Self-adaptive Search Equation-Based Artificial Bee Colony Algorithm with CMA-ES on the Noiseless BBOB Testbed

Doğan Aydin  
Dumlupınar University  
Computer Engineering Dept. 43000  
Kütahya, Turkey  
dogan.aydin@dpu.edu.tr

Gürcan Yavuz  
Dumlupınar University  
Computer Engineering Dept. 43000  
Kütahya, Turkey  
gurcan.yavuz@dpu.edu.tr

ABSTRACT
Self-Adaptive Search Equation based Artificial Bee Colony (SSEABC) is a recent variant of Artificial Bee Colony (ABC) algorithm. SSEABC proposed three enhancements on the canonical ABC algorithm. These are the self-adaptive search equation selection strategy, hybridization with a local search procedure and incremental population size strategy. The performance of SSEABC is tested on CEC 2015 benchmark suite and ranked third within all participants of competition. In this paper, we benchmark SSEABC using the noise-free BBOB function testbed. We also compare SSEABC performance to PSO, ABC and GA algorithms.

CCS CONCEPTS
• Mathematics of computing → Probabilistic algorithms; • Theory of computation → Mathematical optimization; Random search heuristics; Theory of randomized search heuristics; • Computing methodologies → Search methodologies;

KEYWORDS
Artificial Bee Colony, Continuous Domains, Self-Adaptation, Benchmarking

ACM Reference format:
Doğan Aydin and Gürcan Yavuz. 2017. Self-adaptive Search Equation-Based Artificial Bee Colony Algorithm with CMA-ES on the Noiseless BBOB Testbed. In Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017, 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084204

1 INTRODUCTION
Ever since the Artificial Bee Colony (ABC) algorithm came into existence [11], it has been used in solving continuous optimization problems. However, failure to produce successful results in some types of problems has led to the emergence of many improved ABC variants in recent years. Many of these algorithms have suggested enhancements over one or more of the steps of the ABC algorithm [1, 3]. A recent research [1] has shown that the best improvements can be made with changes to the employed bees and onlooker bees steps or with new extensions to the canonical ABC algorithm.

A recent ABC variant, Self-adaptive Search Equation based Artificial Bee Colony (SSEABC) [14], is focused on these remediation methods. The SSEABC algorithm solves the problem of finding the appropriate search equation in the employed bees and onlooker bees steps in a self-adaptive way. On the other hand, the algorithm has been improved with iteratively increasing the number of populations and using local search procedures.

SSEABC algorithm performance has been compared with ABC and many contemporary algorithms on CEC 2016 benchmark functions suite and it has been observed that we have obtained successful results [14]. In this paper, the performance of SSEABC algorithm on the BBOB functions testbed has been tested.

2 ALGORITHM PRESENTATION
SSEABC proposes three modifications on the original ABC algorithm to improve performance. These strategies are based on the self-adaptive search equation selection, hybridization with a local search procedure and increasing population size during execution. The pseudo-code of SSEABC is presented in Algorithm 1.

Self-adaptive search equation selection: In solving numerical optimization problems, the most important factor affecting the performance of ABC algorithm is the search equations that take place in the steps of employed bees and onlooker bees. In addition to the search equation, the number of dimensions considered to be changed is another important factor affecting the performance of the algorithm. When considering the structure of the problem and that is supposed to be solved; determining the appropriate search equation becomes a difficult task. Thus, in this study, a mechanism has been developed that determines the appropriate search equation among the various candidates. To do this, SSEABC has proposed a search equation pool which is filled with randomly generated search equations. The general template of candidate search equation is as seen in Algorithm 2.
used in the employed bees and onlooker steps, $ps$, which is the size of the pool, is scaled down by the equation 1:

$$ps = \frac{ps^2}{itr_{MAX}}$$

where $MAXFES$ is the maximum number of function evaluations for one execution and $2 \times SN$ is the number of function evaluations at each iteration and where $itr_{MAX}$ is the approximated value of the maximum number of iterations. $itr_{MAX}$ is the approximated value because the incremental population size strategy in which $SN$ is changing over time. Finally, when the algorithm finishes its execution, very few search equations, which are the appropriate ones, remain in the pool only.

**hybridization with a local search procedure.** In SSEABC algorithm, bees move using the SSEABC rules and by the invocation of a local search procedure. Specifically, best-so-far solution is used as the initial solution a local search procedure is called from. The final solution found through local search becomes the new best-so-far solution if it is better than the initial solution. In SSEABC, the local search procedure is not called every iteration. The local search procedure is called only when it is expected that its invocation will result in an improvement of the best-so-far solution. In previous implementation of SSEABC [14], competitive local search selection procedure was used. However, for the BBOB testbed, we used CMA-ES algorithm [12] as the local search procedure because competitive local search selection provides a wasteful use of function evaluations.

Table 1: Alternative options for each component in the generalized search equation. $x_{G,j}$, $x_{GD,j}$, $x_{SC,j}$, $x_{MD,j}$, $x_{WO,j}$ and $x_{AVE,j}$ are best-so-far, best-distance, second best, median, worst foods sources at dimension $j$, respectively. On the other hand, $x_{r1,j}$ and $x_{r2,j}$ are two randomly selected food source and $x_{AVE,j}$ refers to average positions of the food source at dimension $j$. $\phi_N$ can take two possible ranges: $[-1, -1]$ and $[-SF, SF]$ where $SF$ is randomly selected positive real value. These ranges are decided randomly while creating each component of randomly generated search equation.

| $M$ | $x_{i,j}$ | $x_{G,j}$ | $x_{r1,j}$ | $x_{SC,j}$ | $x_{MD,j}$ | $x_{WO,j}$ | $x_{AVE,j}$ | DoNotUse |
|-----|--------|--------|--------|--------|--------|--------|--------|--------|
| 1   |        |        |        |        |        |        |        |        |
| $k (1 \leq k \leq D)$ | $\phi_{N}(x_{i,j} - x_{G,j})$ | $\phi_{N}(x_{i,j} - x_{r1,j})$ | $\phi_{N}(x_{r1,j} - x_{G,j})$ | $\phi_{N}(x_{r1,j} - x_{SC,j})$ | $\phi_{N}(x_{r1,j} - x_{MD,j})$ | $\phi_{N}(x_{r1,j} - x_{WO,j})$ | $\phi_{N}(x_{r1,j} - x_{AVE,j})$ |        |
We have used the default parameter values for SSEABC and CMAES where $\bar{x}$ is 6 MB L2 cache and 8 GB RAM. The time per function evaluation continues until the maximum population value is reached. The maximum of 10 depends on a given target function value.

### RESULTS

Results from experiments according to [10] and [6] on the benchmark functions given in [5, 9] are presented in Figures 1, 2 and 3. The experiments were performed with COCO version 2.0, the plots were produced with version 2.0.

#### EXPERIMENTAL PROCEDURE

We have used the default parameter values for SSEABC and CMAES algorithms which were given in [14] and [12] respectively. A maximum of $10^4 D$ function evaluations was used. Every periodic 2500D function evaluations SSEABC restarts without forgetting the best-so-far solution.

#### CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the SSEABC on the f8 without restarts 30 seconds and until a maximum budget equal to $1000 D$ is reached. The C++ code was run on an Intel Xeon E5410 quadcore CPUs running at 2.33 GHz with 2 x 6 MB L2 cache and 8 GB RAM. The time per function evaluation for dimensions 2, 3, 5, 10, 20, 40 equals 0.0041, 0.0082, 0.0146, 0.627, 1.421, and 2, 751 seconds respectively.

#### 5 RESULTS

Comparison of SSEABC algorithm to PSO, ABC and GA in previous BBOB workshops are presented in Figure 2. We have seen that SSEABC outperforms PSO, ABC and GA for almost all functions. Moreover, SSEABC obtains better run-time performance than reference algorithms on the moderate, ill-conditioned and multi-modal functions. When the comparison results are examined, SSEABC for f4 and f20 seems to give bad results from ABC. Although SSEABC is an improved variant of the ABC algorithm, it is surprising at first glance that this situation has emerged. However, this is related to the fact that the entirely selected local search algorithm does not work well on these problems. The use of a certain amount of the function evaluations budget by CMA-ES yields this result.

#### 6 CONCLUSION

In this paper, we present the benchmark results of SSEABC algorithm on BBOB functions tested. We have also compared the performance of SSEABC to the data obtained by PSO, ABC and GA algorithms. The comparison results showed that SSEABC algorithm can outperform the compared algorithms and it is very competitive to (1+1)-CMA-ES and BIPOP-CMA-ES in moderate and ill-conditioned functions.

### REFERENCES

[1] Doğan Aydın. 2015. Composite artificial bee colony algorithms: From component-based analysis to high-performing algorithms. Applied Soft Computing 32 (2015), 266–285.
[2] Doğan Aydın, Tianjun Liao, Marco A Montes de Oca, and Thomas Stützle. 2011. Improving performance via population growth and local search: the case of the artificial bee colony algorithm. In International Conference on Artificial Evolution (Evolution Artificielle). Springer, 85–96.
[3] Doğan Aydın, Gürcan Yavuz, and Thomas Stützle. 2017. ABC-X: a generalized, automatically configurable artificial bee colony framework. Swarm Intelligence 11, 1 (2017), 1–38.
[4] Marco A Montes de Oca, Doğan Aydın, and Thomas Stützle. 2011. An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re) design of optimization algorithms. Soft Computing 15, 11 (2011), 2233–2259.
[5] S. Finck, N. Hansen, R. Ros, and A. Auger. 2009. Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noisy Functions. Technical Report 2009/20. Research Center PPE. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.
[6] N. Hansen, A. Auger, D. Brockhoff, D. Tuia, and T. Tuia. 2016. COCO: Performance Assessment. ArXiv e-prints arXiv:1605.03560 (2016).
[7] N. Hansen, A. Auger, S. Finck, and R. Ros. 2012. Real-Parameter Black-Box Optimization Benchmarking 2012: Experimental Setup. Technical Report. INRIA. http://coco.gforge.inria.fr/bbob2012-downloads
[8] N. Hansen, A. Auger, O. Mersmann, T. Tuia, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. ArXiv e-prints arXiv:1603.08785 (2016).
[9] N. Hansen, S. Finck, R. Ros, and A. Auger. 2009. Real-Parameter Black-Box Optimization Benchmarking 2009: Noisy Functions Definitions. Technical Report RR-6829. INRIA. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.
[10] N. Hansen, T. Tuia, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. ArXiv e-prints arXiv:1603.08776 (2016).
[11] Devir Karaboga and Bahriye Basturk. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of global optimization 39, 3 (2007), 459–471.
[12] Václav Klement Petr Fovsik. 2012. Benchmarking the Differential Evolution with Adaptive Encoding on Noisy Functions. In GECCO 2012 (Companion). Terence Soule (Ed.). ACM.
[13] Kenneth Price. 1997. Differential evolution vs. the functions of the second ICPO. In Proceedings of the IEEE International Congress on Evolutionary Computation. 153–157.
[14] Gürcan Yavuz, Doğan Aydın, and Thomas Stützle. 2016. Self-adaptive search equation-based artificial bee colony algorithm on the CEC 2014 benchmark functions. In Evolutionary Computation (CEC), 2016 IEEE Congress on. IEEE, 1173–1180.
Figure 1: Average running time (aRT in number of $f$-evaluations as $\log_{10}$ value), divided by dimension for target function value $10^{-8}$ versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of $f_1$ and $f_2$. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: $\diamondsuit$: PSO, $\lozenge$: GA, $\circledast$: ABC, $\triangledown$: SSEABC
Figure 2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The “best 2009” line corresponds to the best aRT observed during BBOB 2009 for each selected target.
Figure 3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{-8..2}$ for all functions and subgroups in 20-D. The “best 2009” line corresponds to the best aRT observed during BBOB 2009 for each selected target.
Table 2: Average runtime (aRT in number of function evaluations) divided by the respective best aRT 2009 divided by the respective best. The median number of conducted function evaluations is additionally given in italics, if the target in the last column was never reached.

Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p < 0.05$ or $p = 10^{-k}$ where the number $k$ following the star is larger than 1, with Bonferroni correction of 110. A ↓ indicates the same tested against the best algorithm from BBOB 2009. Best results are printed in bold.
Table 3: Average runtime (aRT in number of function evaluations) divided by the respective best aRT measured during BBOB-2009 in dimension 20. The aRT and in braces, as dispersion measure, the half difference between 10 and 90%-tile of bootstrapped run lengths appear for each algorithm and target, the corresponding reference aRT in the first row. The different target Af-values are shown in the top row. #succ is the number of trials that reached the (final) target \( f_{opt} + 10^{-8} \). The median number of conducted function evaluations is additionally given in italics, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with \( p = 0.05 \) or \( p = 10^{-k} \) when the number \( k \) following the star is larger than 1, with Bonferroni correction of 110. A \( \dagger \) indicates the same tested against the best algorithm from BBOB 2009. Best results are printed in bold.

Data produced with COCO 2.1.7.