On Local Convergence of Stochastic Global Optimization Algorithms*

Eligius M. T. Hendrix[1][0000−0003−1572−1436] and Ana Maria A. C. Rocha[2][0000−0001−8679−2886]

[1] Computer Architecture, Universidad de Málaga, 29080 Málaga, Spain
eligius@uma.es
[2] ALGORITMI Center, University of Minho, 4710-057 Braga, Portugal
arocha@dps.uminho.pt

Abstract. In engineering optimization with continuous variables, the use of Stochastic Global Optimization (SGO) algorithms is popular due to the easy availability of codes. All algorithms have a global and local search character, where the global behaviour tries to avoid getting trapped in local optima and the local behaviour intends to reach the lowest objective function values. As the algorithm parameter set includes a final convergence criterion, the algorithm might be running for a while around a reached minimum point. Our question deals with the local search behaviour after the algorithm reached the final stage. How fast do practical SGO algorithms actually converge to the minimum point? To investigate this question, we run implementations of well known SGO algorithms in a final local phase stage.

Keywords: Stochastic Global Optimization; Evolutionary algorithms; Convergence; Nonlinear optimization

1 Introduction

In many engineering applications [6], we consider a black-box global optimization problem

$$f^* = \min_{x \in X} f(x), \tag{1}$$

where $f(x)$ is a continuous function and $X \subset \mathbb{R}^n$ is a feasible region. The idea of the black-box optimization is that function evaluations imply running an external (black-box) routine that may take minutes or hours to provide the evaluated objective function value. In engineering applications, often the question is to obtain a good, but not necessarily optimal solution within a day, several days, or a week.

The choice of engineers for solution algorithms is often driven by ease of availability and the attractiveness of the intuitive behaviour. Therefore, Stochastic

* This paper has been supported by The Spanish Ministry (RTI2018-095993-B-I00) in part financed by the European Regional Development Fund (ERDF) and by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UIDB/00319/2020.

Global Optimization (SGO) algorithms based on evolutionary equivalences have been most popular over the last decades. According to the generic description of [13], a new iterate to be evaluated is generated according to:

$$x_{k+1} = \text{Alg}(x_k, x_{k-1}, \ldots, x_0, \xi), \tag{2}$$

where ξ is a pseudo-random variable and index k is the iteration counter. Description (2) represents the idea that a next point x_{k+1} is generated based on the information in all former points $x_k, x_{k-1}, \ldots, x_0$ and a random effect ξ based on generated pseudo-random numbers.

If the algorithm behaves well, it might converge to a global optimum solution or even several of them. In our investigation, we focus on the situation where an algorithm converges to one of the global minimum points $x^* \in \text{argmin}_{x \in X} f(x)$ which is interior with respect to the feasible region. In fact, it is better to switch to a nonlinear optimization algorithm from the iterate or one of the population points in that case. However, as the user is not always aware that the algorithms reached the final stage, it may continue for a while up to reaching a predefined stopping criterion.

In nonlinear optimization, the concept of convergence speed is well defined, see e.g., [3]. It deals with the convergence limit of the series x_k. Let $x_0, x_1, \ldots, x_k, \ldots$ converge to point x^*. The largest number α for which

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^\alpha} = \beta < \infty \tag{3}$$

gives the order of convergence, whereas β is called the convergence factor. In this terminology, the special instances are

- linear convergence with $\alpha = 1$ and $\beta < 1$
- quadratic convergence with $\alpha = 2$ and $0 < \beta < 1$
- superlinear convergence: $1 < \alpha < 2$ and $\beta < 1$, i.e. $\beta = 0$ if $\alpha = 1$ in (3).

How is this limit behaviour for SGO algorithms? First of all, we are dealing with (pseudo-)random iterates that may or not behave in a Markovian way in the limit situation. Second, many SGO algorithms are population algorithms as sketched in the generic Algorithm 1. This means that stopping criteria in

Algorithm 1 GPOP(X, f, N)

Generate set $\mathcal{P} = \{p_1, \ldots, p_N\}$ of N random points in X and evaluate
repeat
    Generate a set of trial points $\mathcal{X}$ based on $\mathcal{P}$ and evaluate
    Replace $\mathcal{P}$ by a selection of points from $\mathcal{P} \cup \mathcal{X}$
until (stopping criterion)

limit (when no maximum number of evaluations is set) may be based on the

convergence of the distance in the population (swarm) $\max_{p,q\in\mathcal{P}} \|p-q\|$, or the function value $\max_{p,q\in\mathcal{P}} |f(p)-f(q)|$. The first criterion is not appropriate when the population clusters around several minimum points. In [5], the convergence behaviour of Controlled Random Search (CRS) is studied also for the case where we have convergence to several minimum points.

In this study, we focus on the behaviour of six SGO algorithms for convergence to one minimum point of a smooth function. This facilitates considering the limit situation as the behaviour over a convex quadratic function

$$f(x) = f(x^*) + \frac{1}{2} x^T H(x^*) x, \tag{4}$$

where $H(x^*)$ is the Hessian in the minimum point x^*.

According to [5], the speed of convergence of a SGO algorithm is determined by the so-called success rate

$$S_k := P\left(f(x_{k+1}) < \max_{p\in\mathcal{P}} f(p) \right), \tag{5}$$

as the probability that the next generated iterate is better than the worst function value in the population. Our research question is how S_k is behaving when practically used population SGO algorithms solve problem (4). Specifically

1. Are the algorithms going to behave in a stationary way, i.e. is S_k converging?
2. How do the algorithms compare with respect to convergence speed on the same Hessian?
3. How do the algorithms behave for varying dimension and condition number of the Hessian $H(x^*)$ and are algorithms sensitive to rotation, i.e. does their behaviour depend on the eigenvectors?

The latter is related to the roundness of the ellipsoidal level sets of function (4).

To investigate these questions, in Section 2, we describe the experimental design of instances that vary in dimension and condition number of the Hessian. Section 3 provides pseudo-code for each investigated algorithm and presents its convergence behaviour. Section 4 summarizes our findings.

2 Experimental Setting

In order to measure the convergence, we focus on the worst function value in the population $\mathcal{P}_k$ at iteration k

$$\text{fworst}_k := \max_{p\in\mathcal{P}_k} f(p)$$

and measure the success rate as the number of points in the population, that are better than the worst value of the last population $\mathcal{P}_{k-1}$:

$$S_k := \frac{1}{N} \sum_{p\in\mathcal{P}_k} (f(p) < \text{fworst}_{k-1}), \tag{6}$$

where N is the population size. The starting population $\mathcal{P}_0$ consists of a uniform random sample over the level set defined by $\mathcal{P}_0 := \{x \in \mathbb{R}^n | x^T H x \leq 1\}$. The same starting population is used over different SGO algorithms. The population algorithms have different moments to update the population. CRS [11] updates the population after every iteration (function evaluation), i.e. $|\mathcal{X}| = 1$. It decides to replace the worst point in the population or not. In an algorithm based on swarms, the population is updated after all points in the population have been moved, i.e. $|\mathcal{X}| = N$. For the latter type of algorithms, we will update the measured success rate after replacement of the population.

The condition number and eigenvectors of the Hessian H allow us to vary the ellipsoidal shape of the initial population and the experimental setting. We are going to use 7 instances as depicted in Table 1, where the dimension n of the problem and the roundness of the level set are varied. The latter is related to the condition number of the Hessian which is the ratio of the largest and smallest eigenvalue in our context, [3].

For the two dimensional cases, we consider 3 instances of the Hessian; the identity matrix I_n, the Hessian of the Trid[3] quadratic function H_{Tn}, which is said to be difficult for population algorithms and a skewed matrix H_S in two dimensional space: $H_{T2} := \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$ and $H_S := \begin{pmatrix} 101 & -99 \\ -99 & 101 \end{pmatrix}$. Moreover, we also extend the identity matrix and Trid function to dimension $n = 2, 5, 10$. In general, we inspect the development of the indicators up to complete convergence, i.e. the final population is practically one point, or at most up to $1000 \times n$ function evaluations. We use the same seed for the generated pseudo-random numbers for all experiments with a long run instead of repeating the run several times. This implies that the starting population $\mathcal{P}_0$ is the same for each algorithm.

Table 1: Experimental setting varying the Hessian in the minimum point

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
n	2	5	10	2	2	5	10
cond nr	1	1	1	100	3	13.92	48.37

3 Evaluation of Algorithms

We investigate the behaviour of a total of six stochastic population algorithms. In the sequel, we leave out the iteration counter k from their pseudo-code description.

[3] `https://www.sfu.ca/~ssurjano/trid.html`

3.1 Controlled Random Search

Price [11] introduced CRS in 1979. It has been widely used and also modified into many variants by himself and other researchers. Investigation of the algorithm shows mainly numerical results. Algorithm 2 describes the initial scheme. It generates points in a Nelder-Mead-way on randomly selected points from the current population.

Algorithm 2 CRS(f, X, N)

Generate and evaluate a set $\mathcal{P}$ of N random points uniformly on X
repeat
  Find the worst point q in the population $\mathcal{P}$, $q \leftarrow \operatorname{argmax}_{p\in\mathcal{P}} f(p)$
  fworst $\leftarrow f(q)$
  select at random a subset $\{y_0, \ldots, y_n\}$ from $\mathcal{P}$
  $x \leftarrow \frac{2}{n}\sum_{i=1}^{n} y_i - y_0$ and evaluate $f(x)$
  if $(f(x) <$ fworst$)$
    Replace, in $\mathcal{P}$, point q by x
until (stopping criterion)

In later versions, the number of parents $n + 1$ is a parameter m of the algorithm. A so-called secondary trial point, which is a convex combination of the parent points is generated when the first type of points does not lead to sufficient improvement. In that version, a rule keeps track of the success rate.

Table 2: Order of magnitude reached function value for the worst point in the population fworst after $1000n$ evaluations and average success rate $\overline{S}$ for CRS

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	50	50	100	50	50	50	100
fworst	10^{-11}	10^{-10}	10^{-8}	10^{-9}	10^{-9}	10^{-10}	10^{-7}
$\overline{S}$	0.47	0.43	0.47	0.47	0.47	0.43	0.47

We run the algorithm over all test instances. The population size is $N = 50$ for $n \leq 5$ and $10n$ otherwise. Figure 1 gives insight in the course of the algorithm for the Trid function in dimension $n = 2$. In all dimensions the algorithm practically converges to the minimum point. In dimension $n = 2$ after 500 evaluations and in dimension $n = 10$ after 3000 iterations, the maximum value of the population reaches a value of about 0. The average success rate is $\overline{S} = 0.469$ for both cases H_{T10} and I_{10}, as illustrated in Fig. 2. This is no coincidence. The specific behaviour if CRS is that the generation mechanism with the same (pseudo) random numbers is the same if dimension and population size N is fixed *independent* of the shape of the Hessian. This means, for $n = 2$, the same average success rate is

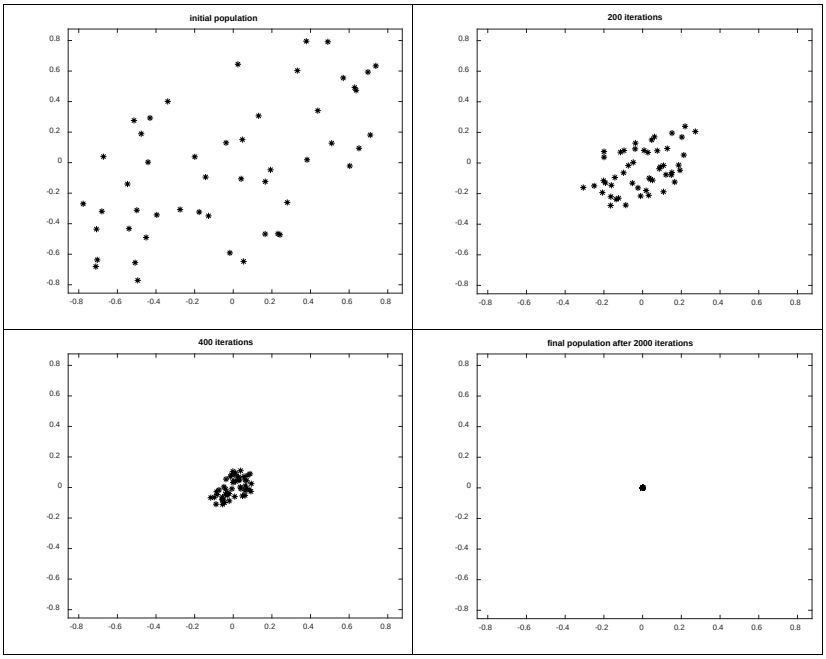


Fig. 1: Population development of CRS for the Trid Hessian H_{T2}, $N = 50$.

reached for I_2, H_{T2} and H_S, i.e. $\overline{S} = 0.466$. In 5-dimensional space, for both H_{T5} and I_5, the reached value fworst after 5000 iterations is about fworst$\approx 10^{-10}$, but the average success rate is exactly the same, i.e. $\overline{S} = 0.430$.

What we learn from this experiment is that the local search behaviour of CRS is independent of the Hessian in the minimum point and that the average success rate, although varying, does not reveal a trend during the iterations. Moreover, it is quite independent of the dimension n and population size N. Of course the speed of convergence goes down with the population size N as analyzed in [5]. We expect this constant behaviour to be quite unique among the stochastic population algorithms.

3.2 Uniform Covering by Probabilistic Rejection

An alternative for CRS which focuses on only generating points around global minimum points to get a uniform cover of the lower level set is called Uniform Covering by Probabilistic Rejection (UCPR) introduced in 1994, [9]. The method has mainly been developed to be able to cover a level set $S(f^* + \delta)$ which represents a confidence region in nonlinear parameter estimation. The idea is to cover with a sample of points $\mathcal{P}$ as if they are from a uniform distribution or with a so-called Raspberry set $R := \{x \in X \mid \exists p \in \mathcal{P}, \|x - p\| \leq c \times r\}$, where r is a small radius following the idea of the average nearest neighbor distance

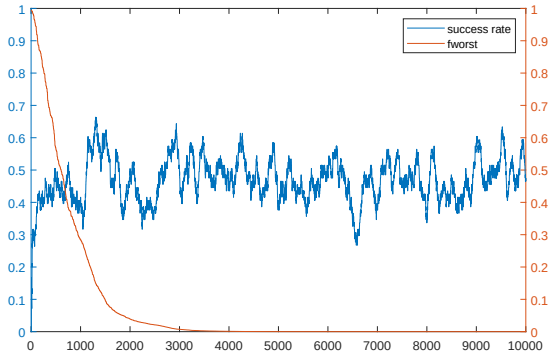


Fig. 2: Development of CRS for the Trid Hessian H_{T10}. In orange the reached function value of the worst point in the population and in blue the success rate.

for approximating the inverse of the average density of points over S and c a parameter, see Algorithm 3. We will use a value of $c = 1.3$ in the experiments as suggested in [4].

Algorithm 3 UCPR(f, X, N, c)

Generate and evaluate a set $\mathcal{P}$, of N random points uniformly on X
repeat
  Find the worst point q in the population $\mathcal{P}$, $q \leftarrow \text{argmax}_{p\in\mathcal{P}} f(p)$
  fworst$\leftarrow f(q)$
  determine the average inter-point distance r in $\mathcal{P}$
  Raspberry set $R := \{x \in X \mid \exists p \in P, \|x - p\| \leq c \times r\}$
  Generate and evaluate x from a uniform distribution over R
  if ($f(x)$ <fworst)
    Replace, in $\mathcal{P}$, point q by x
until (stopping criterion)

Like CRS, the UCPR algorithm has a theoretical fixed success rate with respect to spherical functions that does not depend on level fworst$= \max_{p\in P} f(p)$ that has been reached. The success rate goes down with increasing dimension, as the probability mass goes to the boundary of the level set if dimension n increases. Therefore, more of the Raspberry set sticks out. Moreover, it suffers more from highly skewed Hessians, i.e. the condition number differs from 1.

We are going to measure these effects running the algorithm for the 7 instances in Table 1 using the same population size as for CRS. First, the development of the population for the highest condition number case H_S is sketched in Fig. 3. One can observe a fast convergence and the aim to cover all the level set with the population.

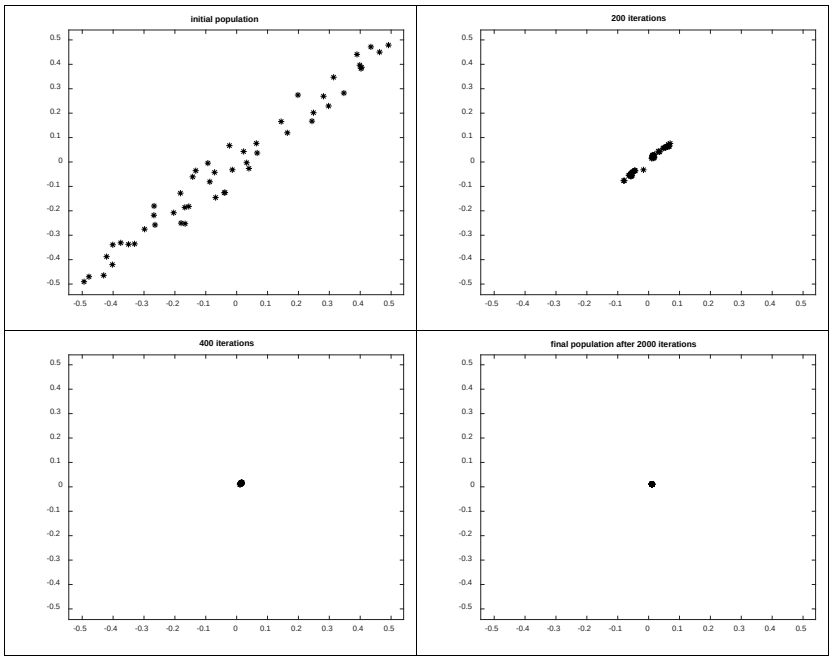


Fig. 3: Population development of UCPR on Hessian H_S.

The results for the 7 instances in Table 3 show us that UCPR has a strong local search behaviour, but in contrast to CRS, the success rate is going down with the dimension. Moreover, we can observe, that the behaviour is far more affected by the condition number. Actually, for the 2-dimensional cases, the algorithm converges too fast to a non-optimal point. For the largest cases, it seems not to have reached the optimum after 10,000 evaluations, due to a low success rate. Probably the algorithm requires a better tuning of the parameter c to the dimension n. In the presented theory, this should be correct automatically due to the nearest neighbour distance concept. Figure 4 depicts the relatively constant behaviour of the success rate for the Trid instance H_{T5}.

Table 3: Order of magnitude reached function value for the worst point in the population fworst after $1000n$ evaluations and average success rate $\overline{S}$ for UCPR

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	50	50	100	50	50	50	100
fworst	10^{-2}	10^{-8}	0.40	10^{-4}	0.01	10^{-8}	0.14
$\overline{S}$	0.94	0.41	0.05	0.92	0.93	0.40	0.09

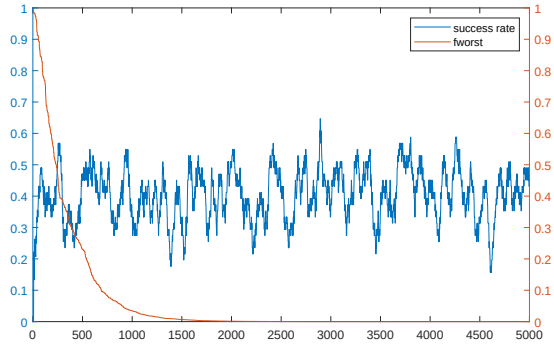


Fig. 4: Development of UCPR for the Trid Hessian H_{T5}. In orange the reached function value of the worst point in the population and in blue the success rate.

3.3 Genetic Algorithm

Although evolutionary algorithms existed before, Genetic Algorithms (GA) became known after the appearance of the book by Holland in 1975 [7] followed by many other works such as [2]. The generic population algorithm generates at each iteration a set of trial points based on a terminology from biology and nature: evolution, genotype, natural selection, reproduction, recombination, chromosomes etc. For instance, the average inter-point distance in a population is called diversity. The basic concepts as depicted in Algorithm 4 are the following

- the objective function is transformed into a fitness value,
- the points in the population are called individuals,
- points of the population are selected for making new trial points: parent selection for generating offspring,
- candidate points are generated by combining selected points: crossover,
- candidate points are varied randomly to become trial points: mutation,
- new population is composed from selecting from old and new points.

Algorithm 4 GA(f, X, N, M)

Generate and evaluate a set $\mathcal{P}$, of N random points uniformly on X
repeat
  Parent selection: select points used for generating candidates
  Crossover: create M candidates from selected points
  Mutation: vary candidates towards M trial points (offspring)
  Selection: create a new population out of $\mathcal{P}$ and the M trials
until (stopping criterion)

The fitness $F(x)$ of a point giving its objective function value $f(x)$ to be minimized can be taken via a linear transformation using extreme values $\max_{p\in\mathcal{P}} f(p)$

and $\min_{p\in\mathcal{P}} f(p)$. A higher fitness provides a higher probability to be selected as a parent. The probability for selecting point $p \in P$ is often taken as $\frac{F(p)}{\sum_{q\in\mathcal{P}} F(q)}$. Parameters of the algorithm deal with choices on: fitness transformation, the way of probabilistic selection, the number of parents, etc. We used the `ga` function from MATLAB2018b® implementation with the default values for the parameters to measure the convergence. Actually, we found that we have to include bounds to the search space in a box (bounds on the variables) around the initial population $\mathcal{P}_0$ in order to have convergence of the complete population. The observed convergence values can be found in Table 4.

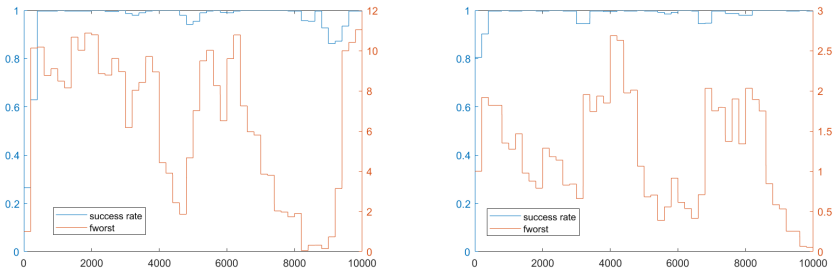
In earlier experiments, we were surprised to observe that using smaller population size, it appeared possible that duplicates of the same points appeared in the population. Probably therefore, every now and then a refreshment of the population takes place, as can be observed very well for the higher dimensional cases in Fig. 5. It may be clear that the algorithm, in contrast to CRS and UCPR, also accepts points that are worse in the population. Basically, the behaviour means that up to a stopping criterion is reached, GA keeps on a global search behaviour, even when the population settled around the minimum point. This can be observed very well for the H_{T10} case in Fig. 5, where the population converged after evaluation 8000 and then diverged again towards the total search space. There is no notable distinction with respect to the condition number of the Hessian.

Table 4: Order of magnitude reached function value fworst for the worst point in the population after $1000n$ evaluations and average success rate $\overline{S}$ for GA

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	50	50	200	50	50	50	200
fworst	10^{-5}	10^{-5}	0.05	10^{-4}	10^{-6}	10^{-6}	11.04
$\overline{S}$	0.98	0.98	0.99	0.98	0.93	0.97	0.97

3.4 Particle Swarm Algorithm

Kennedy and Eberhart in 1995 [8], came up with an algorithm with a terminology of "swarm intelligence" and "cognitive consistency". In each iteration of the algorithm, each member (particle) of the population $\mathcal{P}$, called swarm, is modified and evaluated. The traditional nonlinear programming modification by direction and step size is termed "velocity". Instead of considering $\mathcal{P}$ as a set, one better thinks of a list of elements, $p_j, j = 1, \ldots, N$. So we will use the index j for the particle. Besides its position p_j, also the best point z_j found by p_j is stored. A matrix of modifications (velocities) $[v_1, \ldots, v_N]$ is updated at each iteration containing random effects. The velocity v_j is based on points p_j, z_j and the best point found so far $x_b = \text{argmin}_j f(z_j)$ in the complete swarm. In the next



Hessian based on Trid, H_{T10} Identity matrix I_{10}

Fig. 5: Behaviour of GA for $n = 10$. In orange the reached function value fworst of the worst point in the population and in blue the success rate.

iteration, simply $p_j \leftarrow p_j + v_j$ as sketched in Algorithm 5. We will use the notation $D(x)$ for a diagonal matrix with the elements of x on the diagonal.

Algorithm 5 PSO(f, X, N, ω)

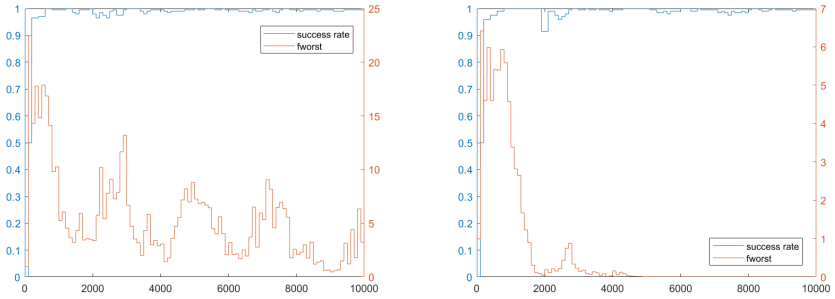
Generate and evaluate a set $\mathcal{P}$, of N random points uniformly on X
$\mathcal{Z} \leftarrow \mathcal{P}$; $v_j \leftarrow 0, \quad j = 1, \ldots, N$
repeat
  $x_b \leftarrow \text{argmin}_{z \in \mathcal{Z}} f(z)$, $f_b \leftarrow f(x_b)$
  for ($j = 1$ to N) do
    generate vectors r and u uniformly over $[0, 1]^n$
    $v_j \leftarrow \omega v_j + 2D(r)(z_j - p_j) + 2D(u)(x_b - p_j)$
    $p_j \leftarrow p_j + v_j$; evaluate $f(p_j)$
    if ($f(p_j) < f(z_j)$)
      $z_j \leftarrow p_j$
until (stopping criterion)

To study its convergence behaviour, we used the `particleswarm` function from MATLAB2018b® with its default parameter values. Like for GA, we also put bounds on the search space around the initial population $\mathcal{P}_0$ to avoid the swarm to go far away from the initial population. We were surprised by a far stronger local search behaviour of the swarm towards the minimum point than the GA algorithm. Very small values were reached. This effect is not directly clear from the pseudo-code description in Algorithm 5. The question with respect to the condition number, is very relevant for the PSO algorithm, as we can observe in Fig. 6. The geometric mechanism of generating new points is not insensitive to scaling of the swarm. First of all, we were surprised by the strange acceptance of worse points for I_{10} after about 2400 iterations. This is not a random effect, i.e. we repeated the same run with many random numbers and observed the

Table 5: Order of magnitude reached function value fworst of the worst point in the population after $1000n$ evaluations and average success rate $\overline{S}$ for PSO

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	20	50	100	20	20	50	100
fworst	10^{-20}	10^{-12}	10^{-6}	10^{-9}	10^{-24}	10^{-10}	3.25
$\overline{S}$	0.97	0.98	0.98	0.97	0.97	0.98	0.98

same bubble in the graph, corresponding to a diversity action. More impressive is the behaviour with respect to instance H_{T10}. It seems the swarm keeps on expanding cyclically hindering a clear convergence like for instance I_{10}.



Hessian based on Trid, H_{T10} Identity matrix I_{10}

Fig. 6: Behaviour of PSO for dimension $n = 10$. In orange the reached function value of the worst point in the population and in blue the success rate.

3.5 Differential Evolution

The algorithm, published by Storn and Price in 1997 [12], has a big impact on application in engineering environments. Probably its schedule is very attractive due to its simplicity. For each point p in the population, a trial point is created based on three other points in the population using a so-called differential weight β, which we will draw uniformly from a box. Each component i is replaced with a certain so-called crossover probability γ. If the new trial point is better, then individual p is replaced by the trial point.

This means that like in CRS and UCPR, only better points are allowed in the new population. Like in these algorithms, when the population $\mathcal{P}$ is caught by a basin around a minimum, no global search in other regions is done. One can find an analysis of the non-convergence from some specific populations in [10]. This

Algorithm 6 DE$(f, X, N, \gamma, [\underline{B}, \overline{B}])$

Generate and evaluate a set $\mathcal{P}$, of N random points uniformly on X
repeat
  for $(p \in \mathcal{P})$ do
    draw three points (a, b, c) randomly from $\mathcal{P}$
    pick component j uniformly from $\{1, \ldots, n\}$
    generate β uniformly over $[\underline{B}, \overline{B}]^n$
    trial point $x \leftarrow p$ and set component $x_j \leftarrow a_j + \beta_j(b_j - c_j)$
    for the other components $(i \neq j)$ do
      with probability γ set component $x_i \leftarrow a_i + \beta_i(b_i - c_i)$
    if $(f(x) < f(p))$
      $p \leftarrow x$
until (stopping criterion)

is in contrast to the Genetic algorithm we have seen and the Particle Swarm algorithm. In a Markovian sense, the basin around the minimum works as an attraction state set. This means that we expect a strong local search behaviour of the algorithm.

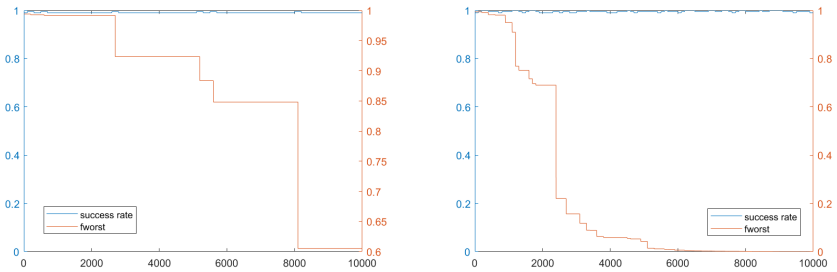
Table 6: Order of magnitude reached function value fworst for the worst point in the population after $k = 1000n$ evaluations and average success rate $\overline{S}$ for DE

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	20	50	100	20	20	50	100
fworst	10^{-23}	10^{-9}	10^{-4}	10^{-5}	10^{-17}	0.04	0.61
$\overline{S}$	0.99	0.99	0.99	0.96	0.98	0.98	0.99

A description of the used experimental variant[4] is depicted in Algorithm 6. The population size was taken as $N = \min(10n, 100)$, the crossover probability was taken as $\gamma = 0.2$ and the differential weight is taken from the interval $[\underline{B}, \overline{B}] = [0.2, 0.8]$.

The strong local search behaviour after the population reaches a minimum point can be observed in Table 6. From the reproduction mechanism, it is more clear than for the PSO or GA algorithms that the process is thought of component-wise. This means that the population does not automatically adapt to the shape of the level set. We can observe this effect very well in Fig. 7. For the high condition number instance H_{T10}, the algorithm does not find improvements easily for the worst point in the population for a long time. For the instance I_{10} with spherical level sets, the probability of finding better points is relatively constant.

[4] Taken from `https://yarpiz.com/231/ypea107-differential-evolution`.



Hessian based on Trid, H_{T10} Identity matrix I_{10}

Fig. 7: Behaviour of DE for dimension $n = 10$. In orange the reached function value of the worst point in the population and in blue the success rate.

3.6 Firefly Algorithm

The Firefly algorithm (FA) was introduced by Yang in 2008 [14], as a variant of the PSO algorithm based on nature inspired mechanisms. Each point in the population is moved towards all better points having a so-called higher lightness in the population adding a random mutation. Then all individuals are evaluated again.

Algorithm 7 FA$(f, X, N, \alpha, \beta, \gamma)$

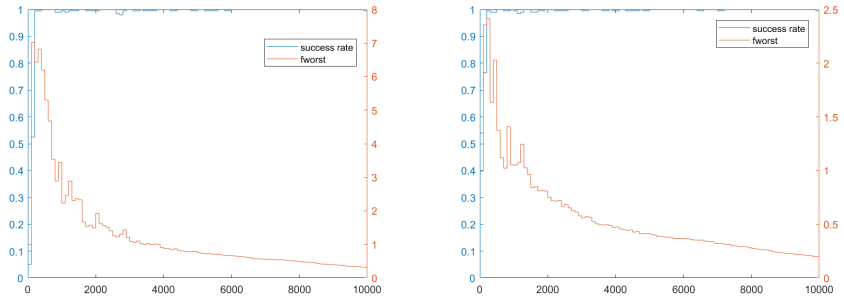
Generate a set $\mathcal{P}$ of N random points uniformly on X
repeat
  Evaluate and rank $f(p)$ for all $p \in \mathcal{P}$
  for all $(p \in \mathcal{P})$ do
    for all $(q \in \mathcal{P}$ with $f(q) \leq f(p))$ do
      generate a pseudo random number ε from the Normal distribution
      $p \leftarrow p + \beta e^{-\gamma \|p-q\|^2} + \alpha\varepsilon$
until (stopping criterion)

For the experiments, we used the MATLAB code that Yang published in [14] with parameter values $\alpha = 0.25$ (randomness), $\beta = 0.20$ (attraction parameter) and $\gamma = 1$ (absorption coefficient). These last two parameters are used in the exponential function for calculating the attractiveness between fireflies. The implementation requires bounds on the variables in order to use a good scaling. However, we found that for the instances we use this is not relevant due to the way the initial population is taken uniformly over the level set with a function value of 1. Actually, the algorithm can easily be extended to deal with constrained problems, [1].

Table 7: Order of magnitude reached function value fworst of the worst point in the population after $1000n$ evaluations and average success rate $\overline{S}$ for FA

Hessian	I_2	I_5	I_{10}	H_S	H_{T2}	H_{T5}	H_{T10}
dim n	2	5	10	2	2	5	10
pop N	20	50	100	20	20	50	100
fworst	10^{-7}	10^{-4}	0.20	10^{-6}	10^{-7}	10^{-4}	0.32
$\overline{S}$	0.98	0.99	0.99	0.97	0.98	0.98	0.99

In contrast to the particle swarm algorithm, no moves are based on the best point in the population. Basically, the population focuses towards the center of the population with only additional random effects. Therefore, we expect the algorithm like UCPR to shape better along the level set of the function. This effect can be observed in Fig. 8, where the algorithm seems not affected by the condition number of the Hessian. We observe however, that for the high dimensional cases the population initially deviates from the initial level set. As depicted in Table 7, it has a strong convergence to the minimum point.



Hessian based on Trid, H_{T10} Identity matrix I_{10}

Fig. 8: Behaviour of FA for dimension $n = 10$. In orange the reached function value of the worst point in the population and in blue the success rate.

4 Conclusions

We investigated the convergence behavior towards a minimum point of 6 population metaheuristics when the initial population starts around the point. We varied the dimension and condition number of the Hessian in the minimum point. The algorithms CRS, UCPR and DE only accept better points within the new population, whereas GA, PSO and FA keep a global search behavior. For GA,

this is due to periodic refreshment of the population. We found that GA and PSO require a bounding around the initial population in order not to diverge.

CRS appears completely insensitive to the shape of the Hessian and converges strongly to one point, whereas UCPR and DE are far more sensitive and for the highest condition number converge much slower. The FA algorithm converges slow for all cases, whereas the GA and PSO do not converge well when the level sets around the minimum are not round.

References

1. Costa, M.F.P., Francisco, R.B., Rocha, A.M.A.C., Fernandes, E.M.G.P.: Theoretical and practical convergence of a self-adaptive penalty algorithm for constrained global optimization. Journal of Optimization Theory and Applications **174**, 875–893 (2017)
2. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
3. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Academic Press, New York (1981)
4. Hendrix, E.M.T., Klepper, O.: On uniform covering, adaptive random search and raspberries. Journal of Global Optimization **18**, 143–163 (2000)
5. Hendrix, E.M.T., Ortigosa, P., Garcia, I.: On success rates for controlled random search. Journal of Global Optimization **21**, 239–263 (2001)
6. Hendrix, E.M.T., Tóth, B.G..: Introduction to Nonlinear and Global Optimization. Springer, New York (2010)
7. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
8. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948. Piscataway, NJ (1995)
9. Klepper, O., Hendrix, E.M.T.: A comparison of algorithms for global characterization of confidence regions for nonlinear models. Environmental Toxicology and Chemistry **13**(12), 1887–1899 (1994)
10. Locatelli, M., Vasile, M.: (Non) convergence results for the differential evolution method. Optimization Letters **9**(3), 413–423 (2014)
11. Price, W.: A controlled random search procedure for global optimization. The Computer Journal **20**, 367–370 (1979)
12. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
13. Törn, A., Žilinskas, A.: Global Optimization, Lecture Notes in Computer Science, vol. 350. Springer, Berlin (1989)
14. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Lunivers Press (2008)