a keynote speaker to the International Joint Conference on Computational Intelligence (IJCCI–2011), held on October 24–26, 2011 in Paris, France. I carried out the work reported in this paper in collaboration with Tianjun Liao, Doğan Aydın, Ken van den Enden, Thomas Stützle, and Marco Dorigo, while I was a doctoral student at the *Université libre de Bruxelles* in Brussels, Belgium.

References

1. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), pp. 1769–1776. IEEE Press, Piscataway (2005)
2. Aydın, D., Liao, T., Montes de Oca, M.A., Stützle, T.: Improving performance via population growth and local search: The case of the artificial bee colony algorithm. In: Proceedings of the International Conference on Artificial Evolution, EA 2011 (2011) (to appear)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, New York (1999)
4. Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraulaz, G., Bonabeau, E.: Self-Organization in Biological Systems. Princeton University Press, Princeton (2001)
5. Couzin, I.D.: Collective minds. Nature 445(7129), 715 (2007)
6. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia 2(9), 1462 (2007), `http://dx.doi.org/10.4249/scholarpedia.1462`
7. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Books. MIT Press, Cambridge (2004)
8. Eshelman, L.J., Schaffer, J.D.: Real-coded genetic algorithms and interval-schemata. In: Whitley, D.L. (ed.) Foundation of Genetic Algorithms 2, pp. 187–202. Morgan Kaufmann, San Mateo (1993)
9. Garnier, S., Gautrais, J., Theraulaz, G.: The biological principles of swarm intelligence. Swarm Intelligence 1(1), 3–31 (2007)
10. Gershenson, C.: Design and control of self-organizing systems. Ph.D. thesis, Vrije Universiteit Brussel, Brussels, Belgium (2007)
11. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the argentine ant. Naturwissenschaften 76(12), 579–581 (1989)
12. Grassé, P.P.: La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. Insectes Sociaux 6(1), 41–80 (1959)
13. Handl, J., Meyer, B.: Ant-based and swarm-based clustering. Swarm Intelligence 1(2), 95–113 (2007)
14. Helbing, D., Vicsek, T.: Optimal self-organization. New Journal of Physics 1, 13.1–13.17 (1999)
15. Herrera, F., Lozano, M., Molina, D.: Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems (2010), `http://sci2s.ugr.es/eamhco/updated-functions1-19.pdf` (last accessed: July 2010)
16. Hsieh, S.T., Sun, T.Y., Liu, C.C., Tsai, S.J.: Efficient population utilization strategy for particle swarm optimizer. IEEE Transactions on Systems, Man, and Cybernetics 39(2), 444–456 (2009)
17. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39(3), 459–471 (2007)
18. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE Press, Piscataway (1995)
19. Kube, C.R., Bonabeau, E.: Cooperative transport by ants and robots. Robotics and Autonomous Systems 30(1-2), 85–101 (2000)

20. Liao, T., Montes de Oca, M.A., Aydın, D., Stützle, T., Dorigo, M.: An incremental ant colony algorithm with local search for continuous optimization. In: Krasnogor, N., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011), pp. 125–132. ACM Press, New York (2011)
21. Lobo, F.G., Lima, C.F.: Adaptive Population Sizing Schemes in Genetic Algorithms. In: Parameter Setting in Evolutionary Algorithms. SCI, vol. 54, pp. 185–204. Springer, Heidelberg (2007)
22. Lozano, M., Molina, D., Herrera, F.: Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. Soft Computing 15(11), 2085–2087 (2011)
23. Matarić, M.J.: Learning social behavior. Robotics and Autonomous Systems 20(2-4), 191–204 (1997)
24. Montes de Oca, M.A.: Incremental social learning in swarm intelligence systems. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium (2011)
25. Montes de Oca, M.A., Aydın, D., Stützle, T.: An incremental particle swarm for large-scale optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. Soft Computing 15(11), 2233–2255 (2011)
26. Montes de Oca, M.A., Stützle, T.: Towards incremental social learning in optimization and multiagent systems. In: Rand, W., et al. (eds.) Workshop on Evolutionary Computation and Multiagent Systems Simulation of the Genetic and Evolutionary Computation Conference (GECCO 2008), pp. 1939–1944. ACM Press, New York (2008)
27. Montes de Oca, M.A., Stützle, T., Birattari, M., Dorigo, M.: Frankenstein's PSO: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation 13(5), 1120–1132 (2009)
28. Montes de Oca, M.A., Stützle, T., Birattari, M., Dorigo, M.: Incremental social learning applied to a decentralized decision-making mechanism: Collective learning made faster. In: Gupta, I., Hassas, S., Rolia, J. (eds.) Proceedings of the Fourth IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010), pp. 243–252. IEEE Computer Society Press, Los Alamitos (2010)
29. Montes de Oca, M.A., Stützle, T., Van den Enden, K., Dorigo, M.: Incremental social learning in particle swarms. IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics 41(2), 368–384 (2011)
30. Montes de Oca, M.A., Van den Enden, K., Stützle, T.: Incremental Particle Swarm-Guided Local Search for Continuous Optimization. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) HM 2008. LNCS, vol. 5296, pp. 72–86. Springer, Heidelberg (2008)
31. Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. Journal of Artificial Evolution and Applications, Article ID 685175, 10 pages (2008)
32. Powell, M.J.D.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7(2), 155–162 (1964)
33. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European Journal of Operational Research 185(3), 1155–1173 (2008)
34. Storn, R.M., Price, K.V.: Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11(4), 341–359 (1997)
35. Stützle, T., López-Ibáñez, M., Dorigo, M.: A concise overview of applications of ant colony optimization. In: Cochran, J.J., et al. (eds.) Wiley Encyclopedia of Operations Research and Management Science, vol. 2, pp. 896–911. John Wiley & Sons, Ltd., New York (2011)
36. Tseng, L., Chen, C.: Multiple trajectory search for large scale global optimization. In: Proceeding of the IEEE 2008 Congress on Evolutionary Computation (CEC 2008), pp. 3052–3059. IEEE Press, Piscataway (2008)