International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

New Approach of Self-Adaptive Simulated Binary Crossover-Elitism in Genetic Algorithms for Numerical Function Optimization

Adriana Fanggidae*[1], Muhammad Iqrom Catur Prasetyo[2], Yulianto Triwahyuadi Polly[3], Meiton Boru[4]

Submitted: 06/12/2023 **Revised:** 17/01/2024 **Accepted:** 27/01/2024

Abstract: One of the critical evolutionary operators in genetic algorithms (GAs) is crossover. Simulated Binary Crossover (SBX) is a commonly employed crossover operator in GA for real-valued encoding. Self-Adaptive SBX introduces a distribution index parameter that is updated in each generation, enabling the offspring solution distance to be independent of the parent solution distance. During evolution, the extinction of the fittest individuals is possible, and elitism is employed to prevent such extinction, thereby preserving the quality of the offspring. This research proposes GA with Self-Adaptive SBX-Elitism to enhance the performance of GA with Self-Adaptive SBX. The performance of GA with Self-Adaptive SBX-Elitism and GA with Self-Adaptive SBX is tested on ten benchmark functions. The test results on ten populations in dimensions ten, twenty, and thirty indicate that GA with Self-Adaptive SBX-Elitism can reduce the average relative error by 99.99%, with an average computation time that is 19.40% faster compared to GA with Self-Adaptive SBX. GA with Self-Adaptive SBX-Elitism performs well across twenty populations in all test dimensions.

Keywords: *Elitism, Genetic Algorithm, Real-Valued Encoding, Self-Adaptive, Simulated Binary Crossover*

1. Introduction

Global optimization problems are frequently encountered in everyday life. Complex global optimization problems cannot be effectively solved using conventional methods due to their lengthy computational time and various limitations [1]. Metaheuristics are efficient methods for solving complex optimization problems [2] that can rapidly generate solutions close to the global optimum and escape from local optima. Metaheuristic methods also incorporate learning strategies during the search process, making them more efficient in discovering the global optimum solution [3].

Genetic algorithms are one of the metaheuristic methods that adopt Darwin's theory of evolution, encompassing selection, crossover, and mutation. Encoding is the initial stage of genetic algorithms, which maps the phenotype space or solutions into the genotype space or code [4]. Binary encoding is the most commonly used encoding due to its simplicity. However, this encoding has limitations in the precision of the generated solutions, as the length of the binary string constrains it [5], and it requires longer computational time due to the need for solution variable conversion processes. Using real-valued encoding can provide a solution as it can work effectively without requiring conversion processes.

Simulated Binary Crossover (SBX) is a crossover method in real-valued encoding that adapts the single-point crossover in binary encoding [6], [7]. SBX is parent-centric, where two offspring values are generated around the parent values based on a probability distribution [8]. According to [9], SBX has been successfully applied to various optimization problems. In SBX, a parameter called the distribution index η controls the spread of offspring. This parameter is predetermined and remains constant during the solution search process. If the η value remains constant, the distance between offspring solutions is proportional to the distance between parent solutions. Thus, if two parent solutions are far apart, the offspring solutions will also be far apart, and vice versa [10]. It highlights the significance and criticality of determining this parameter. Deb, Sindhya, and Okabe [11] propose a self-adaptive procedure to update the value of η dynamically. Testing was conducted on three benchmark functions with one and two objective functions. The results showed that the proposed procedure outperformed the original SBX by finding better solutions.

The weakness of GA lies in its inability to maintain good parents; no matter how good their parent chromosomes are, they cannot be preserved [12]. Elitism in genetic algorithms is often utilized to preserve the quality of individuals within the population. It is because, during evolution, the extinction of the best individuals is possible,

[1] *Department of Computer Science, Faculty of Science and Engineering, Universitas Nusa Cendana, Indonesia*
ORCID ID : 0009-0002-8664-0190
[2] *Department of Computer Science, Faculty of Science and Engineering, Universitas Nusa Cendana, Indonesia*
ORCID ID : 0009-0005-8375-4266
[3] *Department of Computer Science, Faculty of Science and Engineering, Universitas Nusa Cendana, Indonesia*
ORCID ID : 0009-0008-7844-4580
[4] *Department of Computer Science, Faculty of Science and Engineering, Universitas Nusa Cendana, Indonesia*
ORCID ID : 0000-0001-9059-3863
* *Corresponding Author Email: adrianafanggidae@staf.undana.ac.id*

while they are expected to persist in each generation to yield better offspring. In the study conducted by Fanggidae et al. and Singh et al., the performance of GA and elitism GA was compared, and it was found that the use of elitism enhances the exploitation capability of GA in searching for optimal solutions [13], [14]. This research is focused on enhancing the performance of GA with Self-Adaptive SBX by incorporating an Elitism stage, referred to as GA with Self-Adaptive SBX-Elitism. Another objective is determining the optimal population size for GA with Self-Adaptive SBX-Elitism. Testing on ten benchmark functions was conducted to address these research objectives. The test results demonstrate that GA with Self-Adaptive SBX-Elitism can effectively reduce relative error, save computational costs, and perform well with a population of twenty.

2. Method

2.1. Benchmark Function

Ten benchmark functions are used in the testing [15], [16]. The global optimal solutions are denoted by x^*, where D represents the dimension of the solution, and $f(x^*)$ represents the fitness of the global optimal solution. The specific details of the benchmark functions are presented in Table 1.

Table 1. Minimization problem benchmark functions

Function name	Model	Search space	Global optimum	
			x^*	$f(x^*)$
Ackley Function	$f(x) = -20e^{-0.02\sqrt{D^{-1}\sum_{i=1}^{D} x_i^2}} - e^{D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)} + 20 + e$	-35 ≤ x_i ≤ 35	(0,..,0)	0
Alpine Function	$f(x) = \sum_{i=1}^{D} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	-10 ≤ x_i ≤ 10	(0,..,0)	0
Exponential Function	$f(x) = -\exp\left(-0.5\sum_{i=1}^{D} x_i^2\right)$	-1 ≤ x_i ≤ 1	(0,..,0)	-1
Griewank Function	$f(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	-100 ≤ x_i ≤ 100	(0,..,0)	0
Rastrign Function	$f(x) = 10D + \sum_{i=1}^{D} [x_i^2 - 10\cos(\pi x_i)]$	-5.12 ≤ x_i ≤ 5.12	(0,..,0)	0
Rosenbrock Function	$f(x) = \sum_{i=1}^{D-1} [\,100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	-5 ≤ x_i ≤ 10	(1,..,1)	0
Salomon Function	$f(x) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2}$	-100 ≤ x_i ≤ 100	(0,..,0)	0
Schaffer F6 Function	$f(x) = \sum_{i=1}^{D-1} 0.5 + \frac{\sin^2\left(\sqrt{x_i^2 + x_{i+1}^2}\right) - 0.5}{[1 + 0.001(x_i^2 + x_{i+1}^2)]^2}$	-100 ≤ x_i ≤ 100	(0,..,0)	0
Schwefel 2.4 Function	$f(x) = \sum_{i=1}^{D} (x_i - 1)^2 + (x_i + {x_i}^2)^2$	0 ≤ x_i ≤ 10	(1,..,1)	0
Sphere Function	$f(x) = \sum_{i=1}^{D} x_i^2$	-5.12 ≤ x_i ≤ 5.12	(0,..,0)	0

2.2. Selection

The selection operator chooses two chromosomes from the population to be parents in the crossover stage [17], [18]. A chromosome can be selected as a parent in the selection process based on its fitness. Chromosomes with better fitness are more likely to be selected as parents [19], [20]. Equations (1) to (3) describe that each chromosome has a selection probability p_k and cumulative probability q_k, where f_k is the fitness of the k-th chromosome, pop_size is the population size, $k = 1,2,\dots,pop\_size$, and F is the total fitness.

$$F = \sum_{k=1}^{pop\_size} f_k \quad (1)$$

$$p_k = \frac{f_k}{F} \quad (2)$$

$$q_k = \sum_{j=1}^{k} p_j \quad (3)$$

If r_k satisfies the condition $q_{k-1} \leq r_k < q_k$, then the k-th individual is selected. Here, r_k is a uniformly distributed random number between [0,1] generated as many times as the population size.

2.3. Crossover

The selected parents are subsequently mated to produce offspring as replacements for their parents.

The SBX algorithm [19]:

- Generate random numbers uniformly distributed in the range [0,1].
- Perform crossover if the randomly generated number is smaller than the crossover probability, pc.
- The spreading factor β is a non-negative variable that determines the distance between the offspring generated and the parent. The rules derived from the value of β are as follows:

1. $\beta = 1$, the distance between offspring and parent is zero; in other words, the offspring generated is identical to the parent.

2. $\beta > 1$, the distance between offspring and parent is distant.

3. $\beta < 1$, the distance between offspring and parent is close.

The value of β can be computed using (4).

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{\eta+1}} & ,u \leq 0.5 \\ \left[\frac{1}{2(1-u)}\right]^{\frac{1}{\eta+1}} & ,u > 0.5 \end{cases} \quad (4)$$

Where u is a random uniform [0,1], and η is a non-negative constant distribution index.

- The value of β generates two offspring using (5), where C_1 is offspring 1, C_2 is offspring 2, P_1 is parent 1, and P_2 is parent 2. The generated offspring chromosomes replace the parental chromosomes.

$$C_1 = \frac{1}{2}[(1+\beta)P_1 + (1-\beta)P_2]$$
$$C_2 = \frac{1}{2}[(1-\beta)P_1 + (1+\beta)P_2] \quad (5)$$

In Self-Adaptive SBX, the value of η is dynamically adjusted by comparing offspring quality to their parent [11], as shown in (6).

$$\begin{aligned} &if\ mean\big(fitness(C_1), fitness(C_2)\big)\ worse\ than \\ &\quad mean\big(fitness(P_1), fitness(P_2)\big) \\ &\qquad \eta = \alpha(1+\eta) - 1 \\ &else \\ &\qquad \eta = \frac{1+\eta}{\alpha} - 1 \end{aligned} \quad (6)$$

Where α is a factor satisfying $\alpha > 1$, the value of η is bounded within the range [0, 50].

2.4. Mutation

The mutation alters the value of a single gene within a chromosome in a population. Mutation facilitates the population to achieve diversity more rapidly, thereby preventing premature convergence. In the case of random mutation, gene values are altered by introducing a random number. The random mutation algorithm:

- Generate random uniform numbers in the range [0,1].
- If the randomly generated number is smaller than the mutation probability pm, then randomly select one gene within the chromosome for mutation.
- Perform mutation using (7).

$$g' = g + r(g_{max} - g_{min}) \quad (7)$$

Where g_{max} is the maximum value of the gene in the population, g_{min} is the minimum value of the gene in the population, r is a random number uniformly distributed within the range [-0.1, 0.1] [21], g represents an allele of the selected gene, and g' is a novel allele for the selected gene. The illustration of mutations can be observed in Figure 1.

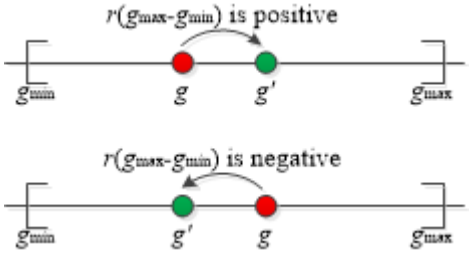


Fig. 1. Illustration of mutations

2.5. Elitism

The evaluation process can result in losing the best chromosome within a population. Elitism is a process where the best chromosome is retained in the population. Classical elitism duplicates the best chromosomes from the previous generation's population to replace the worst chromosomes in the new population [22]. Algorithm 1 illustrates the implementation of Self-Adaptive SBX-Elitism in a genetic algorithm.

Algorithm 1. Genetic Algorithm with Self-Adaptive SBX-Elitism

```
max_generation = 1000
convergence_threshold = 0.9
Generate population
Evaluation
Find best_fitness {chromosome with the best fitness in one generation}
Calculate population_convergence
global_fitness = best_fitness {chromosome with the best fitness from all generations}
While (population_convergence <= convergence_threshold && generation <= max_generation)
    For i = 1 to population
        Selection {selecting two chromosomes}
        Crossover
    End for
    For i = 1 to population
        Mutation
    End for
    Elitism {replace worst_fitness with best_fitness}
    Evaluation
    Find best_fitness
    Calculate population_convergence
    If global_fitness worse than best_fitness
        global_fitness = best_fitness
    End if
    generation++
End while
solution = global_fitness
```

3. Result and Discussion

The genetic algorithm with Self-Adaptive SBX and Self-Adaptive SBX-Elitism was implemented in C# on a computer with the configuration of AMD Ryzen 3 3250U, CPU @2.6GHz - Up to 3.5GHz, 8 GB RAM, and a 64-bit operating system. The testing was conducted on ten benchmark functions, as presented in Table 1. The predetermined parameters were η initial = 2 and α = 2 [11], pc = 0.9, pm = 0.2, and $trial$ = 100. The testing results included the average best, mean, and execution time. Tables 2 and 3 display the testing results for the benchmark functions across ten populations. Additionally, testing was performed on populations of twenty and thirty for the

benchmark functions, and the results are presented in Table 4 and Figures 2 to 4.

The results of testing on ten populations, as shown in Tables 2 and 3, indicate that GA with Self-Adaptive SBX-Elitism successfully reduced the average relative error by 99.99% with an average computation time that is 19.40% faster compared to GA with Self-Adaptive SBX. It proves that elitism in GA with Self-Adaptive SBX-Elitism can provide solutions that approach the global optimum. Figure 2 demonstrates that GA with Self-Adaptive SBX-Elitism, when tested on functions 5, 6, and 9, encountered difficulty finding solutions within populations of sizes 20 and 30.

Table 2. Testing of minimization benchmark functions across ten populations

Function	GA with Self-Adaptive SBX				GA with Self-Adaptive SBX-Elitism			
	Worst	**Best**	**Mean**	**Time**	**Worst**	**Best**	**Mean**	**Time**
***D* = 10**								
F1	7.12	0.00	3.11	49.64	0.77	0.00	**0.37**	**44.39**
F2	49.96	0.00	8.74	47.94	0.13	0.00	**0.00**	**33.93**
F3	-0.86	-1.00	-0.99	43.46	-1.00	-1.00	**-1.00**	**35.59**
F4	11.98	0.00	2.22	50.32	0.42	0.00	**0.05**	**36.80**
F5	171.23	0.00	72.68	48.44	11.02	0.00	**0.84**	**35.27**
F6	3359956.50	9320.22	132764.35	42.57	8.85	7.78	**8.55**	**36.69**
F7	13.38	0.00	4.60	46.88	0.60	0.00	**0.29**	**35.53**
F8	4.49	0.00	3.34	58.57	2.63	0.00	**0.99**	**44.22**
F9	10341.27	7.50	4887.45	71.05	0.03	0.00	**0.00**	**59.23**
F10	141.06	0.00	11.37	43.16	0.00	0.00	**0.00**	**20.51**
***D* = 20**								
F1	6.68	0.00	2.88	62.92	3.13	0.00	**0.79**	**56.72**
F2	75.98	0.00	15.50	63.01	0.54	0.00	**0.09**	**54.11**
F3	-0.90	-1.00	-0.99	**50.44**	-1.00	-1.00	**-1.00**	52.02
F4	20.83	0.00	3.29	66.32	0.44	0.00	**0.02**	**44.38**
F5	577.57	0.00	148.60	60.36	38.87	0.00	**12.92**	**57.29**
F6	3039372.30	25700.74	249437.04	52.72	18.99	17.95	**18.67**	**51.80**
F7	16.61	0.00	6.89	54.22	1.30	0.00	**0.94**	**51.03**
F8	9.48	0.00	7.16	71.93	6.47	0.00	**3.64**	**64.31**
F9	30712.66	5551.60	10831.78	95.56	1.14	0.00	**0.17**	**92.75**
F10	524.29	0.00	28.73	53.31	0.00	0.00	**0.00**	**32.18**
***D* = 30**								
F1	6.75	0.00	3.00	78.83	2.98	0.00	**1.33**	**70.54**
F2	85.29	0.00	21.46	90.21	1.70	0.00	**0.36**	**67.77**

F3	-0.88	-1.00	-0.98	73.26	-1.00	-1.00	**-1.00**	**59.34**
F4	23.77	0.00	4.26	92.35	0.18	0.00	**0.00**	**47.69**
F5	679.44	0.00	210.32	75.36	83.98	0.00	**29.11**	**67.72**
F6	11695568.96	39505.56	703246.66	63.21	28.98	28.30	**28.73**	**61.56**
F7	27.48	0.00	7.94	67.44	2.80	0.00	**1.54**	**58.89**
F8	14.42	0.00	9.66	105.09	11.28	0.00	**7.12**	**87.54**
F9	35339.22	10013.21	14999.14	167.33	10.90	0.04	**2.18**	**127.50**
F10	688.87	0.00	40.24	75.49	0.00	0.00	**0.00**	**42.08**

Table 3. Summary of benchmark function testing across ten populations

Dimension	**Mean of global optimum for ten benchmark functions**	**GA with Self-Adaptive SBX**			**GA with Self-Adaptive SBX-Elitism**		
		Average		**Relative error**	**Average**		**Relative error**
		Mean	**Time**		**Mean**	**Time**	
10	-0.10	13775.69	50.20	137757.90	**1.01**	**38.22**	**11.10**
20	-0.10	26048.09	63.08	260481.90	**3.62**	**55.66**	**37.20**
30	-0.10	71854.17	88.86	718542.70	**6.94**	**69.06**	**70.40**
Average	-0.10	37225.98	67.38	372260.83	**3.86**	**54.31**	**39.57**

Table 4. Testing of minimization problem benchmark functions using GA with Self-Adaptive SBX-Elitism

Function	**Solution with population = 20**				**Solution with population = 30**			
	Worst	**Best**	**Mean**	**Time**	**Worst**	**Best**	**Mean**	**Time**
***D* = 10**								
F1	0.72	0.00	0.39	118.71	0.87	0.00	0.38	187.87
F2	0.04	0.00	0.00	100.24	0.15	0.00	0.03	186.59
F3	-1.00	-1.00	-1.00	95.09	-0.98	-1.00	-1.00	173.99
F4	0.25	0.00	0.03	103.48	0.30	0.00	0.01	188.88
F5	9.35	0.00	1.74	105.43	9.95	0.00	2.71	188.34
F6	9.00	8.41	8.95	98.45	9.26	8.80	9.00	173.83
F7	0.60	0.00	0.29	95.15	0.83	0.00	0.26	169.81
F8	2.03	0.00	0.05	106.64	0.00	0.00	0.00	187.33
F9	0.68	0.00	0.06	132.15	12.33	0.01	7.71	215.77
F10	0.00	0.00	0.00	95.20	0.00	0.00	0.00	169.18
***D* = 20**								
F1	1.57	0.00	0.66	134.13	1.26	0.00	0.54	232.72
F2	0.38	0.00	0.08	130.25	0.46	0.00	0.13	226.62
F3	-1.00	-1.00	-1.00	121.58	-1.00	-1.00	-1.00	208.38
F4	0.95	0.00	0.02	137.83	0.00	0.00	0.00	239.57

F5	39.44	0.00	10.84	134.52	29.21	0.00	7.44	234.66
F6	19.00	18.77	18.98	125.09	19.28	18.95	19.00	208.65
F7	1.50	0.00	0.66	120.41	1.90	0.00	0.58	202.26
F8	6.45	0.00	0.39	147.61	7.47	0.00	0.14	239.59
F9	3.34	0.23	1.13	192.47	21.63	5.22	18.01	291.67
F10	0.00	0.00	0.00	119.02	0.00	0.00	0.00	206.18
***D* = 30**								
F1	1.89	0.00	0.87	169.26	1.50	0.00	0.64	281.15
F2	2.50	0.00	0.29	169.23	1.20	0.00	0.32	276.24
F3	-1.00	-1.00	-1.00	144.60	-1.00	-1.00	-1.00	246.32
F4	1.10	0.00	0.01	174.34	0.00	0.00	0.00	290.83
F5	76.88	0.00	22.78	167.19	90.70	0.00	16.55	285.24
F6	29.00	28.75	28.99	147.98	29.00	28.99	29.00	250.14
F7	2.50	0.00	1.20	146.42	2.81	0.00	0.85	242.53
F8	5.34	0.00	0.18	181.99	0.00	0.00	0.00	298.09
F9	10.91	1.41	4.60	239.50	36.19	17.47	28.39	366.23
F10	0.00	0.00	0.00	138.72	0.00	0.00	0.00	243.17

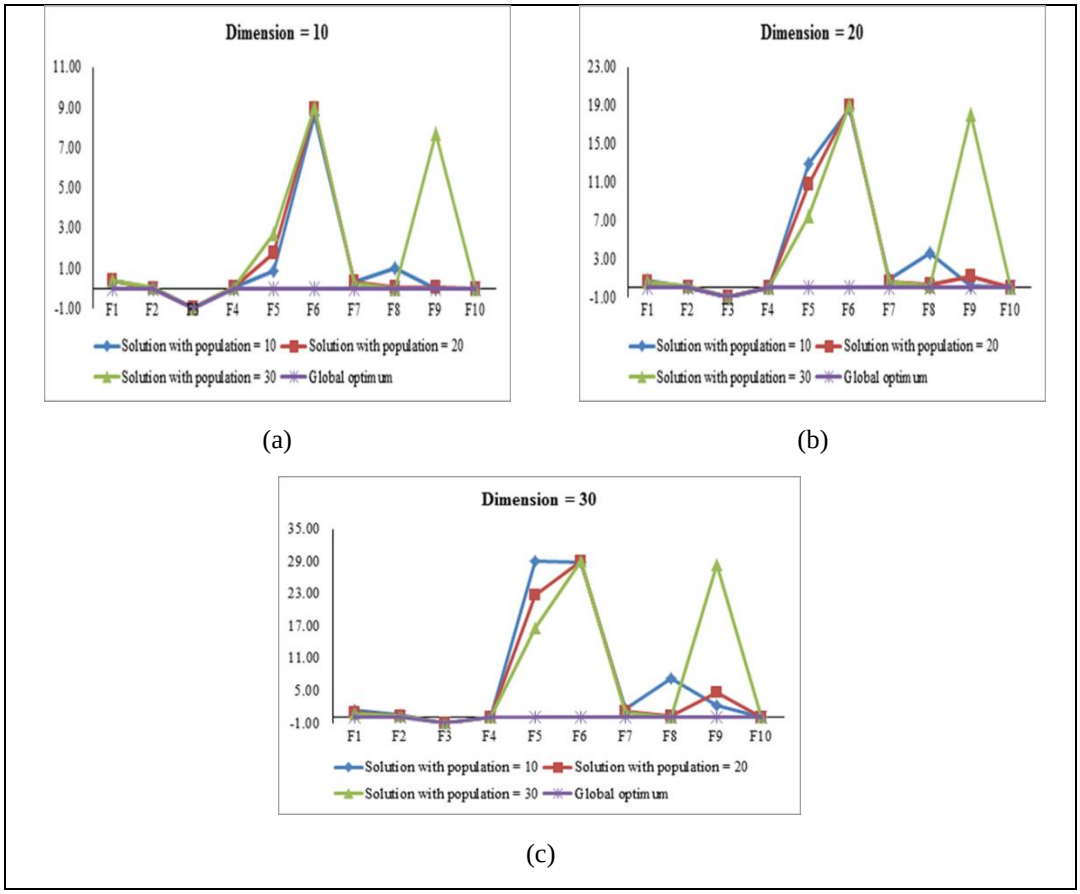
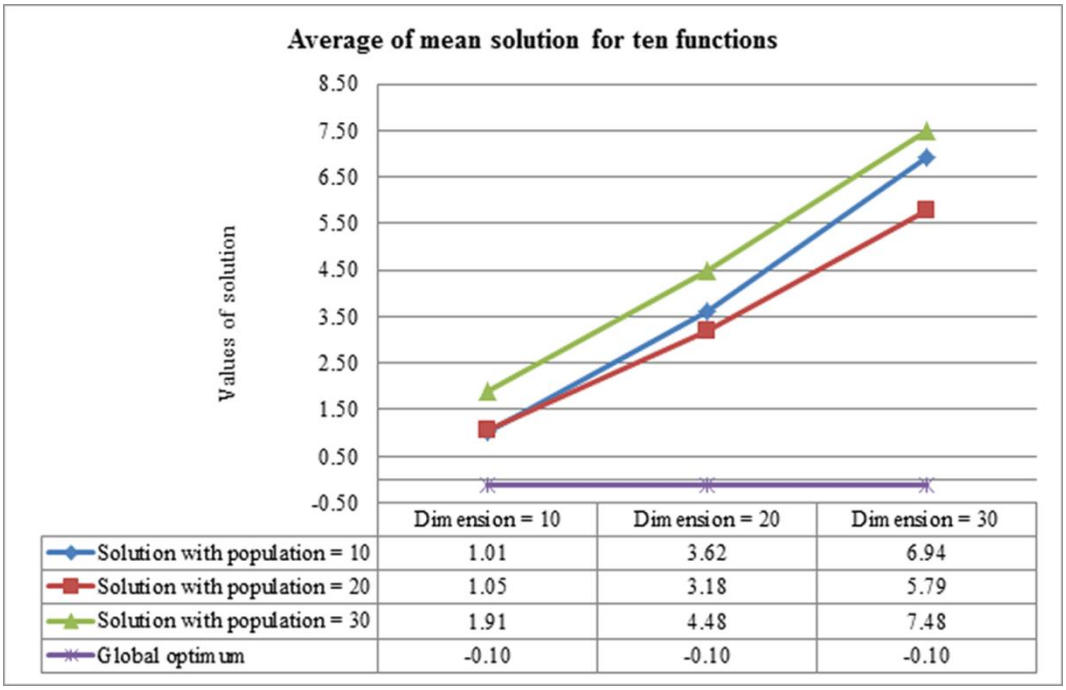


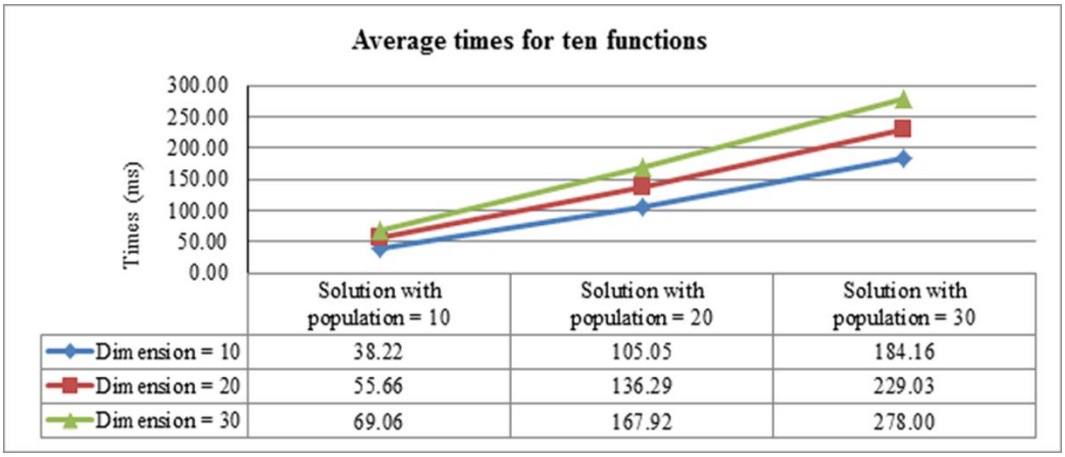
Fig. 2. Mean of minimization problem benchmark functions using GA with Self-Adaptive SBX-Elitism: (a) Dimension 10; (b) Dimension 20; (c) Dimension 30

However, in a population of size 10, GA with Self-Adaptive SBX-Elitism faced challenges in finding solutions for functions 5, 6, and 8. Testing was conducted on populations of sizes 20 and 30 for dimensions 10, 20, and 30 to obtain the optimal population size. Figures 3 and 4 show that GA with Self-Adaptive SBX-Elitism performed well with a population size of 20 across all tested dimensions, while the computation time ranked second.



	Dimension = 10	Dimension = 20	Dimension = 30
Solution with population = 10	1.01	3.62	6.94
Solution with population = 20	1.05	3.18	5.79
Solution with population = 30	1.91	4.48	7.48
Global optimum	-0.10	-0.10	-0.10

Fig. 3. Average of mean minimization problem benchmark functions using GA with Self-Adaptive SBX-Elitism in dimensions 10, 20, and 30



	Solution with population = 10	Solution with population = 20	Solution with population = 30
Dimension = 10	38.22	105.05	184.16
Dimension = 20	55.66	136.29	229.03
Dimension = 30	69.06	167.92	278.00

Fig. 4. Average computation time of ten minimization problem benchmark functions using GA with Self-Adaptive SBX-Elitism in dimensions 10, 20, and 30

4. Conclusion

This study tested the genetic algorithm with Self-Adaptive SBX and Self-Adaptive SBX-Elitism on ten benchmark functions. The test parameters were set as η initial = 2, α = 2, pc = 0.9, pm = 0.2, and $trial$ = 100. The testing results across ten populations in dimensions 10, 20, and 30 demonstrated that GA with Self-Adaptive SBX-Elitism was able to reduce the average relative error by 99.99% with an average computation time that was 19.40% faster compared to using GA with Self-Adaptive SBX. GA with Self-Adaptive SBX-Elitism performed well with a population size of 20 across all tested dimensions and ranked second in computation time. GA with Self-Adaptive SBX-Elitism ensures convergence toward the global optimum. For further research, the performance of the GA algorithm with Self-Adaptive SBX-Elitism on functions 5, 6, and 9 can be improved through experiments involving the utilization of different selection operators and elitism strategies.

Acknowledgements

The authors would like to thank the Department of

Computer Science, Faculty of Science and Engineering, Universitas Nusa Cendana for funding this publication.

Author contributions

The contributions of each author are described as follows: "conceptualization and methodology, Fanggidae, Prasetyo, and Polly; implementation of methodology, Prasetyo; data validation, Fanggidae and Prasetyo; formal analysis, Fanggidae, Polly, and Boru; investigation, Fanggidae, Prasetyo, Polly, and Boru; writing—original draft preparation, Fanggidae; writing—review and editing, Fanggidae, Polly and Boru; supervision, Fanggidae, Polly and Boru".

Conflicts of interest

The authors declare there are no conflict interest in this work.

References

[1] H. Wu, Z. Ni, and L. Ni, "Solving Roots of Complex Functional Equation Based on Improved Ant Colony Algorithm," in *2010 International Conference on E-Product E-Service and E-Entertainment*, IEEE, Nov. 2010, pp. 1–4. doi: 10.1109/ICEEE.2010.5661471.

[2] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci. (Ny).*, vol. 237, pp. 82–117, Jul. 2013, doi: 10.1016/j.ins.2013.02.041.

[3] Z. Beheshti and S. M. H. Shamsuddin, "A review of population-based meta-heuristic algorithm," *Int. J. Adv. Soft Comput. its Appl.*, vol. 5, no. 1, pp. 1–35, 2013.

[4] T. T. Tanyimboh, "Redundant binary codes in genetic algorithms: multi-objective design optimization of water distribution networks," *Water Supply*, vol. 21, no. 1, pp. 444–457, Feb. 2021, doi: 10.2166/ws.2020.329.

[5] M. V. Pathan, S. Patsias, and V. L. Tagarielli, "A real-coded genetic algorithm for optimizing the damping response of composite laminates," *Comput. Struct.*, vol. 198, pp. 51–60, Mar. 2018, doi: 10.1016/j.compstruc.2018.01.005.

[6] K. Deb and R. Bhushan Agrawal, "Simulated Binary Crossover for Continuous Search Space," *Complex Syst.*, vol. 9, pp. 115–148, 1995.

[7] K. Deb and A. Kumar, "Real-coded Genetic Algorithms with Simulated Binary Crossover : Studies on Multimodal and Multiobjective Problems," *Complex Syst.*, vol. 9, pp. 431–454, 1995.

[8] J. Chacon and C. Segura, "Analysis and Enhancement of Simulated Binary Crossover," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, Jul. 2018, pp. 1–8. doi: 10.1109/CEC.2018.8477746.

[9] S. Gunasekaran and M. W. Iruthayarajan, "Contour optimization of suspension insulators using real coded genetic algorithm with simulated binary crossover," in *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, IEEE, Feb. 2013, pp. 360–364. doi: 10.1109/ICPRIME.2013.6496501.

[10] L. Pan, W. Xu, L. Li, C. He, and R. Cheng, "Adaptive simulated binary crossover for rotated multi-objective optimization," *Swarm Evol. Comput.*, vol. 60, p. 100759, Feb. 2021, doi: 10.1016/j.swevo.2020.100759.

[11] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, New York, NY, USA: ACM, Jul. 2007, pp. 1187–1194. doi: 10.1145/1276958.1277190.

[12] S. S. Pabboju and T. Adilakshmi, "An Improved Approach for Scheduling in Cloud Using GA and PSO," *J. Theor. Appl. Inf. Technol.*, vol. 100, no. 18, pp. 5298–5307, 2022.

[13] A. Fanggidae and E. S. Y. Pandie, "Elitisme algoritma genetika pada fungsi nonlinear dua peubah," *J. Komput. dan Inform.*, vol. 8, no. 2, pp. 145–148, Oct. 2020, doi: 10.35508/jicon.v8i2.2894.

[14] V. K. Singh and V. Sharma, "Elitist Genetic Algorithm Based Energy Balanced Routing Strategy to Prolong Lifetime of Wireless Sensor Networks," *Chinese J. Eng.*, vol. 2014, pp. 1–6, Mar. 2014, doi: 10.1155/2014/437625.

[15] M. Jamil and X. S. Yang, "A literature survey of benchmark functions for global optimisation problems," *Int. J. Math. Model. Numer. Optim.*, vol. 4, no. 2, p. 150, 2013, doi: 10.1504/IJMMNO.2013.055204.

[16] S. Surjanovic and D. Bingham, "Virtual Library of Simulation Experiments: Test Functions and Datasets." Accessed: Mar. 22, 2023. [Online]. Available: http://www.sfu.ca/~ssurjano

[17] B. Farah, M. Awad, and A. Rutrot, "Prediction for Non-Revenue and Demand of Urban Water Using Hybrid Models of Neural Networks and Genetic Algorithms," *J. Theor. Appl. Inf. Technol.*, vol. 100, no. 21, pp. 6537–6551, 2022.

[18] Y. P. Makimaa and R. Sundarmani, "Genetic Algorithm Based Energy Efficient Cluster Head Selection and Cluster Formation and Establishment for Hierarchical Wireless Sensor Networks," *Int. J. Intell. Syst. Appl. Eng.*, vol. 10, no. 4, pp. 173–178, 2022.

[19] E. Wirsansky, *Hands-on genetic algorithms with Python : applying genetic algorithms to solve real-world deep learning and artificial intelligence problems*. Birmingham: Packt Publishing Ltd, 2020.

[20] S. Kadam and T. Srinivasarao, "ElitGA: Elitism Based Genetic Algorithm for Evaluation of Mutation Testing on Heterogeneous Dataset," *Int. J. Intell. Syst. Appl. Eng.*, vol. 11, no. 4s, pp. 509–516, 2023.

[21] D. David, T. Widayanti, and M. Q. Khairuzzahman, "Performance Comparison of Cat Swarm Optimization and Genetic Algorithm on Optimizing Functions," in *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, Denpasar, Indonesia: IEEE, Aug. 2019, pp. 35–39. doi: 10.1109/ICORIS.2019.8874901.

[22] Z. Avdagic, A. Smajevic, S. Omanovic, and I. Besic, "Path route layout design optimization using genetic algorithm: based on control mechanisms for on-line crossover intersection positions and bit targeted mutation," *J. Ambient Intell. Humaniz. Comput.*, vol. 13, no. 2, pp. 835–847, Feb. 2022, doi: 10.1007/s12652-021-02937-z.