The unscented genetic algorithm for fast solution of GA-hard optimization problems

Article *in* Applied Soft Computing · March 2023

DOI: 10.1016/j.asoc.2023.110260

1 author:

Anton Aguilar-Rivera
CTTC Catalan Telecommunications Technology Centre
11 PUBLICATIONS **183** CITATIONS

Highlights

The Unscented Genetic Algorithm for Fast Solution of GA-Hard Optimization Problems

Anton Aguilar-Rivera

- A novel competent evolutionary algorithm is proposed based on unscented Kalman filter theory.
- The proposed approach attained better performance than other competent evolutionary algorithms for increasing problem sizes.

The Unscented Genetic Algorithm for Fast Solution of GA-Hard Optimization Problems

Anton Aguilar-Rivera[a]

[a]*Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) Sustainable AI Research Unit , Parc Mediterrani de la Tecnologia (PMT), Building B4, Av. Carl Friedrich Gauss 7, Castelldefels, 08860, Catalunya, Spain*

Abstract

This work introduces the Unscented Genetic Algorithm (U-GA), which combines ideas from evolutionary computation and Kalman filters to devise a novel approach to solve GA-hard problems. The approach is justified based on how other Bayesian methods make strong assumptions on data, which could limit their performance in the long run. U-GA applies theory from unscented Kalman filters to relax this assumptions via Monte-Carlo simulation. The algorithm is explained in detail, showing how unscented Kalman filters equations could be adapted for the evolutionary computation framework. In the experiments, the proposed approach is compared to Bayesian optimization algorithm (BOA) and genetic algorithms (GAs) to investigate the strengths and limitations of U-GA. The results show how U-GA attains better performance than the benchmarks, even when the problem size is increased. Also U-GA attained a considerable speed-up (around 400%) when compared with similar methods.

Keywords: Competent evolutionary algorithms, Unscented Kalman filters

1. Introduction

Genetic algorithms are methods inspired by both genetics and Darwinian evolution. They have been mainly used for optimization where traditional approaches usually struggled. For example, it has been reported in the literature the success of GAs for combinatorial optimization [1], network design

Email address: aaguilar@cttc.es (Anton Aguilar-Rivera)

[2], identification of the flux linkage map of permanent magnet synchronous machines [3], and others.

GAs solve a problem by processing a set of possible solutions that are encoded in the form or binary strings. In this context, each solution represents individuals from a population that compete with each other to prevail in future generations. Individuals are selected based on their fitness using an objective function. Then they are combined with each other using genetic operators, like crossover and mutation, to obtain the final offspring. Several variations of these operators could be found in the literature [4]. Repetition of this cycle (i.e. a generation) is expected to lead the population towards optimal solutions.

Although the potential of GAs was clearly reported in the literature, an explanation about their inner workings remained elusive. It was not clear why that particular combination of operations (i.e. selection, crossover, and mutation) could guide the search towards global optima even under adverse conditions like non-linearity or noise [5]. Further research was made about defining GA-hard problems, introducing the concepts of building blocks (i.e. schema)[6], deception [7], and linkage [8]. It was concluded GAs worked assembling building blocks and that linked, deceptive problems were the ones that could be considered GA-hard.

1.1. GA-hard Problems

Competent GAs are the ones that can solve GA-hard problems quickly, reliably and accurately, and GA-hard problems are the ones where deception and linkage are present [9]. Understanding the inner working of GAs is based on the concept of building blocks. GAs work on solutions encoded into binary strings, identifying those bits combinations that contribute to increase fitness values (assuming maximization). Then, they are passed to the next generation with higher probability than the rest of individuals. The success of GAs could be explained from the fact they process all the building blocks contained into a single binary string at the same time, speeding-up the search. This is known as implicit parallelism.

In this sense, GAs perform decomposition of the problem to find the good building blocks that will be later assembled together to conform the optimal solution [9]. Therefore, a GA-hard problem is the one where the solution is composed by building blocks that are hard to find. Deception refers to the case where the objective function leads the search consistently away from the optimum. Figure 1 shows an example of this kind of problem. On the

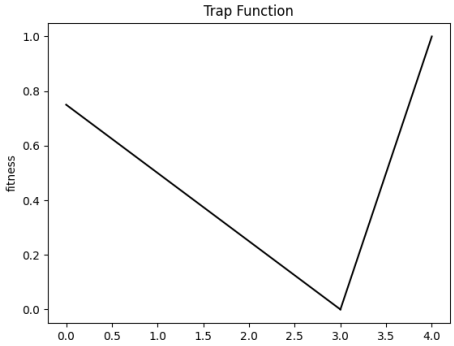

Figure 1: The figure shows and example of fully deceptive problem. Trap function with string length $b = 4$ and $k = 0.25$. The optimum is 1111, but the problem leads the algorithm towards 0000.

other hand, linkage refers to the case where the bits that form one building block are split along the string, making hard for the GA to decompose the problem properly. Both conditions prevent GAs to find good building blocks efficiently.

1.2. Competent GAs

We can find different approaches to tackle GA-hard problems in the literature. Goldberg an Ohsawa [9] mentioned three groups: Perturbation techniques, linkage adaptation techniques, and estimation of distribution techniques (EDAs). Besides, in the literature has been also reported model identification techniques [10]. The Fast Messy GA (fmGA) is an example a competent GA using perturbation techniques, while the Linkage Learning GA (LLGA) applies linkage adaptation; the Bayesian Optimization Algorithm (BOA) is probably the most known EDA, which implements a Bayesian network to model the population.

In general, all the approaches mentioned above understand traditional genetic operators disrupt the formation of linked building blocks, and propose solutions to preserve them along generations.

1.3. Revisiting the Bayesian Approach

The objective of this work is introducing a new type of EDA. Assuming normality, we could represent the population's bit distribution using

$$\boldsymbol{\mu} = \left[\mu_0, \mu_1, \cdots, \mu_m\right], \tag{1}$$

and

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & \cdots & \sigma_{0m} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{m0} & \sigma_{m1} & \cdots & \sigma_{mm} \end{bmatrix}, \tag{2}$$

where $\boldsymbol{\mu}$ is a vector with the mean of each variable and $\boldsymbol{\Sigma}$ is the co-variance matrix. In EDAs, this distribution is sampled to create the next generation. This allows discovering new individuals that preserve the relationships between bits while avoiding linkage problems.

BOA [11] (and hBOA [12]) use a Bayesian network to model relations between bits. A new Bayesian network is created each generation. The network is built based on maximization of some metric. One possibility is using the Bayesian Dirichlet metric [11]:

$$\begin{gathered} p\left(D, B|\xi\right) = p\left(B|\xi\right) \prod_{i=0}^{n-1} \prod_{\pi_{xi}} \frac{\Gamma(m'(\pi_{Xi}))}{\Gamma(m(\pi_{Xi})+m'(\pi_{Xi}))} \cdot \\ \prod_{\pi_{xi}} \frac{\Gamma(m(x_i,\pi_{Xi})+m'(x_i,\pi_{Xi}))}{\Gamma(m'(x_i,\pi_{Xi}))}. \end{gathered} \tag{3}$$

Equation 3 computes the probability the current population D fits the network. $p\left(D, B|\xi\right)$ represents the probability described by the network, which was created given prior information ξ. Here $= p\left(B|\xi\right)$ represents the prior distribution, which is represented by the network created the last generation. $m(\pi_{Xi})$ is the number of instances in D where the parents of bit X_i are instantiated to the combination Π_{X_i}. $m\left(x_i, \pi_{Xi}\right)$ denotes the number of instances where a specific combination of Π_{X_i} has set X_i. Besides, $m(\cdot)$ and $m'(\cdot)$ refer the current and the past network, respectively. This equation makes a comparison between B and the last network B'. When B is able to cover more instances of D, $p(D, B|\xi)$ will be increased. The best network maximizes $p(D, B|\xi)$. Once B is defined, it is sampled to obtain the next generation of individuals. Besides using a Bayesian network to represent data, equation 3 performs a belief update, which incorporates the information gathered from the selection process about the optimum.

1.4. Room to new Bayesian EDAs

Although, even when BOA has proved to be successful in solving GA-hard problems, there is still room for improvement. We recall the Bayes'

theorem equation:

$$P(R|S) = \frac{P(S|R)P(R)}{P(S)}, \tag{4}$$

where $P(R)$ which is the prior distribution, and $P(R|S)$ is the posterior distribution, which incorporates information from past beliefs and current data. $P(S)$ could be treated as a normalization factor without the loss of generality. $P(S|R)$ represents the likelihood S is represented by R. The fact the posterior probability is increased or decreased by the information in S depends of the ratio $\frac{P(S|R)}{P(S)}$.

This work's hypothesis is based on the fact equation 3 is only an approximation to equation 4 [13]. The approximation comes at the moment we assume data is truly generated by a Bayesian network, following a Dirichlet distribution. Besides, in the beginning of the run, an initial network structure (normally a non-connected set of nodes) should be assumed as well to start the optimization process, even when no information is available at the moment. These assumptions have allowed the development of algorithms like BOA, but if a Bayesian EDA could be devised without relying on these conditions, this new algorithm could be a better realization of equation 4 and possibly attain better performance than other EDAs.

One second improvement opportunity comes from the fact BOA requires high computation power to work. It is reported in the literature building Bayesian networks are a NP-hard problem [13], making unfeasible performing a full optimization process to search the best Bayesian network for each generation. This occurs because the algorithm must search and evaluate all the feasible candidates to find the new network, and perform this operation at each generation. Therefore, a method with the ability to implement the Bayesian approach without using Bayesian networks should speed-up over-all computation.

Based on this analysis, this work introduces Unscented Genetic Algorithms (U-GAs), which combines Kalman filters to GAs to implement a new Bayesian EDA. A description of U-GA is presented in section 2. Experiments are found in section 3 and the results are discussed in section 4. Section 5 is the conclusion. Final work is described on section 6.

2. Unscented Genetic Algorithms (U-GAs)

U-GAs are an implementation of Bayes' theorem based on Kalman filters (KFs) [14]. Kalman filter is a recursive algorithm that implements Bayes'

theorem and it is used for estimation of noisy data, forecasting, smoothing, estimation of unobserved variables, and other applications. In general, KFs perform two basic operations: prediction of prior distribution, and update of the prior distribution with information from data to obtain the posterior distribution. In the context of KFs, prediction is implemented with the following equations:

$$\bar{\boldsymbol{\mu}} = \boldsymbol{F}\boldsymbol{\mu} + \boldsymbol{B}\boldsymbol{w} \tag{5}$$

$$\bar{\boldsymbol{\Sigma}} = \boldsymbol{F}\boldsymbol{\Sigma}\boldsymbol{F}^{\boldsymbol{T}} + \boldsymbol{Q} \tag{6}$$

and update operation is described by

$$\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\bar{\boldsymbol{\mu}} \tag{7}$$

$$\boldsymbol{K} = \bar{\boldsymbol{\Sigma}}\boldsymbol{H}^{\boldsymbol{T}}\left(\boldsymbol{H}\bar{\boldsymbol{\Sigma}}\boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{R}\right)^{-1} \tag{8}$$

$$\boldsymbol{\mu} = \bar{\boldsymbol{\mu}} + \boldsymbol{K}\boldsymbol{y} \tag{9}$$

$$\boldsymbol{\Sigma} = \left(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H}\right)\bar{\boldsymbol{\Sigma}} \tag{10}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ were already introduced in equations 1 and 2. In equations 5 and 6, $\boldsymbol{F}$ is called the transition matrix, which usually holds a set differential equations that models the evolution of the system state with time. $\boldsymbol{w}$ models manipulations to the system, and $\boldsymbol{B}$ maps this manipulation to system states. These equations deal with prediction of the prior distribution.

Equations 7 to 10 describe the update operation. $\boldsymbol{z}$ represents data and $\boldsymbol{y}$ is the difference between data and the value predicted by equation 5. $\boldsymbol{K}$ is know as the Kalman gain. Equations 9 and 10 update both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ given the likelihood of data and uncertainty's of the state. $\boldsymbol{Q}$ and $\boldsymbol{R}$ represent our estimation of uncertainty in the model and in the measurements, respectively.

2.1. Prediction in U-GAs

Any algorithm implementing equation 4 must perform both prediction and update operations. The problem is that in GAs we have not a definition of the matrices used in KFs. Although, in the case of prediction, we could assume the selection operator is naturally moving the population towards the optimum. Then, we could express this assumption in the following equation:

$$\mathcal{S}\left(\mathrm{pop}_t\right) \propto \mathcal{F}\left(\mathrm{pop}_t\right), \tag{11}$$

where $\mathcal{S}$ represents the selection operator and $\mathcal{F}$ is a function applying $\boldsymbol{F}$ to the equivalent system represented by the population at time t. Under this

assumption, we can see the objective function provides information about the population dynamics and state transition occurs when the algorithm performs selection. The same assumption was used to explain the inner workings of BOA in subsection 1.3.

Therefore, we can implement equations 5 and 6 with

$$m_{10} = \operatorname{sgn}\left(\operatorname{sgn}\left(f_{D_s} - f_{D'}\right) + \mathbf{1}\right), \tag{12}$$

$$m_{01} = \mathbf{1} - m_{10}, \tag{13}$$

$$D = D_s \odot m_{10} + D' \odot m_{01}, \tag{14}$$

$$f_D = f_{D_s} \odot m_{10} + f_{D'} \odot m_{01}. \tag{15}$$

Equations 12 to 15 represent tournament selection. We are assuming maximization without the loss of generality. D and D' are the current and past generation's populations, respectively, while D_s is a new sample population used for the tournament. m_{10} and m_{01} are masks that can be either 0 or 1, depending on which individual has better fitness values. f_D, f_{D_s}, and $f_{D'}$ are the vectors with the fitness values of individuals in each sample. Operator $\odot$ represents element-wise multiplication.

2.2. Update in U-GAs

In a similar manner, fitness values could be regarded as measurements, and this information could be incorporated to obtain the posterior distribution (i.e. equations 1 and 2). To achieve this task we can recall to the update version of unscented Kalman filters (UKFs) [15][16]. In UKFs, a Monte-Carlo approach is used to compute the prediction and update operations, but it uses a carefully chosen set of points (i.e. sigma points) that guarantees a good parameters estimation while using very few values. This approach was chosen because it naturally allows computing $\boldsymbol{K}$ from a set of points, which is well suited for evolutionary algorithms. Update operation in UKFs is implemented in the following equations [15][16]:

$$\boldsymbol{\mathcal{Z}} = h\left(\boldsymbol{\mathcal{Y}}\right) \tag{16}$$

$$\boldsymbol{P}_{xz} = \sum \left(w^c \boldsymbol{\mathcal{X}} - \mu\right)\left(\boldsymbol{\mathcal{Z}} - \mu_z\right) \tag{17}$$

$$\boldsymbol{K} = \boldsymbol{P}_{xz}\boldsymbol{P}_z^{-1} \tag{18}$$

$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu_x} + \boldsymbol{K}\boldsymbol{y} \tag{19}$$

$$\bar{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}\boldsymbol{P}_z\boldsymbol{\Sigma}^T \tag{20}$$

Where $\boldsymbol{\mathcal{X}}$ denote the sigma points used by UKFs. $\boldsymbol{\mathcal{Y}}$ are the measurements, while $\boldsymbol{\mathcal{Z}}$ is their transformation to the states-space. $\boldsymbol{P}$ is used here to denote cross-covariance. In the case of U-GAs, individuals could be used as sigma points, and we could combine equations 16, 17, 18 and re-write them to be

$$K = \frac{1}{nb\sigma_f^2}\mathbf{1}\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mu_x}\right)^T \boldsymbol{f}. \tag{21}$$

In equation 21, $\boldsymbol{\mathcal{X}}$ is being used as a short-hand to refer to the population. $\boldsymbol{\mu_x}$ denotes a vector with the mean values for each column of $\boldsymbol{\mu_x}$. $\boldsymbol{f}$ is a vector with fitness values. n and b refers to population size and bit-string length, respectively, while σ_f^2 is the population's fitness variance. $\mathbf{1}$ is a vector of ones of suitable size to allow the computation. In the end, $\boldsymbol{K}$ is a scalar.

2.3. U-GA Implementation

U-GA keeps information of the population's distribution in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The population $\boldsymbol{\mathcal{X}}$ is sampled from a normal multi-variate distribution:

$$\boldsymbol{\mathcal{X}} \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right). \tag{22}$$

And then it is converted to bits using

$$\boldsymbol{\mathcal{X}_b} = \frac{\operatorname{sgn}\left(\boldsymbol{\mathcal{X}} - \boldsymbol{\mu}\right) + \mathbf{1}}{2}. \tag{23}$$

Equation 23 assigns 1 to those values above the mean and 0 otherwise. Both prediction and update operations described above are applied to $\boldsymbol{\mathcal{X}}$ instead of $\boldsymbol{\mathcal{X}_b}$. Although, $\boldsymbol{\mathcal{X}_b}$ is used for evaluation.

Implementation of U-GA is shown in figure 2. The input arguments are the objective function obj_fun, number of generations m, number of individuals n, and string length b. Both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can be initialized to uninformed states. The algorithm's loop is not different from the one found in regular GAs. The population is generated from the distribution defined by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using equations 22 and 23. Then, the population $\boldsymbol{\mathcal{X}_b}$ is evaluated using the objective function. PREDICT() implements the procedure described in

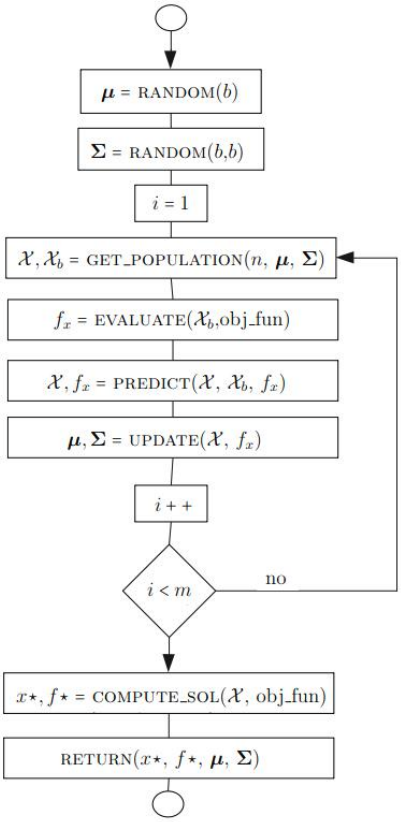

Figure 2: U-GA Flow Diagram.

subsection 2.1, which uses tournament selection to find the new state of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. UPDATE() implements the procedure described in subsection 2.2, which applies equations 16 to 20 update $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with the data from new evaluations. Finally, the algorithm returns $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with its estimation of the posterior distribution.

The optimal solution $x\star$ and its fitness $f\star$ is computed from the final population $\mathcal{X}$ using the COMPUTE_SOL() function. This function takes an average of each bit and evaluates the resulting individual. $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ could be regarded as metrics of the certainty the algorithm has about the result. Other criteria could be implemented as well.

3. Description of Experiments

Benchmark functions were taken from the literature [17] [18] to test the proposed approach. Five of them were chosen to be included in this article. In table 1 below, the names and properties of these functions are described. The problems could be classified whether they are (or are not) continuous, differentiable, separable, scalable and unimodal. The problems were chosen to explore the performance of algorithms under different conditions. It is expected non-separable problems to be the hardest ones because of linkage.

Other evolutionary algorithms are considered in the experiments for comparison purposes. BOA was chosen because both U-GA and BOA work under similar principles. Also, a regular GA was chosen because it is expected to be the lower bound in performance. Any competent evolutionary algorithm must perform significantly better than a regular GA for hard problems.

All three algorithms were run with the following setup: $m = 100$, $n = 200$, and $b = 16$. Only Dixon-Price problem was run using also $b = 24$ (more details in section 4). In the experiments, the data of 100 runs were collected for each algorithm.

Problem	Continuous	Differentiable	Separable	Scalable	Unimodal
One-max	No	No	Yes	Yes	Yes
Dixon-Price	Yes	Yes	No	Yes	Yes
Schwefel 1.2	Yes	Yes	No	Yes	Yes
Stepint	No	No	Yes	Yes	Yes
Type-1 Dec.	No	No	No	Yes	No
Type-2 Dec.	No	No	No	Yes	No

Table 1: Properties of benchmark functions used in the experiments.

4. Results

The results are shown in figures 6 to 11 and in table 2. The figures are presented in pairs. The first one shows the RMS error between the population's fitness mean and the optimum result of the problem. This is possible because the optimal solutions of all the benchmarks are known beforehand. They are obtained by aggregation of 100 runs. Also, this figure shows the evolution of fitness with time. The second figure shows a box plot of the speed-up attained by U-GA and GA in relation to BOA. BOA was

chosen as the benchmark because the other algorithms were faster. Speed-up is computed in the following manner:

$$\text{speed-up}() = \frac{t_{\text{new}}}{t_{\text{old}}} \times 100\% \tag{24}$$

Times were chosen to show speed-up because all three algorithms are of order $\mathcal{O}(n)$. Therefore the difference in running time should be caused by other steps in the algorithms. For example, the construction of the Bayesian network seems to be the main cause of time-consumption in BOA to the point it could be not considered negligible. One of the advantages of using relative comparison metric like the one shown in equation 24 is the division cancels the effect of similar conditions applied during the runs (e.g. specific hardware used). Therefore speed-ups seemed a suitable way to report performance when comparing disparate algorithms or computer architectures.

In table 2, the final mean and standard deviation of RMS between population fitness and the optimum is reported in the form mean/standard deviation. The results of 100 runs were aggregated to compute these values.

In table 2 we can see the convergence of all the algorithms. We can see both U-GA and BOA were able to solve the benchmarks, while GA struggled. This result is expected and in clearly shows the proposed approach is better than a regular GA. No increments of n or b were needed for further clarification.

We can observe the final mean and standard deviation of BOA for Dixon-Price-16 problems are better than U-GA's but the situation reverses for the 24-bits case. In general, it was observed that for small values of b (e.g. $b \approx 8$) there was not significant difference between the convergence power of BOA and U-GA, but the performance of BOA degrades with increasing values of b. In figures 6, 8, 10, and 11, we can see U-GA was able to find the optimum faster than BOA.

One interesting result is shown in figure 11. This problem is a pair of concatenated traps as the one shown in figure 1. We can see a clear difference in performance between U-GA and BOA. The solution could be increasing the number of individuals, and the number of generations to allow BOA having more information to process, but in the end, this would lead to a slower performance because BOA would require more evaluations than U-GA to find the optimum. We can see in the box plots the median in speed-up is around 500%. In this sense, we could say U-GA dominates BOA when

considering both RMS and speed-up for the performed experiments. This trend is clearer for the type-2 deceptive problem case.

Another point to be discussed in regards the results is it could be observed both U-GA and BOA performed better for discontinuous problems than continuous ones. This could be explained because both algorithms are designed to work with bit strings. Therefore, a mechanism to perform continuous optimization would be needed to increase performance. Different solutions could be found in the literature [19].

Finally, the box plots in figures 6 to 11 shows both GA and U-GA are faster than BOA. GA is proven to be fastest one , because is the most simple algorithm, but even BOA was able to attain speed-ups up to 3000% in the experiments. Also, GA could be tuned up to run for longer numbers of generations and individuals, but in the end, this would have negative impact on performance. It seems the results show both U-GA and BOA outperform a GA for hard problems.

Problem	U-GA	BOA	GA
One-Max	0/0	0/0	0.194/0.0378
Dixon-Price-16	0.146/0.149	0.026/0.036	724.533/458.2763
Dixon-Price-24	0.131/0.119	0.322/0.262	804.502/415.498
Schwefel-1.2	0.154/0	0.166/0.067	138.731/77.204
Stepint	0/0	0/0	0.316/0.071
Type-1 Deceptive	0/0	0.089/0.029	0.0418/0.032
Type-2 Deceptive	0/0	0.529/0.022	0.028/0.006

Table 2: Final Averages and Standard Deviations of RMS Values for Studied Algorithms and Benchmarks.

U-GA's behavior is further analyzed in figures 3, 4, and 5; where some examples of the evolution of μ and Σ along the algorithm execution are shown. The plots were obtained computing the average RMS of the elements in those matrices. The plots show how the RMS values stabilize themselves along the run, showing the convergence of the population towards the optimal solution.

5. Conclusion

This work introduced U-GA, a competent evolutionary algorithm based on UKFs. The introduction made a brief discussion about hard-GA problems

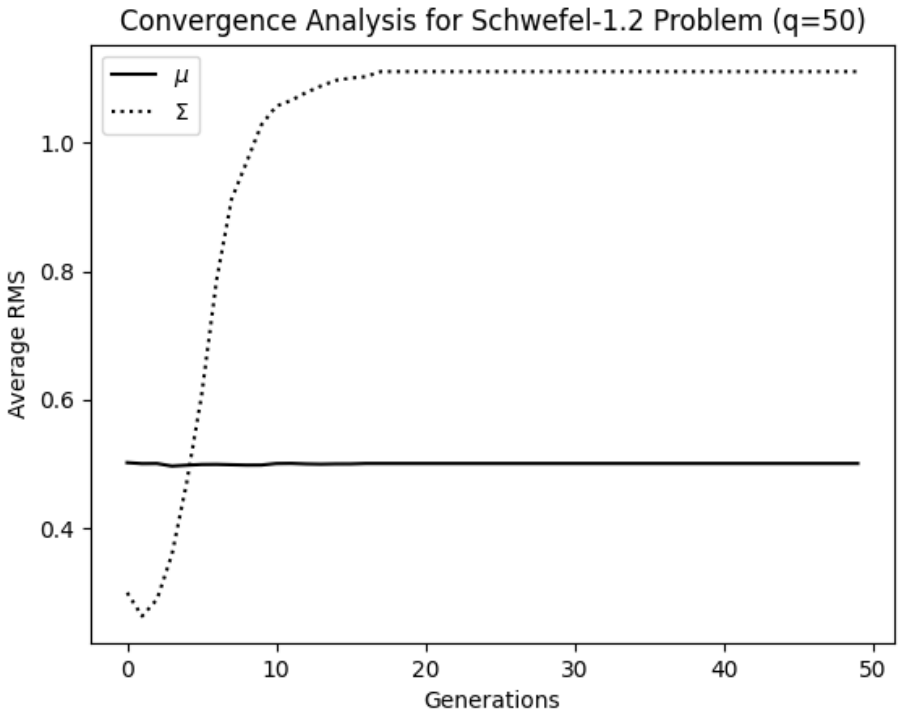

Figure 3: Convergence of U-GA Example.

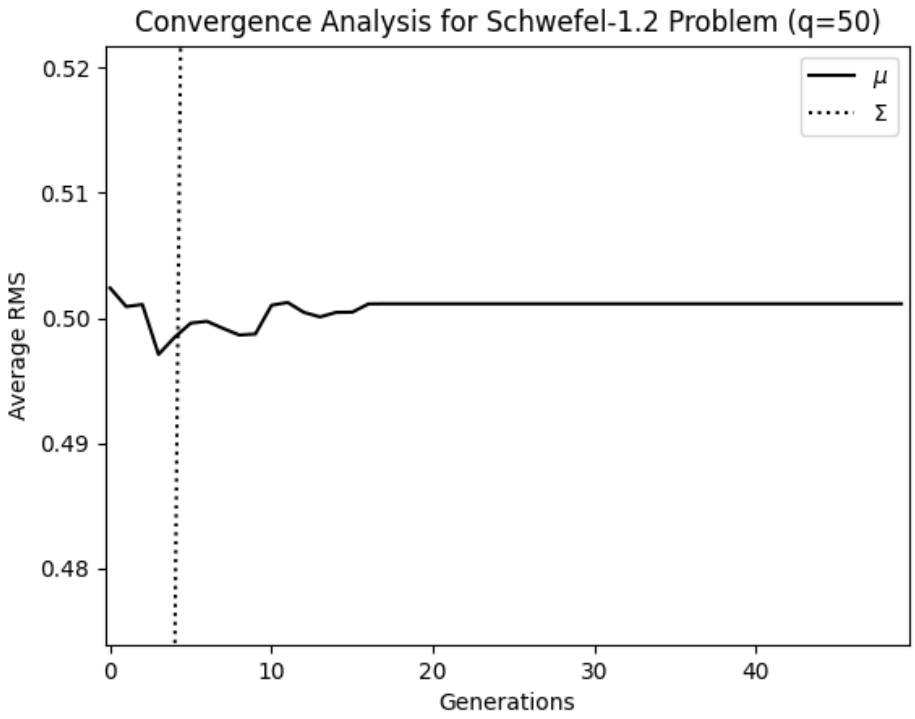

Figure 4: Convergence of U-GA Example (Detail).

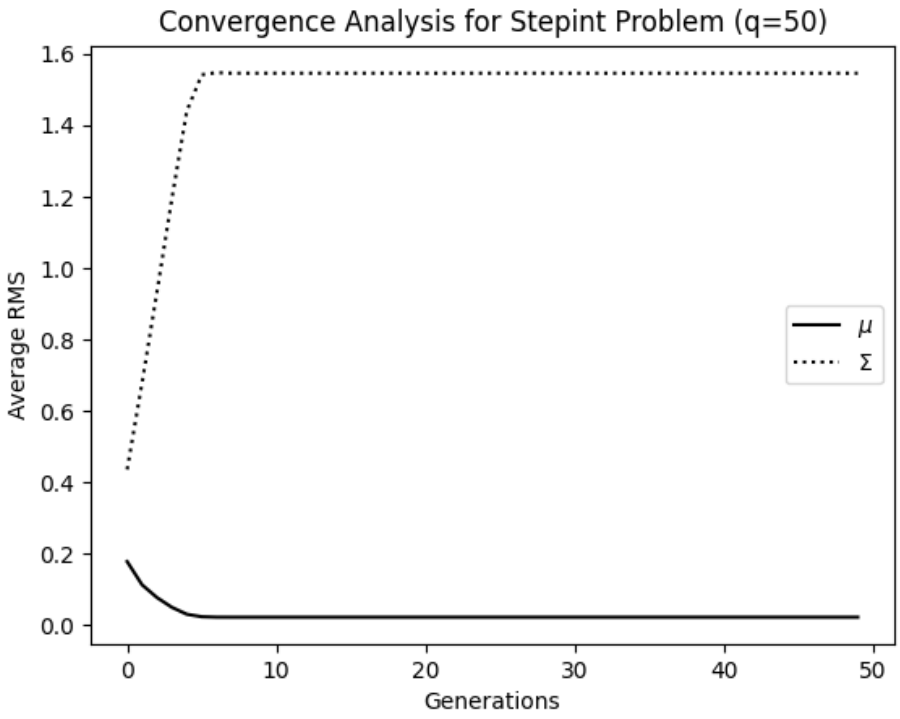

Figure 5: Convergence of U-GA Example.

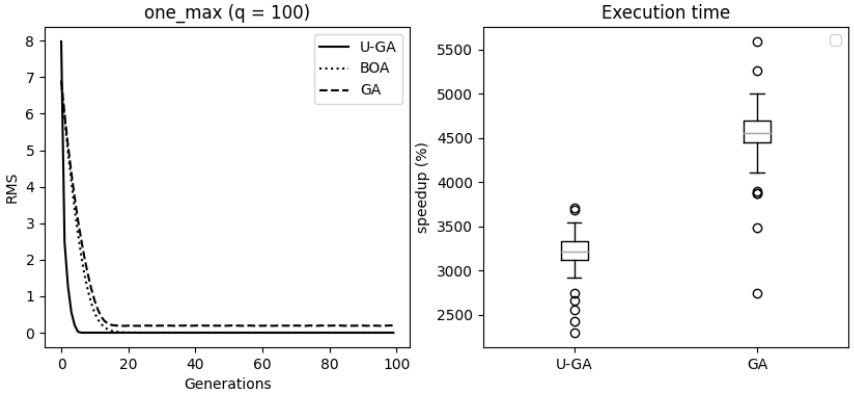

Figure 6: Results for one-max problem.

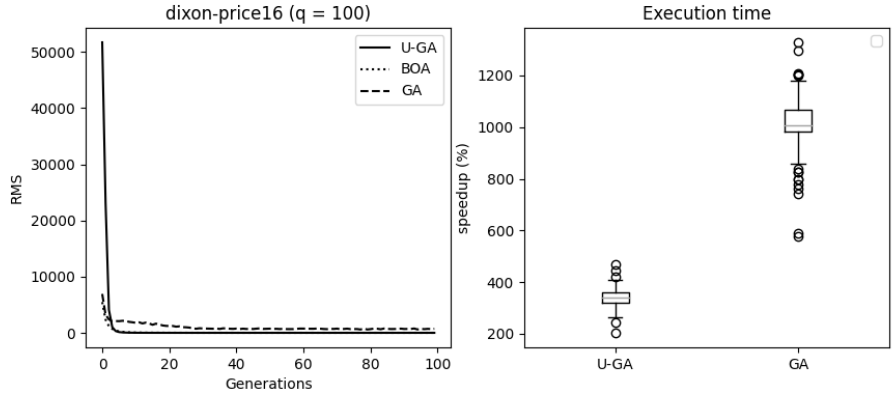

Figure 7: Results for Dixon & Price problem (16-bits).

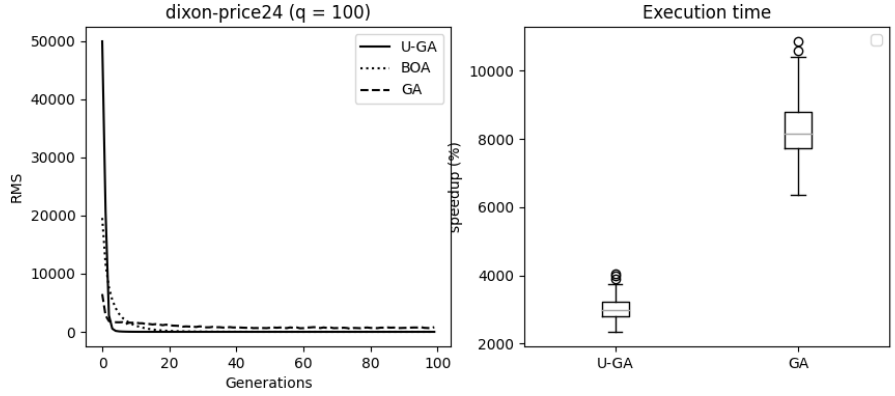

Figure 8: Results for Dixon & Price problem (24-bits).

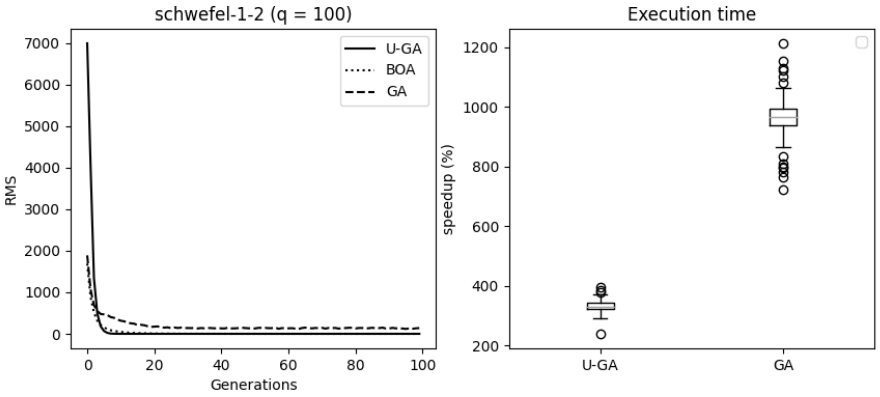

Figure 9: Results for Schwefel 1.2 Problem.

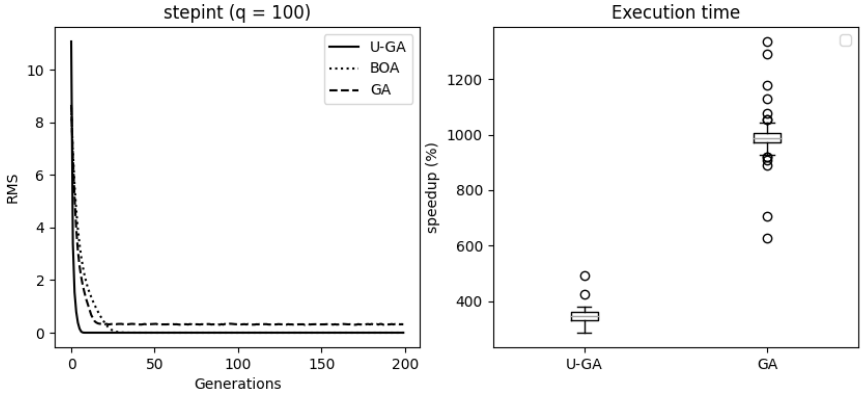

Figure 10: Results for Stepint Problem.

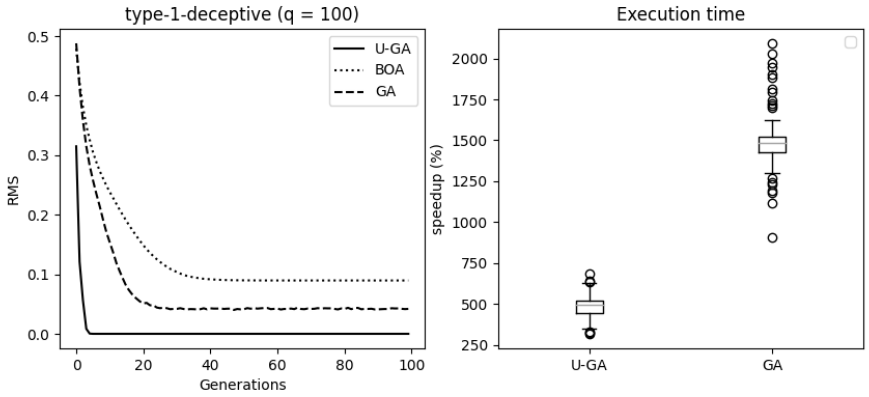

Figure 11: Results for Type-1 Deceptive Problem.

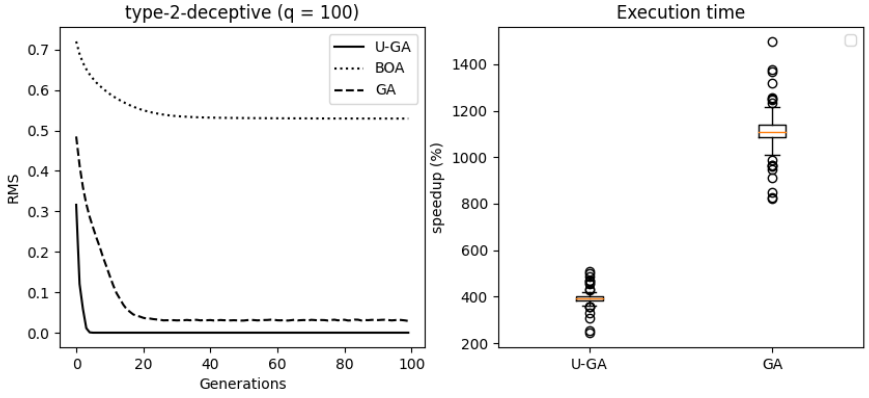

Figure 12: Results for Type-2 Deceptive Problem.

and their relation to deception and linkage. Unscented Kalman filters were a good starting point to create U-GA because it naturally integrates Kalman filtering theory with Monte-Carlo methods, a combination that could be easily extended to evolutionary algorithms.

U-GA was tested using different benchmark problems proposed in the literature and compared using other evolutionary algorithms (GA and BOA). It was shown it could be outperform a regular GA at tackling hard problems and it could attain better results than BOA for larger problems. It seems BOA would require larger populations and generations to attain better results.

The difference in performance could be explained based on the idea some EDAs (e.g. BOA) make strong assumptions about the distribution of data. For example, Bayesian networks assume a Dirichlet distribution. On the other hand, UKFs approach was devised to overcome these restriction, allowing KFs to be applied to non-linear problems applying Monte-Carlo methods. Although, we were not able to find a completely agnostic method in regards of probability distributions, U-GA represent and advancement in this direction, which could be the subject of further research.

6. Future Work

We could further inquire on the performance of U-GA considering it is fundamentally based on UKFs. It is known UKFs perform better than regular KFs for non-linear problems, but even UKFs would struggle to track very fast moving objects (e.g. a fighter jet). Therefore it could be expected U-GA to struggle when facing non-stationary problems. Also, other approaches could be investigated. For example, extended KFs (EKFs) are more widely used to deal with non-linear problems than UKFs. Also, comparison against other type of algorithms and application to real-world problems are worth to be explored in the future.

Finally, devising U-GA comes from the combination of two frameworks that appear unrelated (i.e. evolutionary algorithms and KFs). It could be interesting further combinations to create new algorithms. One of the advantages of this approach is the theoretical framework for the new algorithm would be already created. For example, we could assume U-GA would be competent-GA because UKFs are an already tested method. Besides, we could take advantage of the already developed theory from the base frameworks to justify the new method and explain its behaviour.

Acknowledgements

This work was made as part of the author's research activities at Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Sustainable Artifitial Intelligence (SAI) research unit.

References

[1] C. R. Reeves, Genetic algorithms, in: Handbook of metaheuristics, Springer, 2010, pp. 109–139.

[2] M. Tsuji, M. Munetomo, K. Akama, A network design problem by a GA with linkage identification and recombination for overlapping building blocks, in: Linkage in Evolutionary Computation, Springer, 2008, pp. 441–459.

[3] K. Liu, J. Feng, S. Guo, L. Xiao, Z.-Q. Zhu, Identification of flux linkage map of permanent magnet synchronous machines under uncertain circuit resistance and inverter nonlinearity, IEEE Transactions on Industrial Informatics 14 (2) (2017) 556–568.

[4] A. J. Umbarkar, P. D. Sheth, Crossover operators in genetic algorithms: a review., ICTACT journal on soft computing 6 (1) (2015).

[5] E. Cantú-Paz, Adaptive sampling for noisy problems, in: Genetic and Evolutionary Computation Conference, Springer, 2004, pp. 947–958.

[6] J. H. Holland, et al., Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1992.

[7] D. Goldberg, Simple genetic algorithms and the minimal, deceptive problem, Genetic Algorithms and Simulated Annealing (1987) 74–88.

[8] M. Tsuji, M. Munetomo, Linkage analysis in genetic algorithms, in: Computational Intelligence Paradigms, Springer, 2008, pp. 251–279.

[9] D. E. Goldberg, K. Sastry, Y. Ohsawa, Discovering deep building blocks for competent genetic algorithms using chance discovery via keygraphs, in: Chance Discovery, Springer, 2003, pp. 276–301.

[10] M. Valenzuela-Rendón, Black-box optimization by deterministic identification: The BODI report, Tech. rep., BODI Report (2016).

[11] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, Bayesian optimization algorithm, population sizing, and time to convergence., in: GECCO, 2000, pp. 275–282.

[12] M. Pelikan, D. E. Goldberg, Escaping hierarchical traps with competent genetic algorithms, in: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, 2001, pp. 511–518.

[13] D. M. Chickering, D. Geiger, D. Heckerman, et al., Learning bayesian networks is np-hard, Tech. rep., Citeseer (1994).

[14] A. Akca, M. Ö. Efe, Multiple model kalman and particle filters and applications: a survey, IFAC-PapersOnLine 52 (3) (2019) 73–78.

[15] S. J. Julier, The scaled unscented transformation, in: Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301), Vol. 6, IEEE, 2002, pp. 4555–4559.

[16] R. Labbe, Kalman and bayesian filters on python, `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python` (2020).

[17] M. Jamil, X.-S. Yang, A literature survey of benchmark functions for global optimization problems, arXiv preprint arXiv:1308.4008 (2013).

[18] A. R. Al-Roomi, Unconstrained Single-Objective Benchmark Functions Repository (2015).
URL `https://www.al-roomi.org/benchmarks/unconstrained`

[19] A. Agapie, M. Agapie, G. Zbaganu, Evolutionary algorithms for continuous-space optimisation, International journal of systems science 44 (3) (2013) 502–512.