2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Differential Evolution in Constrained Sampling Problems

Gervasio Varela, Pilar Caamaño, Felix Orjales, Alvaro Deibe, Fernando Lopez-Pena
Richard J. Duro

***Abstract*— This work proposes a set of modifications to the Differential Evolution algorithm in order to make it more efficient in solving a particular category of problems, the so called Constrained Sampling problems. In this type of problems, which are usually related to the on-line real-world application of evolution, it is not always straightforward to evaluate the fitness landscapes due to the computational cost it implies or to physical constraints of the specific application. The fact is that the sampling or evaluation of the offspring points within the fitness landscape generally requires a decoding phase that implies physical changes over the parents or elements used for sampling the landscape, whether through some type of physical migration from their locations or through changes in their configurations. Here we propose a series of modifications to the Differential Evolution algorithm in order to improve its efficiency when applied to this type of problems. The approach is compared to a standard DE using some common real-coded benchmark functions and then it is applied to a real constrained sampling problem through a series of real experiments where a set of Unmanned Aerials Vehicles is used to find shipwrecked people.**

I. Introduction

Evolutionary Algorithms (EAs) have demonstrated a strong capacity for solving a large number of hard optimization problems, showing complicated fitness landscapes, that are difficult to address using more traditional and non-stochastic techniques. However, unlike other algorithms, such as Swarm Intelligence based strategies, they are usually relegated to off line optimization and are hardly ever used on-line in the case of real time real world applications. That is, they are seldom used as an intrinsic part of the operation of a real system, but rather as optimizers of the parameters of the system that has to work in real time and hardly ever used in applications where the fitness evaluation is not straightforward or where it is very expensive in terms of computational time.

Some inroads have been made by several authors when considering the problem of costly evaluations. Examples of these are to use surrogate models [1], which are approximations to the fitness functions using coarser models with lower computational requirements. Several techniques to approximate fitness such as approximation levels, approximate model management schemes or model construction techniques [2] have been analyzed for application to problems where the calculation of the fitness is extremely expensive, as in the case of, for instance, structural design optimization [1], protein structure prediction [3] or protein design and drug design [4].

Nevertheless, these techniques are quite specific to off-line modeling and evolution, and only address the problem of costly evaluations through approximations. There are other types of problems where the bottleneck is not really the evaluation of the individuals but rather their decoding (usually taken as part of the evaluation phase). In general, all EAs assume that every point of the search space can be directly, or almost directly, decoded, i.e., it is possible to create the instance of the solution represented by the individual in a reasonably easy way and, thus, it is possible to easily measure the fitness of any point in the fitness landscape.

There are, however, problems where this is not the case, especially when addressing real world problems for which evolutionary algorithms could be a good tool to be used as a part of the operation of the system that has to solve them. Just to name one, we could think of a system that needs to determine the area with the highest plankton content in the Pacific Ocean. This problem would require measuring or sampling the plankton at different points and depths in the ocean and using a strategy (perhaps evolutionary) to determine the next points to sample in order to be able to efficiently find the point with the highest plankton level. This is basically an optimization problem over a real fitness landscape and EAs are not very efficient in these situations as the points that should be sampled are usually far from the points over which the sampler is located after measuring the previous fitness value. Here, the decoding phase involves moving the sampler to a physical position of the solution space. In other words, in order to obtain the fitness of an offspring, its decoding is only possible through a series of transformations (in this case motions) from the decoded version of one of the parents or from that of another individual in the parent population.

It seems obvious that this poses the same requirements as exploring unknown search or solution spaces in optimization problems in order to find their optimum or optima. The only difference is the fact that in real time real world problems, this exploration is carried out by a set of real units or devices. Thus, as such, they are constrained by the limitations of reality in terms of their own physical structure and the physics of the real world (they cannot be omnipresent, change positions instantaneously, or perform certain maneuvers). The consequence of these limitations is that they are constrained in terms of what can be sampled from the environment at a given point in time or in a given interval of time. Here we are going to denote the type of problems where

All authors are with the Integrated Group for Engineering Research, University of A Coruña, Ferrol, Spain (phone: +34981337400, e-mail: {gvarela, pcsobrino, felix.orjales, adeibe, flop, richard}@ udc.es).

This work was partially funded by the Spanish MICINN and European Regional Development Funds through project TIN2011-28753-C02-01.

the sampling of the search space is limited by the physics of reality as Constrained Sampling (CS) problems.

Thus, strictly speaking, Constrained Sampling Problems (CS-Problems) are problems in which the constraints do not correspond to areas of the search space that are forbidden, but rather, to the fact that the entity that has to sample the points only has access to a limited neighborhood of its current position each instant of time. That is, they can be described as search or optimization problems in which it is necessary to physically reach a specific point of the solution space through a series of transformations from the decoded version of a previous point in order to be able to produce a fitness value. Physically reaching a point in the solution space may involve, in some cases, moving a sensor to that point or, in others, constructing a physical entity. The key to this process is that it is not instantaneous, it takes time and requires following a trajectory that leads to the solution point that needs to be evaluated (either through the motion of a sensing entity or through a series of construction or transformation steps) from the decoded version of a previous individual. Thus, in general, a CS-Problem implies a bottleneck in the application of the EA induced by the, sometimes very long times required for decoding the individuals, that is, for getting to a position or configuration that allows evaluating them. Many problems can be included in this category, most of them involving real world applications. Examples are coordinated search in unknown environments [5] which imply physically moving sensors in order to evaluate points; the evolution of modular robots [6], in which, every time an individual has to be evaluated, a new robot needs to be constructed (usually from pieces that were used in a previous instance and that must be disassembled); or Facility Layout Problems (FLPs) [7], in which the individuals of the population represent the configuration of production plants and, in order to measure the fitness of each one, a simulation model has to be constructed.

Several authors have proposed and tested certain types of modern evolutionary algorithms that have been shown to be very adequate in complex and rugged fitness landscapes or in flat fitness landscapes that provide little information for evolution; examples are CMA-ES [8], DE [9], and many others. However, as indicated before, they assume that reaching any point in the solution space is simple and immediate and thus impose no restriction within their operation on how the points to be sampled or evaluated next should be chosen. This leads to very inefficient behaviors in CS problems as they ignore information about the landscape that could be gathered during the decoding phase, as each step produces a new solution. Consequently, these problems have usually been addressed using other types of metaheuristics such as Particle Swarm Optimization (PSO) [11, 5]. EAs have only been used indirectly in these cases by some authors that concentrated on the off-line evolution of control strategies for the samplers that were able to use information from the environment for real-time improvements [12, 13].

To solve the efficiency problems of EAs in general, and the Differential Evolution algorithm in particular, the approach followed in this paper has been to adapt it to the CS nature of the problem. Specifically, we present a new version of the traditional Differential Evolution (DE) Algorithm. This version of the algorithm has been called the Constrained Sampling Differential Evolution Algorithm (CS-DE).

The approach is compared to a standard DE using some common real-coded benchmark functions and then it is applied to a real constrained sampling problem through a series of real experiments where a set of UAVs is used to find shipwreck survivors in the open sea and its efficiency compared to a traditional swarm optimization method.

The remainder of the paper is structured as follows. Section 2 describes the implementation and details of the constrained sampling Differential Evolution algorithm (CS-DE). It first considers a general presentation of Constrained Sampling Evolutionary Algorithms and then particularizes it to the Differential Evolution algorithm. Section 3 is devoted to testing the CS-DE over a set of benchmark functions and providing a comparison of its performance with respect to a standard DE. In Section 4 we provide an example of the application of CS-DE as an intrinsic part of the control of a set of Unmanned Aerial Vehicles (UAVs) that must perform a collaborative search operation. Finally, Section 5 provides some conclusions to this work.

II. Algorithms for Constrained Sampling Problems

In order to describe the approach presented here, we are going to consider the general case of any EA in the initial description and then particularize it to the Differential Evolution algorithm.

A. General Approach: CS-EA

The CS-EA (Constrained Sampling Evolutionary Algorithm) is designed as a series of modifications over a standard EA to intrinsically deal with the sampling constraints induced by CS-problems. Thus, a *base EA*, i.e. the Differential Evolution algorithm (DE), undergoes modifications in the way it handles its populations.

Thus, the execution cycle of any EA requires an offspring population ($P'(t)$, where t is the current generation) to be evaluated and compared to the parent population ($P(t)$) in order to produce the next generation parent population ($P(t+1)$). As indicated in the introduction, CS problems usually entail following lengthy and constrained trajectories (sequences of transformations) in solution space from the individuals in $P(t)$ to those in $P'(t)$ before the individuals in the latter population can be evaluated. Thus, $P'(t)$ can be taken as the target of the trajectories and in what follows we will denote it as $T(t)$, that is, the *target population*. The main idea underlying this proposal is to use the information that can be gathered when moving along these trajectories in solution space as additional information for the EA in order to make it more efficient. We will say that the individuals following the trajectories towards their targets are evaluating

the points along the trajectory (think of a plankton measurement unit that is moving towards a target point and is evaluating all the points it traverses in that direction). Thus, two additional populations of individuals are considered by the CS-EA:

- A population that includes all the points of the search space that have been evaluated up to that point. That is, not only the target points determined by the EA, but also those that different individuals have evaluated as they were moving towards their targets. These will be denoted as the *evaluated population* (hereafter, *E(t)*).
- A population of *evaluating entities*: these elements are the successive transformations of the individuals in *P(t)* when moving towards *T(t)* and they can be used to measure the fitness of the landscape points they go through and send the measurements to the *evaluated population*. Each *evaluating entity* ($\vec{c}$) has an associated individual from the *target population* ($\vec{t}$) that guides its transformation along a trajectory (basically, the *current individual* changes by moving towards the *target*). We must point out that the number of *evaluating entities* is independent of the *base-EA* population size.

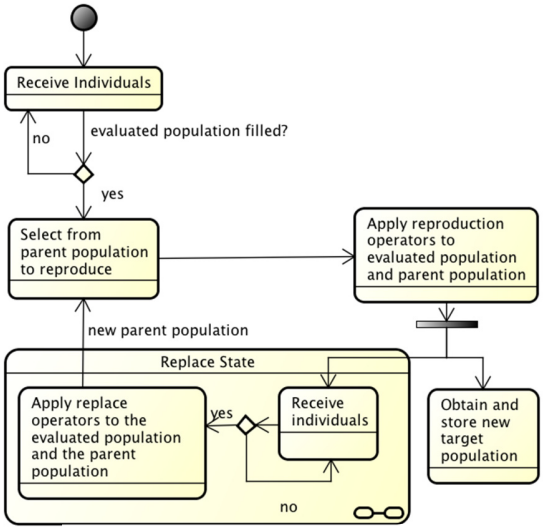

Fig. 1. Constrained Sampling EA flow chart.

In addition to these two populations, the CS-EA also implies changes with respect to regular EAs in the way the fitness landscape is covered. The following elements are necessary for this objective:

- A strategy to select the target individual assigned to each *evaluating entity*, hereafter the *target selection strategy* (τ). In other words, it is necessary to provide a way to decide how the target towards which a given *evaluating entity* moves is chosen.

$$\tau: \mathbb{R}^n \times \mathbb{R}^{m \times n} \to \mathbb{R}^n, \tau\left(\vec{c}, T(t)\right) = \vec{t}', \vec{t}' \in T(t) \qquad (1)$$

where *n* is the problem size and *m* is the target population size.

- A definition of the rules that determine the modifications allowed over the parameters of the *current individual* of the *evaluating entities*. They always adapted to the specific application and are denoted as the *application rules* (σ).

$$\sigma: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n, \sigma(\vec{c}, \vec{t}) = \overrightarrow{c'}, \left|\vec{c} - \vec{t}\right| \geq \left|\overrightarrow{c'} - \vec{t}\right| \qquad (2)$$

The only requirement in the definition of the *application rules* is the following:

$$\sigma^i(\vec{c}) = \vec{t}, i \in \mathbb{N}, i < \infty \qquad (3)$$

The execution cycle of CS-EAs is quite similar to that of regular EAs. The main difference is the inclusion of the two new populations within the structure of the algorithm.

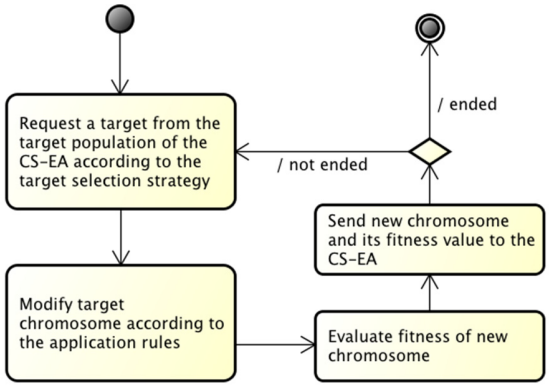

Fig. 2. Evaluating entities behavior flowchart.

First, it is usually the case that we want a minimum size for the *evaluated population* before applying any evolutionary operator. For the sake of simplicity, this size, *N*, generally corresponds to the population size parameter of the *base-EA*. Thus, the algorithm usually waits until there are, at least, *N* evaluated individual and the parent population of the *base-EA* is then filled with the first *N* individuals of the *evaluated population.* At this point, the CS-EA starts running the operators corresponding to a generation of the EA; the selection phase is executed to select the individuals from the parent population that will be reproduced and the reproduction operators are used to generate a new offspring population that will be stored as the *target population.*

After this, the *evaluating entities* start to follow a trajectory from the current parent population to the target population, providing fitness values for the points along the way and sending them to the *evaluated population*. Once the *evaluated population* has, again, at least *N evaluated individuals*, the replacement operators compare the *parent population* to the *evaluated individuals* selected from the *evaluated population,* creating the new *parent* population that leads to a new generation of the CS-EA. The flowchart of Fig. 1 represents the sequence of steps followed by the CS-EA each generation.

As previously mentioned, due to the computational cost or to physical constraints of the applications, the individuals of the *base-EA* cannot directly measure the fitness of the solutions proposed by the *target population*. The *evaluating entities* go through the fitness landscape following a trajectory guided by the individuals of the *target population*.

Each instant of time, the *current individuals* of the *evaluating entities* are modified according to the *application rules*. The *evaluating entities* measure the fitness of their new positions in search space and send them to the *evaluated population*. Note that this fitness value is not necessarily that of the target, but that of a point closer to the target than the initial value. Finally, according to the *target selection strategy*, a new target individual is assigned to the *evaluating entity*. The behavior of the *evaluating entities* is represented in the flowchart of Fig. 2.

Summarizing, the target population generated using the CS-EA operators guides the motion of the evaluating entities along the landscape towards areas that the CS-EA considers promising. However, this target population cannot be used to guide the evolutionary process due to the fact that it is usually impossible to evaluate the fitness value (sensed value) for these positions within the time interval assigned, that is, there is usually no evaluating entity close enough (either in distance or in configuration) to be able to make the measurement. However, as the evaluating entities move throughout the fitness landscape, they are evaluating positions in the direction of the target positions and these evaluations can be used as evaluated individuals in the CS-EA.

B. Particularization to Differential Evolution

As previously indicated, the evolutionary approach proposed here could be coupled to any EA. This work concentrates on the modification of a Differential Evolution (DE) algorithm [9]. It has been chosen due to its exploration capabilities and to the fact that it has shown the best average performance in the most popular algorithm competitions of the field [14, 15]. Thus, the *base-EA* in this case is Differential Evolution algorithm.

For application to constrained sampling problems, the DE must undergo a series of changes. On one hand, it must consider the two new populations mentioned above, but that is completely straightforward. On the other, a target selection strategy and a set of application rules need to be defined.

Regarding the *target selection strategy*, in this work two very different strategies were considered and they were compared in the experimental section. The first one, which we have called *random*, selects a random individual from the target population for each *evaluating entity*. The second one, which we have labeled as *closest*, chooses the individual from the target population that is closest to the current configuration (position in the fitness landscape) of each *evaluating entity*.

The *target selection strategy* is also responsible for determining when the *target* is changed. When working with standard EAs, the *target* is only changed when the *evaluating entities* reach it, as it is mandatory to measure its fitness in order to obtain the fitness of the offspring that, in standard EAs, constitute the *evaluated population*. This is not the case of the CS-EA due to the fact that the *evaluating entities* fill the evaluated population as they move along the search space and there are many more evaluated individuals than just the targets. In fact, the targets may not even be in the evaluated population, as they often cannot be reached in time.

To finish with the configuration of the algorithm, the *application rules* need to be defined. The application rules are completely problem dependent and will be different for each problem considered. A description of the ones chosen for the two types of problems considered in the experimental section will be provided there.

A basic DE scheme, with the best or random mutation strategy, is adequate for the optimization of functions that present only one global optimum. However, there are scenarios in which a CS-DE would be appropriate where the possibility of locating several optima is required. To this end a commonly used approach in the EA field is inspired by the biological concept of niching [16]. Taking advantage of this concept, the EAs preserve the diversity of the population through the formation of niches or sub-populations that explore different and spread-out promising areas of the search space. This allows the algorithm to simultaneously converge to several solutions of the function in the same run. EA niching techniques that can be found in the literature include crowding, fitness sharing, clearing, clustering, parallelization or speciation. In this work we have considered the DE variant for multimodal problems presented in [17] as our base algorithm for multiple optima problems because of its successful performance in solving the most commonly used multimodal benchmark functions. It takes advantage of the clustering behavior of the random mutation strategy and incorporates the concept of vicinity by using the information of the closest individual of the population to generate the mutated individual. More specifically, for each individual of the population $x_g^i; i = 1, 2, \dots, NP$, where *NP* is the total number of individuals in the population and *g* indicates the generation, the trial vector is generated according to the DE/nrand/2/bin scheme. Where nrand is the mutation strategy that generates the mutant vector v_{g+1}^i as follows:

$$v_{g+1}^i = x_g^{NN_i} + F\left(x_g^{r_1} - x_g^{r_2}\right) + F\left(x_g^{r_3} - x_g^{r_4}\right) \quad (4)$$

where $x_g^{NN_i}$ is the closest individual to the current one x_g^i, $r^1, r^2, r^3, r^4 \in \{1, \dots, NP\}/\{i\}$ are mutually different random integers excluding the current index *i* and *F* is a real parameter called scaling factor. The trial vector is obtained through the application of the common binomial crossover operator to the mutant and target vector as follows. For each parameter of the individuals *j*, where *j = 1…n*, the trial vector (u_{g+1}^i) is generated as follows:

$$u_j^i = \begin{cases} v_j^i \; if \; rand(0,1) < CR \; or \; j = k \\ x_j^i \; otherwise \end{cases} \quad (5)$$

where *CR* is the crossover rate and *k* is a random integer value in *[0, n)* which ensures that at least one parameter of the mutant vector is inherited.

III. EXPERIMENTS OVER BENCHMARK FUNCTIONS

In this section, we are going to compare the performance of a CS-DE algorithm to that of a standard DE over a well-known set of benchmark functions. Due to space

TABLE 1
ABSOLUTE ERROR FOR SCHWEFEL'S FUNCTION.

	Random target selection strategy			Closest target selection strategy		
	EE5	**EE10**	**EE20**	**EE5**	**EE10**	**EE20**
St – EA	1.4e-07 ± 0e+00	1.4e-07 ± 0e+00	1.4e-07 ± 0e+00	1.4e-07 ± 0e+00	1.4e-07 ± 0e+00	1.4e-07 ± 0e+00
Cs – EA	3.9e+00 ± 2.0e+01	1.4e-02 ± 7.4e-02	4.4e-04 ± 1.5e-03	1.4e-07 ± 4.5e-14	1.4e-07 ± 8.5e-12	1.4e-07 ± 1.0e-13

TABLE 2
GENOTYPIC ERROR PROVIDED FOR SCHWEFEL'S FUNCTION.

	Random target selection strategy			Closest target selection strategy		
	EE5	**EE10**	**EE5**	**EE10**	**EE5**	**EE10**
St – EA	1.3e-07 ± 4.9e-10	1.3e-07 ± 5.1e-10	1.3e-07 ± 4.3e-10	1.3e-07 ± 6.5e-10	1.3e-07 ± 7.1e-10	1.3e-07 ± 5.8e-10
Cs – EA	3.0e-02 ± 2.0e-01	1.6e-04 ± 6.6e-04	4.4e-04 ± 1.5e-03	1.3e-07 ± 4.7e-10	1.3e-07 ± 6.1e-09	1.3e-07 ± 4.9e-10

limitations, only five benchmark functions will be presented in this paper. The selected functions are the *Sphere Model (f1),* and *Ackelys's (f2), Rastrigin's (f3), Rosenbrock's (f4) and Schwefel's (f5)* functions. The first one is unimodal, being the other four functions multimodal with different optima distributions. Their analytical expressions and characteristics may be found in [15].

A. Experimental Setup

The algorithms are compared in terms of efficiency, effectiveness and precision using the speedup, the absolute error and the genotypic error [18], respectively. To ensure that the results are statistically significant, 50 independent runs of each experimental setup configuration were run. Their stop criterion, as in the most popular algorithm competitions, was based on the maximum number of function evaluations (FEs). Here this value was set to $n \cdot 10^5$, where *n* is the dimension of the problem.

To fairly compare and analyze the results, an implementation of DE with the same configuration will be used as the standard DE. In [18], the authors analyzed the performance of DE over different optimization functions including the five used in this work. The configuration of the algorithms in this paper is the same as that of [18]. The population size was set to 50 individuals and the dimensionality of the functions was set to 2 in all cases.

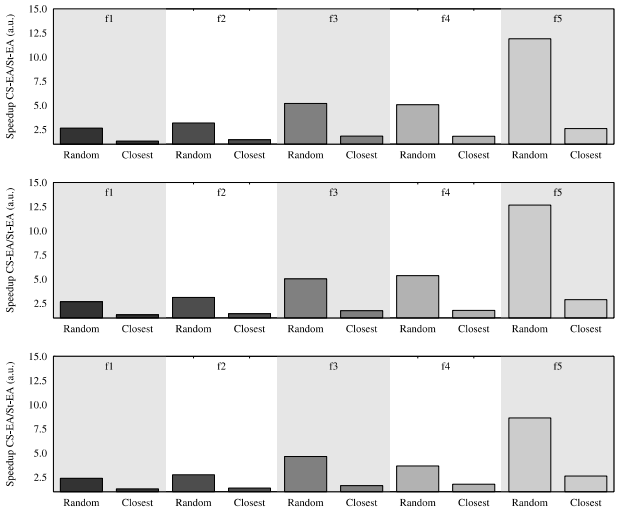

Fig. 3. Speedup comparison between the two algorithms when 5, 10 and 20 (from top to bottom) evaluating entities are used.

The CS-DE does not need to reach the targets in order to fill up the evaluated population. These targets can thus be changed even when they have not been evaluated, that is, before the evaluating entities have reached them. However, to compare the algorithms more evenly in these tests, the *targets* of the CS-DE approach will only be modified when the *evaluating entities* reach them. Notice that this decision is unfair to the CS-DE because its exploration capabilities decrease when forced to reach the targets, reducing its performance with respect to when it changes targets freely.

To analyze the influence of the number of *evaluating entities* on the performance of the two algorithms, the tests were carried out using 5, 10 and 20 *evaluating entities* for the CS-DE. Again, to incorporate the constraints imposed by CS-problems, when evaluating the standard DE we also assumed a limitation in the number of evaluating entities. This means that the offspring population was evaluated, as in the case of any CS-problem, by using the number of samplers available (5, 10 or 20) until the whole offspring population had been covered. Our initial hypothesis is that the larger the number of evaluating entities, the less time required for completing the task. Also, increasing the number of evaluating entities improves the exploration capabilities of the algorithm and, consequently, the absolute and genotypic error should decrease.

To finish with the adaptation of the algorithm to the problem, the *application rules* need to be defined. In these experiments, we simulate a target-finding problem where the target is the optimum of the benchmark function. Thus, the *evaluating entities* will be moved throughout the search space like physical samplers with the same physical constraints, i.e., their movements are limited by a maximum step length, here set to 0.01 units, which represents 0.35% of the search space.

B. Results

After executing the experiments described above, the first thing that must be mentioned is that all the runs were solved successfully. They all produced absolute errors below *1.0e-6*, except for some of the runs over Schwefel's function. In these cases, even though the random target selection strategy led to higher absolute errors (see Table 1) the results in terms of distance to the optimum (Genotypic error in Table 2) show that the differences are not significant.

Finally, the three graphs of Fig. 3 provide an indication of the speed-up obtained over the standard DE algorithm when using the CS-DE, both for a *random* selection strategy and for a *closest* selection strategy. From top to bottom, the plots display the results when 5, 10 and 20 evaluating entities were used. The CS-DE is always superior to the standard DE in terms of efficiency, significantly speeding up the search. As expected, as the number of evaluating entities increases there is less difference between the two models.

Clearly, the random selection strategy leads to the most significant differences between the standard DE and the

CS-DE. The latter is 2.5 times faster in the worst case and 12 times faster in the best one. This difference is more evident in multimodal functions (f2-f5), which require a more exploratory behavior from the DE algorithm. In fact, among the multimodal functions, f2, f3 and f4 present a topographical structure that requires less explorative capacities than function f5 (see [18] for more details). This is the reason why the difference between the CS-DE and the standard DE is larger in the case of function f5. In this case the CS-DE approach explores the solution space in a short span of time, as, unlike the standard DE, it takes advantage from the information it gathers during the trajectories.

IV. Real World Test

A more realistic example of the application of a CS-DE is considered in this section. The task that we contemplate is that of searching for a number of shipwreck survivors floating in the sea wearing life vests with short-range beacons. This search will be carried out by a group of autonomous Unmanned Aerial Vehicles (UAVs) led in their search by a CS-DE, in this case a CS-DE with niching as we need to find all of the survivors simultaneously.

A. Experimental Setup

As the objective here is to test the behavior of the CS-DE extensively and in order to avoid the high costs of performing real test flights for each experiment, it was necessary to create a simulation environment where the algorithms could be evaluated. The parameters of the UAV models used within this simulator were obtained from flight tests of real UAVs with a wingspan of 1660 mm, a length of 1200 mm, and a 40 km range. The dynamic behavior for each plane was simulated using the FlightGear Flight simulator. FlightGear interacts with a Java program, which implements the Virtual Aircraft agent and runs the behavior of the VA and the Autopilot software.

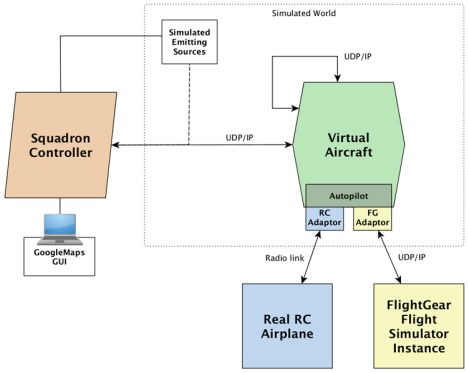

Fig.4 UAV team simulator block diagram.

The simulation environment (Fig. 4) uses a central control point, the SC, for monitoring and visualization purposes (using the Google Maps API), and for global squadron control in some algorithms. The SC communicates remotely, via UDP/IP, with the Virtual Aircraft (VA) software, which can be run in remote computers, so that a computation cluster can be used to run multiple planes. This remote operation capability allows running a VA directly in the flight field, connected via radio control to a real plane and remotely connected to the SC via Internet.

B. Tests and Results

As indicated at the beginning of the section, the task considered in the experiments was that of finding different numbers of shipwreck survivors in a large area of the sea. This task is quite costly through conventional means and is very adequate for UAV teams. In this case, as we were just studying the capabilities of the algorithms, it was assumed, as it is commonly the case, that all these shipwrecked people were wearing a life vest provided with a low power radio beacon. Thus, the search is based on the detection of the emergency signals given off by these beacons, which, for our purposes and to make the problem more general and applicable to other tasks, such as environmental monitoring, we assumed were not directional.

The tests have consisted in searching for different numbers of shipwrecked people within a 100 km^2 area of the sea. To compare to other more common strategies, we have used two different algorithms: the CS-DE and a Swarm Intelligence based approach and different sizes of UAV squadrons.

Two different versions of the virtual emitting source function have been used. One without noise, that could, in principle, be easily solved using gradient based algorithms, and one with noise (the noise levels where modeled after the real data extracted from the real planes) that would theoretically be more difficult for gradient based algorithms.

In order to compare the algorithms in terms of time taken to accomplish the task of finding N emitting sources, we are assuming that planes can "see" the targets when they are within a radius of 100 meters from them. The algorithms were run until all the emitting sources in the area were visually confirmed by, at least, two planes. This was done to minimize the influence of random successes due to planes that visually confirm sources during the exploration stage.

The configuration parameters for the DE used are those recommended in [17]. Specifically, we have used a value of 0.5 for the F parameter, 0.9 for the crossover rate (CR) and a population of 100 individuals. The target selection strategy used was *random* within the detection range of the planes. This detection range is taken as infinite in distance but limited to an angle of 45 degrees in their heading direction. In the case of the swarm approach, the configuration parameters were a decision gap (T) of one minute and a number of closest neighbors taken into account in the selection of the heading direction (N) of 3.

For each test, the times required to find each existing emitting source were recorded. Not all the combinations of squadron sizes and emitting source numbers were solved by both algorithms. Finally, the search was temporally limited to a maximum of sixty minutes. More time than that would exceed the flight autonomy of the real planes and in real conditions in the Atlantic, the average survival time is usually no more than 50 minutes due to hypothermia.

As an example of the test runs, Fig. 5 displays a sequence

of snapshots from the simulator running the CS-DE algorithm with five planes and two shipwrecked people. The planes are shown as arrows, with their trajectories drawn as colored lines. The stars mark the target population of geographical points generated by the CS-DE algorithm in that generation. Initially (a) the algorithm generates a highly dispersed population. As the planes start exploring the space, the target population of the algorithm concentrates around the areas where the agents have sensed the most promising values (b and c). As the simulation progresses, the algorithm starts to exploit the already collected information by concentrating the target population near the emitting beacons of the shipwrecked people (d and e). As the planes tend to move towards the target population, the concentration of the target population obtained by the evolutionary algorithm over the areas where the shipwrecked people are located end up leading the planes to fly over these areas (f).

Fig. 6 displays the results of the performance of the CS-DE compared to that of the swarm approach. Obviously, as the number of shipwrecked people increases the task becomes harder in all cases. Take into account that the area being explored corresponds to 100 Km^2 of sea and the number of planes is quite small (15 at most). For all three squadron sizes, as the number of objectives increases, the time it takes the CS-DE to find them increases on average. In the case of the swarm based approach this is more evident, as when the tasks involve two or three shipwrecked people the number of successful runs decreases dramatically, especially in the case of the smaller squadron (5 planes). In fact, for the smaller squadron, the swarm-based approach is never successful when there are more than one shipwrecked person. This is because, with such a small number of units, it is quite difficult for the algorithm to split the units into teams. On the other hand, when the swarm-based approach is successful in this case it can be marginally faster than the CS-DE due to its higher exploitative capabilities.

The CS-DE, on the other hand, seems more balanced in its exploitation vs. exploration capabilities. This, obviously, induces a slight disadvantage in terms of speed when there is just one shipwrecked person, but as the graphs show, when the number of targets increases it is clearly advantageous, dramatically reducing the chance of missing one.

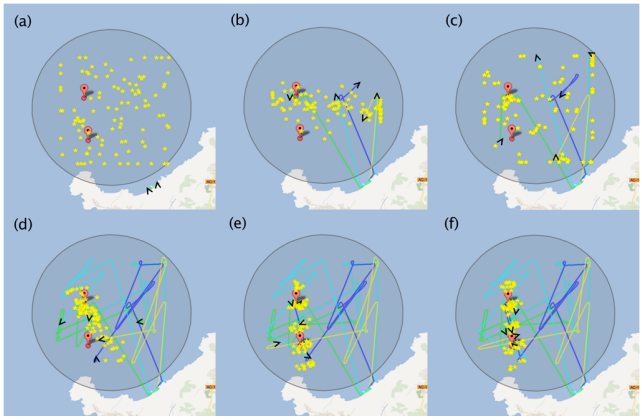

Fig. 5. Sequence of snapshots from the simulator while running a simulation using the CS-DE algorithm, 2 shipwrecked people (circles) and 5 airplanes (arrows). The stars represent the target population provided by the algorithm in a given generation and the lines the trajectories followed by the airplanes.

One can improve the results of the swarm-based approach by increasing the number of planes in the team (bottom left graph). That is, allowing more planes per unit area. However, the CS-DE still produces a higher number of successful runs, solving within the allotted time every run with 2-shipwrecked people and all of the runs but two with 3-shipwrecked people. Additionally, as the number of planes increases the CS-DE becomes as fast as and sometimes even faster than the swarm-based approach in all cases.

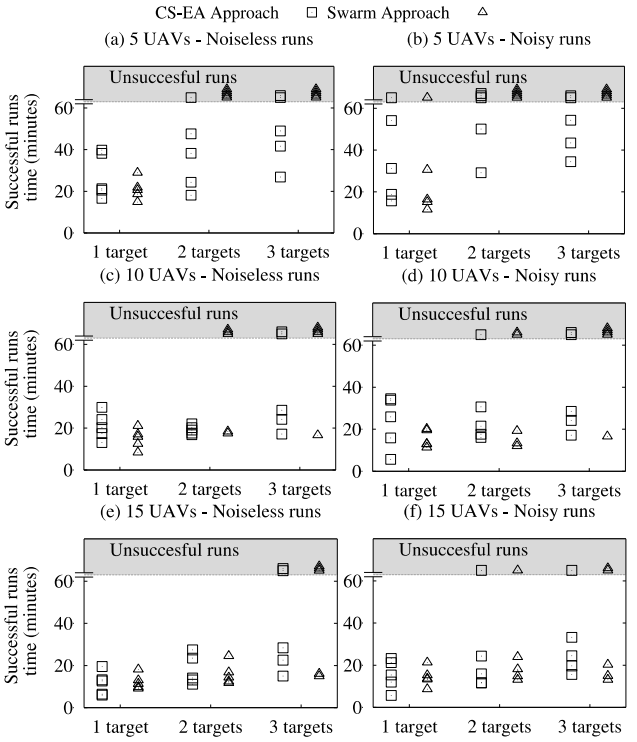

Fig. 6. Results provided by the two approaches in the different test scenarios. From left to right, each graph shows the results for different types of runs: those with no noise in the electromagnetic signal are shown in the left column while those with noise are shown in the right column. From top to bottom, results obtained by teams of 5, 10 and 15 UAVs. Triangles correspond to the Swarm based algorithm and squares to the CS-DE.

The same conclusions hold when analyzing the results of the cases where the electromagnetic signal was noisy (right graphs). Although the average time required to accomplish the tests is slightly longer for both algorithms than in the noiseless scenarios, the reliability of the CS-DE approach remains much better.

Summarizing, although the swarm based approach may sometimes be faster in finding one individual than the CS-DE, the latter is much more reliable and robust in all cases. In addition, as the number of planes increases the time needed to find all the shipwrecked people becomes similar for both approaches when successful. This is because the search process of an evolutionary algorithm exploits the information of the search space provided by the planes more effectively.

V. CONCLUSIONS

This paper proposes a new kind of Differential Evolution algorithm in order to address a niche of problems where EAs have seldom been applied due to their poor performance. These problems have been called Constrained Sampling problems and are characterized by a bottleneck in the process of obtaining the individuals needed in order to evaluate the offspring each generation of an evolutionary process.

We have shown that by appropriately using the information obtained during the decoding process, that is, during the process of positioning the individuals in the locations of the fitness landscape indicated by the evolutionary algorithm when producing offspring, the search could be greatly accelerated. To this end, a series of modifications over the standard DE are proposed allowing it to work over this type of problems. The new algorithm has been called the Constrained Sampling DE algorithm (CS-DE).

The performance of the CS-DE has been compared to that of a standard DE in a set of benchmark constrained sampling problems. These problems are well known benchmark functions where a constraint has been imposed on the maximum size of the change of an individual each time step. The results have shown that the CS-DE outperforms the standard DE in all cases, leading to speedups of up to 12 depending on the target selection strategy and number of evaluating units.

Finally, to show the applicability of this strategy to real problems, the CS-DE has been applied to the coordination of a UAV team when searching for shipwreck survivors in the open sea and compared to a swarm based strategy (which is a more popular strategy in this case). The results show that the CS-DE is much more robust, especially when multiple targets are present and much more balanced with regards to exploration and exploitation.

Current activities involve characterizing the scalability of the CS-DE to higher dimensionalities. In this line, some preliminary experiments have shown that in a 10 dimensional problem, the speedup of the CS-DE with respect to a standard DE can reach values of up to 2000. We are also working on the reformulation of other types of EAs to a CS version as well as on their application to real world problems.

REFERENCES

[1] Ong, Y. S., Nair, P. B., & Keane, A. J. (2003). Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA journal, 41(4), 687-696.

[2] Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. Soft computing, 9(1), 3-12.

[3] Piccolboni, A., & Mauri, G. (1998, January). Application of evolutionary algorithms to protein folding prediction. In Artificial Evolution (pp. 123-135). Springer Berlin Heidelberg.

[4] Schneider, G. (2000). Neural networks are useful tools for drug design. Neural Networks, 13(1), 15-16.

[5] Y. Altshuler, V. Yanovsky, I. a. Wagner, and A. M. Bruckstein, "Efficient cooperative search of smart targets using UAV Swarms," Robotica, vol. 26, no. 04, Feb. 2008.

[6] Faiña, A., Orjales, F., Bellas, F., & Duro, R.J., (2011) First Steps towards a Heterogeneous Modular Robotic Architecture for Intelligent Industrial Operation, Workshop Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo-Hybrid Systems, IROS, 0-6.

[7] Singh, S. P., & Sharma, R. R. K. (2006). A review of different approaches to the facility layout problems. The International Journal of Advanced Manufacturing Technology, 30(5-6), 425-433.

[8] Hansen, Nikolaus, and Andreas Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation." In Evolutionary Computation, Proceedings of IEEE International Conference on, pp. 312-317. IEEE, 1996.

[9] Storn, Rainer, and Kenneth Price. "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces." Journal of global optimization 11, no. 4, 341-359, 1997.

[10] Banks, A., Vincent, J. and Phalp, K. T.,"Particle Swarm Guidance System for Autonomous, Unmanned Aerial Vehicles in an Air Defence Role". Journal of the Royal Institute of Navigation, 61, pp. 9-29, 2008.

[11] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks* Vol. IV: 1942–1948.

[12] E. Besada-Portas, L. de la Torre, J. M. de la Cruz, and B. de Andres-Toro, "Evolutionary Trajectory Planner for Multiple UAVs in Realistic Scenarios" IEEE Trans. Robot., vol. 26, no. 4, pp. 619–634, Aug. 2010.

[13] S. Mittal and K. Deb, "Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms," in Proc. IEEE Congr. Evol. Comput., vol. 7, pp. 3195–3202, 2007.

[14] Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello, C. C., & Deb, K. (2006). Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Journal of Applied Mechanics, 41.

[15] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL Report 2005.

[16] Gulshan Singh and Kalyanmoy Deb, Dr. "Comparison of multi-modal optimization algorithms based on evolutionary algorithms." In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06). ACM, New York, NY, USA, 1305-1312, 2006.

[17] Epitropakis, M. G., Plagianakos, V. P., and Vrahatis, M. N. "Finding multiple global optima exploiting differential evolution's niching capability". In Differential Evolution (SDE), IEEE Symposium on (pp. 1-8). IEEE, 2011.

[18] Caamaño, P., Bellas, F., Becerra, J. A., & Duro, R. J. (2013). Evolutionary algorithm characterization in real parameter optimization problems. Applied Soft Computing, 13(4), 1902-1921.