Abstract

When dealing with metaheuristics, one important question is how many evaluations are worth spending in the search for better results. This work proposes a method to estimate the best moment to stop swarm iterations based on the analysis of the convergence behavior presented during optimization, aiming to provide an effective balance between saving fitness evaluations and keeping the optimization quality. An automated Convergence Stabilization Modeling operating in Online mode (CSMOn) is proposed based on a sequence of linear regressions using exponential and log-like curves. The method was tested on the CEC13 benchmark with CCPSO2-IP E algorithm and on 30 random Max-Set functions with the swarm algorithms PSO, ABC and CCPSO2-IP E. CEC13 results show that up to 90% less fitness evaluations are performed for functions where CCPSO2-IP E has a steady convergence, and up to 49% for functions where convergence is erratic, while penalties for fitness are kept small. Max-Set results demonstrates the robustness of CSMOn for the search algorithms tested. We conclude that CSMOn is capable of adapting to an optimization in progress, producing a good trade-off between result quality and evaluation savings.

Index terms— Convergence stabilization modeling, parameter tuning, curve fitting, online algorithms, swarm intelligence.

http://dx.doi.org/10.1145/3071178.3071326

Estimating Stop Conditions of Swarm Based Stochastic Metaheuristic Algorithms

Peter Frank Perroni, Daniel Weingaertner, Myriam Regattieri Delgado[*][†][‡]

July 20, 2017

1 Introduction

The increasing complexity of real world optimization problems requires powerful tools with robust configuration sets. Among these tools, stochastic metaheuristics play a special role, and there is a wide variety of literature concerning configuration methods and usage. However, despite their importance, not much effort has been done toward algorithm-independent procedures to determine a good cost/benefit number of fitness function evaluations, specially for continuous problems. Apart from being quite contextual (the solution relies on the strengths and weaknesses of the optimization algorithms and the problem being optimized), their behavior also obeys the unpredictable random numbers logic.

Traditionally, convergence modeling is directed toward asymptotic analysis, severely restricted to algorithm properties [1] where rigorous assumptions must be made, and mostly looking for global best solution under a theoretical infinite number of evaluations [2]. This has a limited practical value since real world complex optimizations normally expect high quality solutions (local instead of global ones) with minimal number of evaluations (instead of accepting large processing times). Some works like [3] ally asymptotic analysis with practical usage, but still under hard restrictions and assumptions and mostly for discrete problems with no large search spaces. In [4], a general framework using model convergence is proposed for discrete problems, which instead of using the best solutions only for overall convergence analysis, it uses the population history to interfere in new generations for better quality

[*]P. F. Perroni is with the Department of Informatics, Federal University of Paraná, Curitiba, PR (email: pfperroni@gmail.com).

[†]D. Weingaertner is with the Department of Informatics, Federal University of Paraná, Curitiba, PR (email: danielw@inf.ufpr.br).

[‡]M. R. Delgado is with the Department of Electrical Engineering, Federal University of Technology, Curitiba, PR (email: myriamdelg@utfpr.edu.br).

of solutions. In [5], a deep analysis is made of which conditions and parameters selection guarantee local convergence for Particle Swarm Optimization (PSO), but no novel process is provided for convergence modeling. Model-based optimization (MBO) methods [6] use either distribution-based models or surrogate models to improve convergence. The former includes the well-known Estimation of Distribution Algorithms (EDAs), which generate new candidate solutions according to the probability distribution estimated from the promising candidate solutions of the current population. In the latter, generally used when the evaluation of the cost function is very expensive or time consuming, the original function is replaced by a cheaper coarse-grained function that simplifies the search space, however its performance on high dimensionality problems and the selection of the correct surrogate model are still open research questions to be addressed. A more elaborated approach that also requires specific mathematical formulation for every search algorithm for a good performance is the Optimal Computing Budget Allocation (OCBA) [7], which operates by iteratively allocating function evaluations to individuals (until the total budget is exhausted) based on an allocation rule that identifies the most promising solutions, so that noisy/costly solutions are discarded and accuracy is improved, thus maximizing the convergence rate [8] by estimating the fitness.

This work proposes a different approach for local convergence analysis that is not dependent on a particular algorithm but on the standard behavior presented by general memory-based metaheuristics (like swarm algorithms). It treats the optimization algorithm as a black-box, thus exploring several aspects present in the search algorithm instead of requiring core changes to it. The main goal is to provide an on-line estimation of a good cost/benefit stopping point for every optimization run of a swarm based search algorithm, aiming to save function evaluations.

The hypothesis is that the inherent convergence behavior presented by these search algorithms when entering a local minimum can be modeled by a sequence of auto-adapted exponential and log-like curves.

The reason behind this hypothesis is that the social knowledge provided by the swarms gradually restrict the movement of each individual within a portion of the search area (given that the algorithm guarantees that individuals will reach a stable point [5]). Such movement toward the stable point (and potentially toward a local minimum) yields reasonable exponential convergence rates (when major improvements occur) followed by log-like convergence rates (when improvements slow down), providing run time data to model the convergence speed through linear regression.

When approaching a local minimum, due to the typical large sequence of small improvements in this region of the search space, the curve calculated through linear regression will be biased toward recent data points and away from initial (fast convergence) points, indicating the beginning of the convergence slow down. When the slow down happens during an exponential convergence, the majority of the improvements already occurred and next improvements tend to be modest. When the slow down happens during a log-like convergence, only minor incremental improvements are expected on subsequent evaluations, meaning that the convergence is stabilizing and further search might not be advantageous.

Considering this hypothesis, and given a restricted computational budget (limited swarm iterations, fitness evaluation or run time) on a multiple run scenario, CSMOn can automatically determine a good moment to stop evolving a population of solutions on a particular run and start searching on the next run. In this case, we consider that more important than the budget allocated to the fitness calculation of a specific particle, the challenge here is associated with the analysis of the convergence of the best fitness achieved by the swarm. This way, a good commitment could be found between the drawback (performance loss) of not having optimized until the end of the budget allocated for each independent run, and the advantages of saving computational budget (notice that using the saved effort on subsequent optimization runs is out of the scope of this work).

Therefore, the proposed method Convergence Stabilization Modeling in Online mode (CSMOn) consists in detecting local minimum traps of evolution by collecting convergence data (points comprising best fitness values and their respective number of evaluations) during the optimization run (online). A sequence of 2D exponential and log-like curves is adjusted over this data through linear regression, and a good cost/benefit stopping point is estimated by finding out the moment when the log-like curve takes distance from the initial convergence improvements. The source code of the CSMOn implementation is available online at [9].

2 Mathematical Basis

Taking the convergence data as a list $\ell = \{(x_j, y_j)\}, \forall j \in \{1, \cdots, s\}$ of size s of strictly monotonically decreasing 2D coordinate points, y_j as the jth fitness value obtained at fitness evaluation x_j, p_1 as an initial position in this list, p_2 as a final position and p_c as a position close to p_2 ($1 \leq p1 < p_c \lessapprox p2 \leq s$), one way of defining a convergence stabilization is observing the relative differences of the convergence improvements. When the difference $[y_{p_c}, y_{p_2}]$ is very small compared to the difference for the whole interval $[y_{p_1}, y_{p_2}]$, we can observe a convergence stabilization. In such definition, the stabilization is contextual and depends on how many convergence improvements have been obtained and the relationship between them. Hence, the greater the number of improvements at beginning of the sequence and the difference between them, more improvements with small differences will be required to state that a convergence is reaching a stabilization point at a later stage.

This section describes the curves, the linear regression method and the properties used in this work to model the convergence stabilization behavior.

2.1 Curves

Two curve functions are used to model the general convergence stabilization behavior of a swarm based stochastic metaheuristic: the Exponential and the Power functions. Large improvements typically seen at the initial of the search or when escaping from local minimum traps are better modeled by exponential curves, whereas log-like curves are well suited for moderate and small improvements seen on the remaining of the search.

The natural Exponential Function Exp is defined as:

$$Exp(x) = \alpha e^{-\beta x} \tag{1}$$

where α is the Intersect point with Y axis ($\alpha = Exp(0)$) and β is the relative growth speed of the function.

The deterioration of an exponential convergence can be detected by discovering the point when the sublinear convergence begins. The list ℓ is said to converge sublinearly when:

$$\lim_{s\to\infty} \frac{|y_s - L|}{|y_{s-1} - L|} \to 1 \tag{2}$$

where L denotes the last possible y value of the sequence. Assuming the last possible value as L_M for a maximum number of M evaluations, this criteria can be re-written for a minimization problem by an approximation as the decay equation (3):

$$\mathcal{D}_e(\ell) = 1 - \frac{y_s - L_M}{y_{s-1} - L_M} \tag{3}$$

indicating how much the current value y_s differs from the previous value y_{s-1} concerning the estimation of the limit achieved due to computational limitation.

The log-like Power Function Pow can be defined as

$$Pow(x) = \alpha x^{-\beta} \tag{4}$$

where α is the Scaling factor ($\alpha = Pow(1)$) and β is the growth speed of the function.

A logarithmic convergence commences only after the deterioration of the exponential convergence. Thus, once the criteria (2) is satisfied, the list ℓ is said to converge logarithmically when:

$$\lim_{s\to\infty} \frac{|y_s - y_{s-1}|}{|y_{s-1} - y_{s-2}|} \to 1 \tag{5}$$

As in (3), this limit can be simplified by decay equation (6):

$$\mathcal{D}_l(\ell) = 1 - \frac{y_s - y_{s-1}}{y_{s-1} - y_{s-2}} \tag{6}$$

2.2 Linear Regression

Least Square (LS) is a linear regression method used to find the best-fitting curve that adjusts to a data set ℓ of points (x, y).

Due to the rigid shape of the exponential curve, the precision of the curve calculated through LS is increased if the data set is translated to bring the convergence values $\mathbf{y}$ closer to the X axis. Let $\gamma_j = y_j - min(\mathbf{y}) + 1$, $\forall j \in [p_1 \cdots p_2]$ be the new translated value, then equations (7a) and (7b) can be used to estimate the parameters β and α of the exponential curve [10] between $[p_1, p_2]$:

$$\beta_e(\ell, p_1, p_2) = \frac{\sum_{j=p_1}^{p_2} (x_j - \overline{x})(ln(\gamma_j) - \overline{ln(\gamma)})}{\sum_{j=p_1}^{p_2} (x_j - \overline{x})^2} \tag{7a}$$

$$\alpha_e(\ell, p_1, p_2) = \left(\sum_{j=p_1}^{p_2} ln(\gamma_j) - \beta_e \sum_{j=p_1}^{p_2} x_j \right) / \, (p_2 - p_1 + 1) \tag{7b}$$

where $\overline{x}$ is the average of x_j and $\overline{ln(\gamma)}$ is the average of $ln(\gamma_j)$.

The exponential curve calculated through LS tends to adjust better to the regions where the majority of points are located, quickly moving away from the other points. Considering that in a typical convergence there are more moderate and small improvements than initial major improvements, there are fewer data points available to represent the beginning of the exponential decay than the remaining of the convergence. Therefore, considering that the exponential curve (1) decreases at a rate proportional to its current value, it can be said that the beginning of the convergence slow down occurs when the parameter α_e (intersection with Y axis) has its value reduced, when compared to the previous regression, after adding one more point to the sequence.

The second part of the convergence stabilization modeling uses a power curve whose parameters β and α can be estimated as follows:

$$\beta_p(\ell, p_1, p_2) = \frac{\sum_{j=p_1}^{p_2} (log(x_j) - \overline{log(x)})(log(y_j) - \overline{log(y)})}{\sum_{j=p_1}^{p_2} (log(x_j) - \overline{log(x)})^2} \tag{8a}$$

$$\alpha_p(\ell, p_1, p_2) = \left(\sum_{j=p_1}^{p_2} log(y_j) - \beta_p \sum_{j=p_1}^{p_2} log(x_j) \right) / \, (p_2 - p_1 + 1) \tag{8b}$$

where $\overline{log(x)}$ and $\overline{log(y)}$ are the average of $log(x_j)$ and $log(y_j)$.

Because the growth speed β_p is constant along X axis, the power curve has a behavior similar to the exponential curve on the intersect (parameter α_p) when distantiating from initial data points (since the majority of the points are moderate and small improvements), but occurring more slowly than for the exponential curve (given the growth speed is constant here).

3 CSMOn: Convergence Stabilization Modeling Operating in Online Mode

The proposed method (CSMOn) controls the execution of the optimization algorithm by continuously requesting the metaheuristic to process until the next best result is found. Every new best fitness value (fit) and its associated number of evaluations (ne) are appended to a list of points $\mathbf{gb} = \{(ne_j, fit_j)\}, \forall j \in \{1, \cdots, s\}$, replacing ℓ as the convergence list for the LS (section 2.2). The process is divided in three main phases (Figure 1):

i Initial/Major Improvements: Determine through criteria (2) and (5) the moment when major improvements cease, ensuring the search will stop only after first improvements have occurred. Positions 1 and p_b identify this phase (red area in Figure 1) where no regression is performed.

ii Intermediate Improvements: After phase i, the regression process is started and finds the point where the exponential curve (blue) takes distance from initial convergence data (first red circle). Positions p_b and p_T identify this phase (orange area), with position p_T defining the transition from exponential to logarithmic decay.

iii Final/Convergent Improvements: Once phase ii finishes, a log-like curve (green curve) is adjusted to the remaining improvements until the curve departs from initial points (second red circle). Positions p_T and p_S identify this phase (yellow area), with p_S defining the point where the logarithmic sequence stabilizes (third red circle), saving unnecessary fitness evaluations (grayed area).

Considering the search algorithm can escape from a particular stagnation point, phases i, ii and iii can repeat along evolution.

Metaheuristics like PSO typically show the greatest improvements at the very beginning of the search, and initial fitness values can be quite distant from the general fitness found on the remaining of the search. In such cases, the initial points (two points would suffice) may be considered outliers and are not included in $\mathbf{gb}$.

The following plateau-curve is proposed to model the general convergence stabilization behavior of swarm based optimization algorithms (with

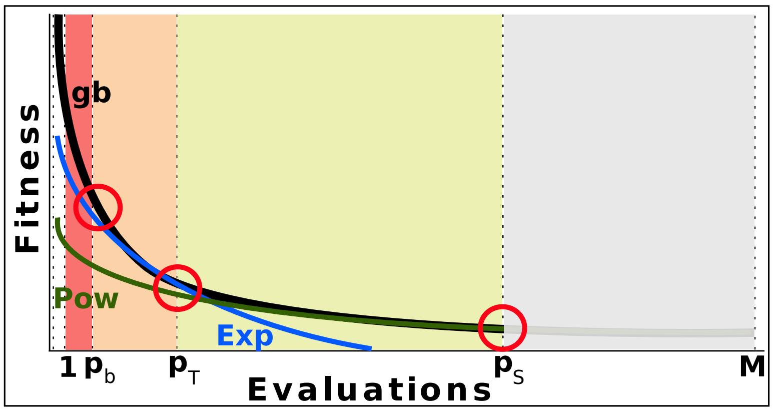


Figure 1: CSMOn Method defining transition points for fast, intermediate and slow convergence
log-like curve represented by the power curve):

$$F_{app}(ne_j) = \begin{cases} Exp(ne_j) \;\; (1) & \text{if } j \leq p_T \\ Pow(ne_j) \;\; (4) & \text{if } j > p_T \end{cases} \tag{9}$$

where F_{app} is the function that approximates the real fitness fit_j at ne_j evaluations.

Algorithm 1 CSMOn Algorithm

```
1: Input: $\{M, R\}$
2: $p_T \leftarrow -1, p_S \leftarrow -1$
3: $r \leftarrow 0.99$                                  ▷ Set $r \leftarrow R$ for a fixed relaxation
4: $append(\mathbf{gb}, GetBest(1, M))$
5: **repeat**
6:     $r \leftarrow max(r^2, R)$                      ▷ Remove this line for a fixed relaxation
7:     **if** $p_S = -1$ **then**                      ▷ Look for Exponential convergence
8:         $p_T \leftarrow AdjustExp(\mathbf{gb}, M, r)$
9:     **if** $p_T > 0$ **then**                       ▷ Log-like convergence started
10:        $p_S \leftarrow AdjustLog(\mathbf{gb}, M, r, p_T)$    ▷ At this point, either finishes for a given $r$ ($p_S > 0$) or a new exponential convergence is detected ($p_S = -1$) and returns to line 5
11: **until** $ne_{p_S} >= M$ **or** ($r = R$ **and** $p_S > 0$)
```

These are the basic CSMOn steps (Algorithm 1):

- Two problem-dependent input parameters are required: the maximum number of fitness function evaluations for every run M (i.e. the portion of the total budget for this run) and the relaxation parameter $R \in]0, 1[$. The relaxation controls the limit sensibility, specifying how close the equations (3) and (6) should be from satisfying the respective criteria (2) and (5).

- Function $GetBest(nBest, M)$ returns from the search algorithm a list $\{(ne_1, fit_1), \cdots\ (ne_{nBest},\ fit_{nBest})\}$. To deal with search algorithms that do not show a minimum initial convergence, if **gb** is empty, the function will start to add elements to the list only after $\mathcal{D}_e(\mathbf{gb}) \geq .0001$ (0.01% improvement). If the maximum number of evaluations M is reached before filling the list, the search algorithm stops and the function appends to the list a dummy entry containing the last best fitness (M, fit_M).

- Initially one best value is obtained from the search algorithm and the initial value for the relaxation is defined: $r \in [R, R_s]$, with $R > 0$ and $R_s < 1$. If a fixed relaxation should be used, then $r = R$, otherwise it will decrease in a quadratic way from the superior to the inferior limit. A decreasing relaxation accepts an initially less compliant exponential and an increasingly more suitable log-like convergence, while fixed relaxation uses the same compliance level throughout the search.

- Function $AdjustExp(\mathbf{gb}, M, r)$ requests new elements for **gb** to detect the end of the major improvements (p_b) and the transition point p_T between an exponential and a log-like convergence.

- Function $AdjustLog(\mathbf{gb}, M, r, p_T)$ requests new elements until the log-like convergence stabilizes at point p_S, indicating a possible end of the search. However, if a new exponential convergence occurs, control is returned to *AdjustExp* (line 8 in Algorithm 2) and p_b and p_T are reset.

- If a fixed relaxation is used, the method finishes at the first p_S found. If a quadratic decreasing relaxation is used, p_S is reset, r has its value reduced for the calculation of a new p_S and the method finishes when the relaxation reaches R. In any case, the method stops when the maximum number of evaluations M is reached.

Algorithm 2 AdjustExp

```
1: Input: ${\mathbf{gb}, M, r}$
2: $s_{prev} \leftarrow s$
3: $append(\mathbf{gb}, GetBest(2, M))$                ▷ Request next 2 bests results
4: if $s - s_{prev} < 2$ then return $-1$
5: $p_b \leftarrow -1$
6: while $ne_s < M$ do
7:     if $\mathcal{D}_e(\mathbf{gb}) < r$ and $\mathcal{D}_l(\mathbf{gb}) < r$ then
8:         if $p_b = -1$ then                ▷ Initialize the regression process
9:             $p_b \leftarrow s - 2$
10:            $\alpha_2 \leftarrow \alpha_e(\mathbf{gb}, p_b, s)$ (7b)
11:        else                ▷ Check the incremental difference on regression
12:            $\alpha_1 \leftarrow \alpha_2$
13:            $\alpha_2 \leftarrow \alpha_e(\mathbf{gb}, p_b, s)$
14:            if $\alpha_2 < \alpha_1$ then
15:                return $s$                ▷ Transition to log-like curve
16:    else                ▷ Stop the regression
17:        $p_b \leftarrow -1$
18:    $append(\mathbf{gb}, GetBest(1, M))$                ▷ Request next best result
   return $-1$
```

The adjustment of the exponential curve (Algorithm 2) has the following steps:

1. A sequence of best values is requested and appended to **gb** as long as the fitness improvements are substantial (i.e. until criteria (2) and (5) are satisfied) (line 7). Then, the start of the weak part of the exponential decay is set $p_b = s - 2$ (line 9), marking the beginning of the regression calculation (7b) (line 10).

2. For every next best value (line 18), a new regression is calculated in the interval $[p_b, s]$ (line 13) and the incremental difference to previous regression is compared to detect if the curve is biased toward recent best values, representing a slow down on the exponential convergence (line 14).

3. When the bias is detected, the end of the exponential curve is set $p_T = s$ (returns the control to line 8 in Algorithm 1).

4. However, if the convergence speed increases above the defined relaxation r, the regression stops (line 17) and the process proceeds requesting the sequence of best values.

Algorithm 3 AdjustLog

```
1: Input: ${\mathbf{gb}, M, r, p_T}$
2: $s_{prev} \leftarrow s$
3: $append(\mathbf{gb}, GetBest(3, M))$
4: **if** $s - s_{prev} < 3$ **then return** $-1$
5: $\alpha_1 \leftarrow \alpha_p(\mathbf{gb}, p_T, s-1)$ (8b)
6: $\alpha_2 \leftarrow \alpha_p(\mathbf{gb}, p_T, s)$
7: **while** $\alpha_2 \geq \alpha_1$ and $ne_s < M$ **do**        ▷ Check incremental difference
8:     **if** $\mathcal{D}_e(\mathbf{gb}) \geq r$ **or** $\mathcal{D}_l(\mathbf{gb}) \geq r$ **then**
9:         **return** $-1$                    ▷ Exponential convergence detected
10:    $append(\mathbf{gb}, GetBest(1, M))$
11:    $\alpha_1 \leftarrow \alpha_2$
12:    $\alpha_2 \leftarrow \alpha_p(\mathbf{gb}, p_T, s)$
   **return** $s$
```

Finnally, log-like curves are adjusted (Algorithm 3):

1. A sequence of best values is requested and the respective incremental regressions (8b) are calculated for the interval $[p_T, s]$ (lines 5, 6 and 12) and then compared to detect if the curve is biased toward recent best values, representing a logarithmic stabilization (line 7).

2. When the bias is detected, the respective position $p_S = s$ is returned, marking the end of the relevant logarithmic decay (line 10 in Algorithm 1).

3. However, if the search algorithm escapes logarithmic convergence moving towards an exponential convergence (line 8), then the whole process starts over from current position s, ignoring previous convergence points.

In a multiple run scenario, the total budget has to be partitioned in a way that every independent run has its own portion of the evaluations. A simplistic schema to use CSMOn under this scenario would set M to the number of evaluations assigned to every run, and the saved evaluations be used on upcoming runs by creating a different update mechanism for relaxation r that takes the overall remaining budget into consideration.

4 Experimental Results

Two test sets are used to experiment with CSMOn: one fixed and one randomly generated. The fixed test set comprises specially difficult competition benchmark functions for optimization called CEC13 [11] and is composed of 15 minimization functions of 1000 dimensions with global best at $L_M = 0$:

- Fully-separable Functions: F1, F2, F3.
- Partially Additively Separable Functions: F4, F5, F6, F7, F8, F9, F10, F11.
- Overlapping Functions: F12, F13, F14.
- Non-separable Function: F15.

The randomized test set uses an automated and parameterized test-problem (landscape) generator called Max-Set of Gaussians (MSG) [12, 13], which generates nonlinear, nonseparable random test functions based on the sum of Gaussians.

A competitive decomposition method [14] for high dimensional problems [15, 16], CCPSO2 [17] (Cooperatively Coevolving Particle Swarm) is a recent state-of-the-art algorithm based on the well known PSO [18] metaheuristic. It is a variant of a previous version called CCPSO [19, 20], being recognized by its computational simplicity and good capacity of dealing with multimodal complex functions and separable problems [21] [22]. CCPSO2-IP [23] is an Iterative Partitioning method for CCPSO2 that improves its capacity of dealing with non-separable problems by automating the task of finding combination of dependent dimensions, so that CCPSO2-IP can escape more easily from local minima traps caused by poor variable interaction. For a comprehensive test of CSMOn under complex optimization scenarios, CCPSO2-IP with exponential boost function (CCPSO2-IP E) has been used to optimize CEC13 functions due to its strong capacity of escaping from local minima and keep converging after long periods of stagnation on high dimensional problems, producing convergence behaviors of hard prediction. For the random MSG functions, the swarm algorithms CCPSO2-IP E, PSO [24] and ABC [25] (Artificial Bee Colony) have been tested.

The results are then summarized with the mixed model based on confidence intervals proposed by [26][26], which creates a simple pairwise comparison plot (between versions with and without CSMOn) that shows how significant is the difference between results.

4.1 Configurations

Considering that the CSMOn stop condition does not entirely depend on the number of evaluations M (i.e. it mainly depends on the convergence behavior and the relaxation parameter), the search algorithms are processed in two phases to compare the fitness obtained with and without the use of CSMOn:

1. under the control of CSMOn method (i.e., until the last convergence stabilization p_S is found).

2. it continues the optimization on the same run until M fitness evaluations are performed (i.e., processing additional $M - p_S$ fitness evaluations).

For the CEC13 test set, CCPSO2-IP E configuration is set as: $p = 30$, $B_r = 0.5$, $maxTries = 2$. The following values are used for M: $M1 = 1e6$, $M2 = 3e6$, $M3 = 6e6$ and $M4 = 1e7$. For each M, the following values are used for R (both for fixed and quadratic decreasing relaxations): {0.01, 0.02, ..., 0.09, 0.1, 0.2,..., 0.9}. Every benchmark function F is processed 30 times for each combination of $M \times R$ and an analysis has been conducted aiming to evaluate:

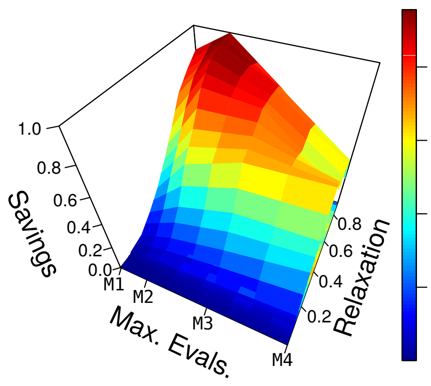
1. The effect of the relaxation parameter: For every function F and each combination of $M \times R$, both the economy on the number of fitness evaluations ($(M - ne_{p_S})/M$) and the complement of the relative approximation error of the fitness ($1 - |(fit_M - fit_{p_S})/fit_M|$) are multiplied to obtain the trade-off matrix. Matrices for all 15 functions F are averaged to illustrate the effect of the relaxation parameter as the maximum number of evaluations increases, where higher values mean better commitment between the fitness approximation and the economy on function evaluations.

2. Overall advantages/drawbacks for best relaxations: From the previous experiment, two relaxation parameters are chosen (R_{low}, R_{high}). For each function F, the best from these two parameters is used to compare the best fitness obtained in every execution (for both M and p_S evaluations) and the averaged evaluation savings for each p_S.

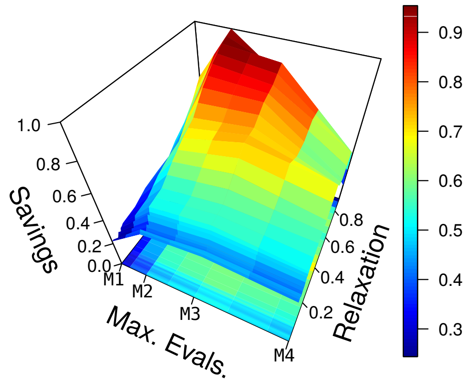
3. Detailed advantages/drawbacks for best functions: This experiment aims to visualize how far the stabilization points (p_S) found by CSMOn are from the best possible values (restricted to M). For the best performed functions F, the percentage of reduction on fitness approximation quality is compared with the percentages of economy on function evaluations obtained by CSMOn, using as configurations all values of M and the two previously selected relaxations.

For the second test set, the MSG landscape generator is used to create test classes of 50 dimensions (varying the number of Gaussians randomly in the interval $[50, 100]$). Configurations are empirically set to CCPSO2-IP E: $\{p = 30, B_r = 0.6, maxTries = 3\}$, PSO: $\{p = 30, c1 = 0.4, c2 = 1.4, w = 0.9\}$[27] and ABC: $\{foodSource = 30, trials = 1e3\}$:

(4) Every search algorithm is executed 30 times and, for each execution, a new test class is created from which a random test function is instantiated and optimized 30 times independently, with $M = 5e4$. At the end, 900 executions were be performed for each search algorithm.



(a) Fixed relaxation



(b) Quadratic relaxation

Figure 2: Averaged Cost/Benefit of CSMOn for CEC13 Functions. ($M1 = 1e6$, $M2 = 3e6$, $M3 = 6e6$, $M4 = 1e7$)

4.2 Results and Analysis

Experiments described in section 4.1 are discussed below. For the CEC13 functions:

Experiment 1: Figure 2(a) shows the results for fixed relaxation and Figure 2(b) for quadratic decreasing relaxation. Although they look similar, fixed relaxation presents a smoother trade-off matrix (soft transitions between configurations), with better results for small M and high R. However, quadratic decreasing relaxation obtains overall better results for both high M and low R.

Experiment 2: From the previous experiment, $R_{low} = 0.08$ and $R_{high} = 0.7$ are chosen. Figure 3 presents the best possible results (in blue) and CSMOn's results (in red). Confirming the good convergence of CCPSO2-IP E for functions F3, F5, F6, F9 and F10 [23], best results are obtained on F5 and F9, with high relaxation $R = 0.7$, exceeding 90% economy (the red values on X axis), showing that for problems where the search algorithm is well suited, large relaxations should suffice (although some medians seem differ, their relative differences are very small).

Function F2 processed almost up to M evaluations, meaning that CCPSO2-IP E keeps converging throughout the search. For the other 9 functions (comprising 36 combinations of F $\times M$), 28 combinations have their medians and 15 have their quartiles similar to the best possible results, with up to 49% economy on latter 15 combinations. Fully separable F1 and F2 are special cases since for those functions CCPSO2-IP E presents an erratic convergence with frequent transitions between poor logarithmic and moderate exponential decays, meaning that the method would process up to M evaluations if smaller relaxations were used.

Experiment 3: Figure 4 shows the percentages of advantage and drawback for functions F3, F5, F6, F9 and F10 with quadratic decreasing relaxations $R_{low} = 0.08$ and $R_{high} = 0.7$. Results clearly show the advan-

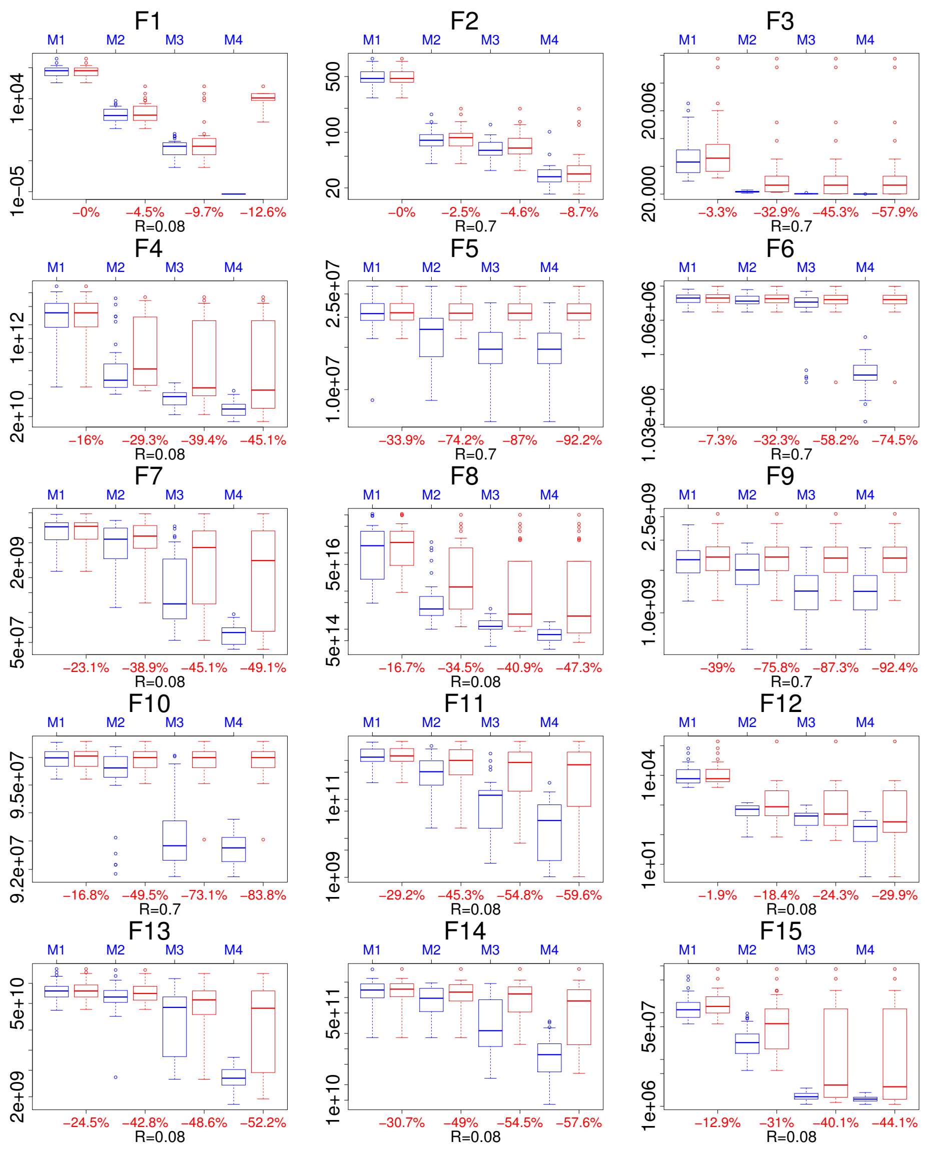


Figure 3: Comparison Between CSMOn stopping point and Processing until M evaluations.

Quadratic decreasing $R = \{0.08, 0.7\}$ (indicated below bottom axis). Red box is CSMOn method and blue box is limited by M. Vertical axis is the fitness value. Actual number of CSMOn's function evaluations is given by M configuration (top axis) minus respective percentage of economy (bottom axis).

tages overcoming the drawbacks in all 40 combinations of F $\times M \times R$, with R_{high} presenting the highest economy, reaching above 80% difference between drawback and advantage on function F10.

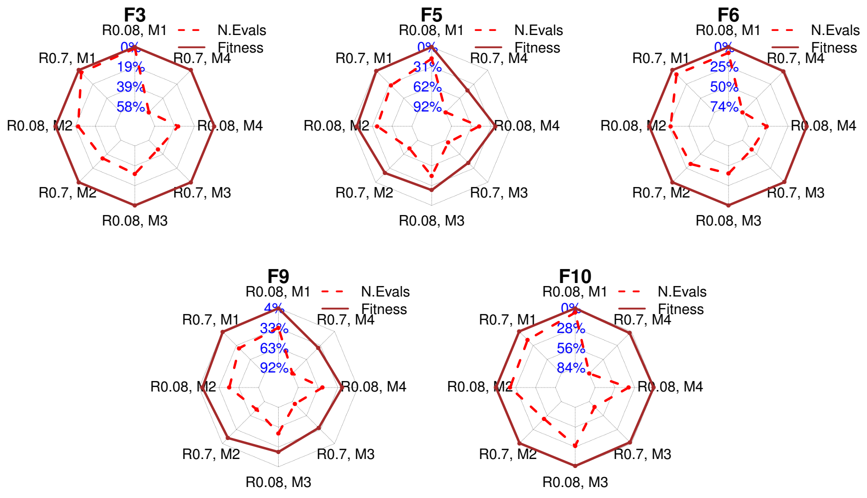


Figure 4: Percentage of Economy on Fitness Evaluations $\times$ Reduction on Fitness due to Quadratic Decreasing Relaxations ($R = \{0.08, 0.7\}$). Best configurations are Fitness on borders and N.Evals on center.

For the MSG functions:

Experiment 4: In the paired comparison plot (Figure 5), overlapping intervals show that the means are not significantly different, leading to the conclusion that there is no significant difference between results with and without CSMOn for CCPSO2-IP E. Despite the fact that the figure shows that PSO with and without CSMOn are not similar, the normalized fitness difference is within a small percentage (3%), with minor importance when compared to the economy of 27% on fitness evaluations. The small economy obtained by CCPSO2-IP E (6%) and ABC (6.7%) occurred because these algorithms were still converging for most of the runs, reaching M evaluations. The 4% significance difference for ABC is due to the "scout bees" phase, that reset the position from current to a different local minimum, suggesting that CSMOn could be used to detect upfront a better moment for the scout phase to occur.

Although not included here, additional tests with fixed relaxations have shown that they are more conservative on convergence stabilization analysis than decreasing relaxation, tending to produce smaller drawbacks with moderate savings on erratic optimization.

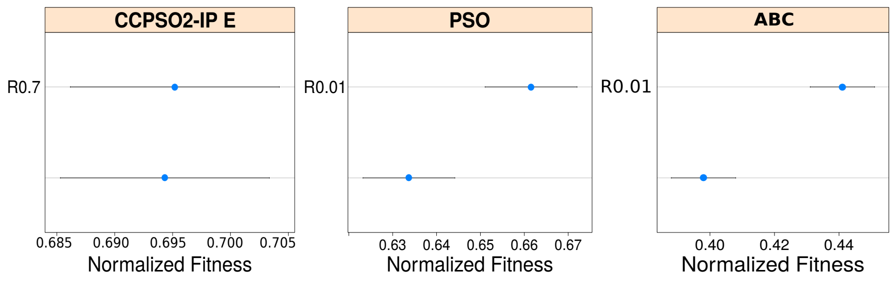


Figure 5: Paired comparison of CCPSO2-IP E, PSO and ABC search algorithms, with and without CSMOn.
MSG landscape generator used to create 30 test classes per algorithm, one test function instantiated per class and 30 executions per function using $M = 5e4$. Overall fitness evaluations economy: CCPSO2-IP E=6%, PSO=27%, ABC=6.7%. Decreasing relaxation values (R) shown on labels.

5 Conclusion

Tracking convergence trends for stochastic algorithms is not a trivial task, however if successfully done could benefit the overall optimization by controlling drawbacks resulting from fitness approximation due to evaluation savings. This work has presented a method to track this trend for swarm based stochastic metaheuristic.

The proposed approach (CSMOn) has been tested with CCPSO2-IP E search algorithm on all 15 CEC13 benchmark functions. Results show that the process presented to model the convergence stabilization behavior is able to effectively adapt to each optimization in progress (online), obtaining a good trade-off between the drawback resulting from the fitness approximation and the advantage of saving fitness function evaluations. The best results are obtained with functions where CCPSO2-IP E has a stable convergence behavior, presenting a great economy with the fitness consistently approaching the best possible values (i.e. values achieved considering the restriction of maximum number of evaluations M). At some cases, CSMOn obtains above 90% economy with moderate drawbacks on the fitness estimation, and in other cases it obtains above 70% positive difference between advantages and drawbacks. Tests with Max-Set random functions show that CSMOn is able to detect, for metaheuristics CCPSO2-IP E, PSO and ABC, when the search is no longer converging, suggesting that it should be either stopped or some additional action should be taken by the search algorithm to reactivate the convergence.

Future work would include: the test to take into consideration long stagnation periods of the search algorithm; different update mechanisms for the relaxation parameter aiming to react based on the distance to the maximum number of evaluations; a relaxation update mechanism to consider the remaining budget in a multiple run scenario; and the test of the CSMOn method with non-swarm memory-based metaheuristics.

References

[1] J. Sun, X. Wu, V. Palade, W. Fang, C.-H. Lai, and W. Xu, "Convergence analysis and improvements of quantum-behaved particle swarm optimization," *Information Sciences*, vol. 193, pp. 81–103, 2012.

[2] "Chapter 21 metaheuristics," in *Simulation*, ser. Handbooks in Operations Research and Management Science, S. G. Henderson and B. L. Nelson, Eds. Elsevier, 2006, vol. 13, pp. 633 – 654.

[3] J. Pichitlamken and B. L. Nelson, "A combined procedure for optimization via simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 13, no. 2, pp. 155–179, 2003.

[4] W. J. Gutjahr, "Convergence analysis of metaheuristics," in *Matheuristics.* Springer, 2009, pp. 159–187.

[5] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[6] T. Bartz-Beielstein and M. Zaefferer, "Model-based methods for continuous and discrete global optimization," *Applied Soft Computing*, vol. 55, pp. 154–167, 2017.

[7] S. Zhang, J. Xu, L. H. Lee, E. P. Chew, W. P. Wong, and C.-H. Chen, "Optimal computing budget allocation for particle swarm optimization in stochastic optimization," *IEEE Transactions on Evolutionary Computation*, 2016.

[8] J. Rada-Vilela, M. Zhang, and M. Johnston, "Optimal computing budget allocation in particle swarm optimization," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation.* ACM, 2013, pp. 81–88.

[9] P. F. Perroni, *CSMOn*, Federal University of Paraná, Curitiba, Brazil, 2017. [Online]. Available: http://web.inf.ufpr.br/vri/software/csmon

[10] J. L. Devore, *Probability and Statistics for Engineering and the Sciences, Eighth Edition, Chapter 13.* Cengage Learning, 2010.

[11] X. Li, K. Tang, M. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the cec'2013 special session and competition on large scale global optimization," Evolutionary Computation and Machine, 2013.

[12] M. Gallagher and B. Yuan, "A general-purpose tunable landscape generator," *IEEE transactions on evolutionary computation*, vol. 10, no. 5, pp. 590–603, 2006.

[13] T. Bartz-Beielstein, "How to create generalizable results," in *Springer Handbook of Computational Intelligence.* Springer, 2015, pp. 1127–1142.

[14] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: a survey," *Information Sciences*, vol. 295, pp. 407–428, 2015.

[15] F. Caraffini, F. Neri, M. Gongora, and B. N. Passow, "Re-sampling search: A seriously simple memetic approach with a high performance," in *2013 IEEE Workshop on Memetic Computing (MC).* IEEE, 2013, pp. 52–59.

[16] F. Caraffini, F. Neri, B. N. Passow, and G. Iacca, "Re-sampled inheritance search: high performance despite the simplicity," *Soft Computing*, vol. 17, no. 12, pp. 2235–2256, 2013.

[17] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *Evolutionary Computation, IEEE*, vol. 16, no. 2, pp. 210–224, 2012.

[18] R. C. Eberhart, J. Kennedy *et al.*, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[19] X. Li and X. Yao, "Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms," in *IEEE Congress on Evolutionary Computation, 2009. CEC'09.* IEEE, 2009, pp. 1546–1553.

[20] C. K. Goh, K. C. Tan, D. Liu, and S. C. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *EJOR*, vol. 202, no. 1, pp. 42–54, 2010.

[21] I. Poikolainen, G. Iacca, F. Caraffini, and F. Neri, "Focusing the search: a progressively shrinking memetic computing framework," *International Journal of Innovative Computing and Applications*, vol. 5, pp. 127–142, 2013.

[22] S. S. Poorjandaghi and A. Afshar, "A robust evolutionary algorithm for large scale optimization," in *World Congress*, vol. 19, no. 1, 2014.

[23] P. F. Perroni, D. Weingaertner, and M. R. Delgado, "Automated iterative partitioning for cooperatively coevolving particle swarms in large scale optimization," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, 2015, pp. 19–24.

[24] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE transactions on evolutionary computation*, vol. 8, no. 3, pp. 225–239, 2004.

[25] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, Tech. Rep., 2005.

[26] M. Chiarandini and Y. Goegebeur, "Mixed models for the analysis of optimization algorithms," in *Experimental Methods for the Analysis of Optimization Algorithms.* Springer, 2010, pp. 225–264.

[27] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters*, vol. 85, no. 6, pp. 317–325, 2003.