Generalized Hybrid Evolutionary Algorithm Framework with a Mutation Operator Requiring no Adaptation

Yong Wee Foo[1,2], Cindy Goh[2], Lipton Chan[2], Lin Li[3,4] and Yun Li[2,4*]

[1] School of Engineering, Nanyang Polytechnic, Singapore 569830, Singapore
Foo_Yong_Wee@nyp.edu.sg;
[2] School of Engineering, University of Glasgow, Glasgow G12 8LT, U.K.
Cindy.Goh@glasgow.ac.uk; Dr.Lipton.Chan@gmail.com
[3] School of Computer Science and Engineering,
South China University of Technology, Guangzhou, Guangdong 510006, China
LinLi@ieee.org;
[4*] School of Computer Science and Network Security,
Dongguan University of Technology, Songshanhu, Guangdong 523808, China
* Corresponding author: Yun.Li@ieee.org

Abstract. This paper presents a generalized hybrid evolutionary optimization structure that not only combines both nondeterministic and deterministic algorithms on their individual merits and distinct advantages, but also offers behaviors of the three originating classes of evolutionary algorithms (EAs). In addition, a robust mutation operator is developed in place of the necessity of mutation adaptation, based on the mutation properties of binary-coded individuals in a genetic algorithm. The behaviour of this mutation operator is examined in full and its performance is compared with adaptive mutations. The results show that the new mutation operator outperforms adaptive mutation operators while reducing complications of extra adaptive parameters in an EA representation.

Keywords: Optimization algorithms, Evolutionary algorithms, Evolutionary computation, Derivative-free optimization, heuristic search

1 Introduction

Conventional optimisation algorithms are often deterministic and are restricted to numerical optimisation with one objective and one optimum [1-2]. Lifting much of the restriction with nondeterministic search, evolutionary algorithms (EAs) are able to deal with extra-numerical, multi-objective and multimodal problems [3]. However, an inevitable cost an EA pays for this is that it converges far more slowly around a local optimum. Therefore, combining the global aspect of EAs with the local strength of conventional algorithms has been an active research topic in evolutionary computation, resulting in hybrid algorithms with both merits and deficiencies [4-5].

Formatted: Indent: First line: 0 cm

Historically, the development of EAs originated from three initial paradigms: the genetic algorithm (GA), the evolution strategy (ES), and the evolutionary programming (EP) [6]. These algorithms have much in common but also have differences. Merging these three EAs, this paper develops a generalized hybrid EA (GHEA) with customizable operations to commission EAs with differing behaviors for different applications. Without the need for adaptation, a simulated binary mutation (SBM) operator is proposed for the GHEA framework. This framework is to improve the flexibility of a maturing population via a number of different selection techniques, so as to enhance both local and global search.

The remainder of this paper is organized as follows. Section 2 analyzes the merits and deficiencies of hybrid EAs reported so far. Based on these analyses, Section 3 develops the details of the GHEA. The SBM operator is detailed in Section 4. Section 5 presents test results and discussions. Section 6 concludes the paper with recommendations for future work.

2 Merits and Deficiencies of Existing Hybrid EAs

A number of hybrid EA techniques were summarized in the *Handbook of Evolutionary Computation* [7]. Subsequently, many effective hybrid EA algorithms have been developed for specific applications. One example is a hybrid EA with fitness approximation for solving the two-stage capacitated facility location problem, which uses genetic operations with a restarting strategy and extreme machine learning to approximate the fitness of most of the individuals [8]. Another example incorporates a tabu search procedure into the framework of an evolutionary algorithm in solving a job shop scheduling problem [9]. For tackling continuous multiobjective optimization problems, a recombination operator is proposed to at the gene level to combine the advantages of simulated binary crossover with local search and differential evolution with global search [10]. However, these hybrid EAs rely on GA operators, and do not accommodate ES or EP for customizability or flexibility.

A hybridized EA framework should take the advantage of the global search capability of the EA and the search speed as well as accuracy of local search algorithms such as hill-climbing. Further, Baldwin effect and Lamarckian evolution also help achieve this goal [11-13]. With the Baldwin effect, individuals will learn locally but the learning does not affect their genetic code. With the Lamarckian principle of “use and disuse”, however, each individual will pass much of the learning results in genetic code to offspring. Hence, a hybrid EA framework should readily support both of these options.

Alternative, conventional optimization algorithms can also be incorporated into a hybrid EA, so that whenever appropriate the EA can evaluate an individual for local optimization with resulting solutions stored back into the individual. Conventional algorithms are deterministic in nature and are dependent of their initial starting position. This strengthens exploitation of the exploration results brought about by the EA population. For example, gradient guided algorithms such as the quasi-Newton methods and the conjugate gradient methods. Quasi-Newton methods store an estimate of the Hessian internally and iterate it using Newton’s method whereas conjugate gradient methods use line optimisations in directions that are conjugate to previously tried directions. These methods are very fast at locating optima, but the derivative function is required.

3 Generalized Hybrid EA Framework

Based on the analyses of the merits and deficiencies of hybrid EAs in the literature, a generalized hybrid EA framework is proposed in this paper, as shown in Fig. 1. The framework also includes other basic elements that are part of applying an EA, such as objective, constraints and other housekeeping tasks. It is to allow customization of EAs with different behaviours, to improve flexibility for the formation of adult population via a number of different selection techniques, and to enhance EA performance through integration of both local and global optimization algorithms. The framework also implements a new non-adaptive mutation operator which does not require any evolving scaling parameters.

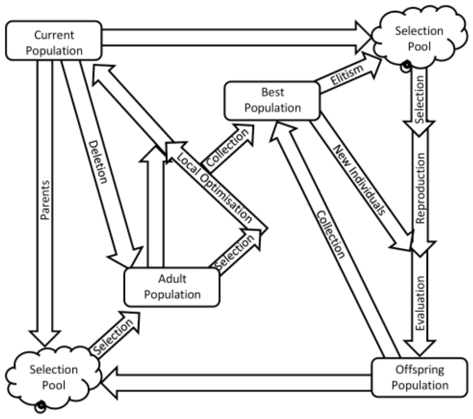

Fig. 1. Generalization of Hybrid EAs

In the GHEA framework, once an offspring population is created, an adult population follows from it, which will eventually replace the current population. A number of different techniques exist to create this adult population. In (μ,λ) ES, for example, the best μ offspring individuals are used to create the adult population. In $(\mu+\lambda)$ ES, however, the parents and the offspring are aggregated and the best μ individuals are used. From GAs, the notion of generation gap was created to describe the formation of the adult population. The generation gap is the percentage of individuals in the current population to be deleted and replace by offspring to form the adult population [7]. The GHEA framework accommodates for all these variations in adult population creation.

A replacement selection pool of individuals is created from the offspring population and the current population can optionally be included. A number of individuals are then selected from this pool and the same number of individuals is selected for deletion from the current population. These replacement individuals and remainder individuals then form the adult population.

4 Simulated Binary Mutation for Improved Efficiency

4.1 Adaptive Mutations

A commonly used real-value mutation scheme in an EA involves a Gaussian distribution, as given by:

$$x_i[t+1] = x_i[t] + N_i(0, \eta_i[t]) \tag{1}$$

where *N(μ,σ)* is a normal distribution of mean *μ* and standard deviation *σ*. Another mutation operator which appears to be better than the Gaussian mutation, especially for multimodal problems, is the Cauchy mutation [14-15], as given by:

$$x_i[t+1] = x_i[t] + \eta_i[t]\delta_i[t] \tag{2}$$

where δ_i is a sample from the Cauchy density function:

$$f(x) = \frac{1}{\pi(x^2+1)} \tag{3}$$

These two mutation operators can be made to adapt their shapes according to the landscape of the objective function, in a way similar to the Estimation of Distribution Algorithm (EDA). Such adaptive mutation operators have generally been found to outperform their static counterparts.

For the adaptive versions, each real-value parameter of the objective function is replaced by a 2-tuple of real values, $x_i \to (x_i, \eta_i)$. The new parameter, η_i, is the scaling parameter for the adaptive mutation and is itself mutated using the following update:

$$\eta_i[t+1] = \eta_i[t]\exp(N(0,\tau') + N_i(0,\tau)) \tag{4}$$

where the recommended values of τ and τ' are $(4n)^{-1/4}$ and $(2n)^{-1/2}$, respectively [16].

4.2 Simulated Binary Mutation

To reduce overhead of the scaling parameters especially for real-time applications of an EA, a new static mutation operator is developed in this section. Binary coding has been used in GAs, where the accuracy of the resulting solutions has been limited by the chromosome length. The mutated bit in an individual chromosome can be the LSB, MSB, or any other bit. Thus the mutation has equal probability of making large changes as of making small changes.

In binary coding, when a real value, $x \in [a, b]$, is represented by a string of l bits, $\boldsymbol{s} \in \{0,1\}^l$, the real value is given by:

$$x = a + r\sum_{i=1}^{l} 2^{i-1}s_i \tag{5}$$

$$r = \frac{b-a}{2^l-1} \tag{6}$$

If the *i*th bit in s is flipped, i.e., $s_i[t+1] = 1 - s_i[t]$, then:

$$x[t+1] = x[t] \pm |\Delta_i| \tag{7}$$

$$|\Delta_i| = 2^{\wedge}(i-1)\ r \tag{8}$$

In a real value representation, however, there may not be any bounds on the value and thus no resolution is set. Therefore, the smallest or largest resolution for a real value parameter is dependent on the order of magnitude of the value itself without the original value losing significance.

$$r[t] = 2^{-m}|x[t]| \tag{9}$$

m is the bits of significance. The perturbation amount due to the mutation can then be defined for real value parameters as:

$$|\Delta[t]| = 2^{i}|x[t]| \tag{10}$$

where $i \in\ [-m, m]$. This can be converted to base-10 so that:

$$|\Delta[t]| = 10^{j}|x[t]| \tag{11}$$

and $j \in\ [-k, k]$ where $k = m\frac{\ln 2}{\ln 10}$. Now *k* is the number of significant digits in base-10. The perturbation amount is now proportional to the value of the parameter and this will result in a perturbation of zero always when the value is zero. Thus, the zero case has to be treated specially and the mutation of the real value is defined as:

$$|\Delta[t]| = \begin{cases} 10^{kU(-1,1)}|x[t]| & \text{if } x[t] \neq 0 \\ 10^{kU(-1,1)}|x_{small}| & \text{if } x[t] = 0 \end{cases} \tag{12}$$

$$p[t] = U(-1,1) \tag{13}$$

$$x[t+1] = \begin{cases} x[t] + |\Delta[t]| & \text{if } p[t] \geq 0 \\ x[t] - |\Delta[t]| & \text{if } p[t] < 0 \end{cases} \tag{14}$$

where *U*(a,b) is a sample from a uniform distribution in the interval [a,b] and x_{small} is the smallest number greater than zero that can be represented.

The probability of mutation for each individual is set as $\frac{1}{n}$, where *n* is the number of real-value parameters in the individual. Thus, in probability there will on average be one real value parameter mutated in each individual. The probability distribution function of the new value of the mutated parameter is thus:

$$\Pr(x[t+1]) \begin{cases} 4k(x[t+1] - x[t] \ln 10)^{-1} & \text{if } x[t](1+10^{-k}) \leq x[t+1] \leq x[t](1+10^{k}) \\ 4k(x[t] - x[t+1] \ln 10)^{-1} & \text{if } x[t](1+10^{-k}) \leq x[t+1] \leq x[t](1+10^{k}) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

This p.d.f. is illustrated in Fig. 2. The range of this mutation is determined by the constant *k* which as mentioned previously is the number of significant digits that this mutation will perturb to. In most cases, after considering the floating-point implementation and accumulated round-off errors, the constant k can be determined.

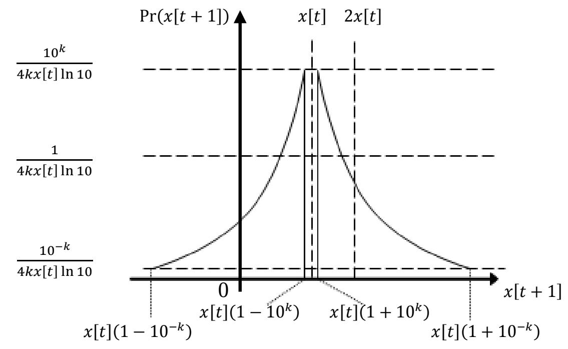

Fig. 2. Probability distribution of the mutated real value where there is on average one real parameter mutated in each individual

If, however, true arbitrary preci sion is desired, then a variant of this mutation can be used instead where (12) is replaced by the following

$$|\Delta[t]| = \begin{cases} 10^{N(0,k)}|x[t]| & \text{if } x[t] \neq 0 \\ 10^{N(0,k)}|x_{small}| & \text{if } x[t] = 0 \end{cases} \tag{16}$$

where the uniform distribution is replaced by the Gaussian distribution. The mutation operator defined by (12) will be referred to as the simulated binary mutation – uniform or SBM-U(*k*) for short, and the mutation operator defined by (16) will be referred to as SBM-G(*k*).

Now that the SBM operators have been defined, their performance is compared with those of the adaptive Gaussian and the adaptive Cauchy mutations in the next section.

5 Tests and Result Discussions

In order to evaluate their performance of SBM operators on different functions, it is further compared with those of those of the adaptive Gaussian and the adaptive Cauchy mutations. The adaptive mutations require a coding representation that stores both the real value parameters and their corresponding scaling parameters. The SBMs, however, does not require any evolving scaling parameters and the coding requires nothing other than the real value parameters themselves. For the SBMs, k was set to 5, and for the adaptive mutations the scaling parameters were bounded to the interval $[10^{-5}, 10^5]$ with the initial scaling set to 3.

Table 1 lists the objective functions used. Functions f_1 to f_7 are unimodal functions with high dimensionality. Functions f_8 to f_{11} are high dimensional multimodal functions and f_{12} to f_{16} are low dimensional multimodal functions. The EP algorithm with the population size, μ, being 100 are performed on each of the objective functions in Table 1 for 50 runs for all of the two adaptive mutations and the two SBMs.

Table 1. Benchmark objective function used to test the performance of mutation operators

Test Function	n	Range	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100,100]^n$	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	$[-10,10]^n$	0
$f_3(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$	30	$[-100,100]^n$	0
$f_4(x) = max_i\{\lvert x_i \rvert, 1 \leq i \leq n\}$	30	$[-100,100]^n$	0
$f_5(x) = \sum_{i=1}^{n-1}(100(x_{i+1} - x^2)^2 + (x_i - 1)^2)$	30	$[-30,30]^n$	0
$f_6(x) = \sum_{i=1}^{n}(\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100,100]^n$	0
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + U\,[0,1)$	30	$[-1.28,1.28]^n$	0
$f_8(x) = -\sum_{i=1}^{n} x_i sin\sqrt{\lvert x_i \rvert}$	30	$[-500,500]^n$	-1.2569.5
$f_9(x) = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10cos(2\pi x_i)\right)$	30	$[-5.12,5.12]^n$	0
$f_{10}(x) = 20 + e - 20exp\left(-\sqrt{\frac{1}{25n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right)$	30	$[-32,32]^n$	0
$f_{11}(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\frac{x_i}{\sqrt{i}}$	30	$[-600,600]^n$	0
$f_{12}(x) = \left(\frac{1}{500} + \sum_{j=1}^{25}\left(j + \sum_{i=1}^{2}(x_i - a_{ij})^6\right)^{-1}\right)^{-1}$	2	$[-65.536,65.536]^n$	~1
$f_{13}(x) = \sum_{i=1}^{11}\left(a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right)^2$	4	$[-5,5]^n$	3.075x10^{-4}
$f_{14}(x) = 4x^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	$[-5,5]^n$	-1.0316285
$f_{15}(x) = \left(x^2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)cosx_1 + 10$	2	$[-5,10]$ x $[0,15]$	0.398
$f_{16}(x) = \left(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right)$ x $\left(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right)^2$	2	$[-2,2]^n$	3

In the unimodal tests, $f_1 - f_7$,, there was very little difference in the performance of SBM-U and SBM-G. Both of them outperformed the adaptive Gaussian and the adaptive Cauchy mutations for most of these benchmark problems. However, the SBMs' convergence has been seen worse in benchmark functions f_3 and f_5. The averaged best results are plotted against the generation number in Fig. 3 and the average final results are tabulated in Table 2. Thus, to be fair, the SBMs results are also compared with the published results of [15] in Table 2. Again, the SBMs performed better in all but functions f_3 and f_5. On average and for almost all the unimodal benchmark problems, SBMs converges linearly with increasing generations.

The results of the tests on the four high-dimensional multimodal benchmark functions, $f_8 - f_{11}$, show that both of the SBMs performed significantly better than the two adaptive mutations. These results are shown in the plots of Fig. 4 and are tabulated in Table 3. The performance of SBM-U and SBM-G were virtually identical in this set of functions and showed linear convergence in f_9 and f_{10}. In all of these the worst performer was undoubtedly the adaptive Gaussian mutation which got trapped in poor local optima for all four test functions. The averaged result of the adaptive Cauchy mutation was, like the adaptive Gaussian, trapped by local optima in f_9. For this test function, both SBMs had found the global optimum in all the trials.

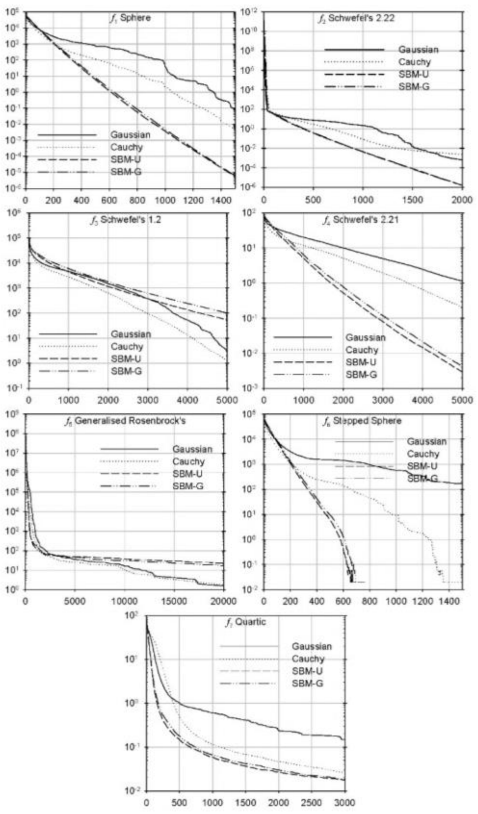

Fig.3. Average of 50 trials on best fitness on high dimensional unimodal benchmark problems

Table 2. Comparison of best results averaged over 50 trials for unimodal benchmark functions

Func-tions	Gen	Gaussian	Cauchy	SBM-U	SBM-G	Yao et al. [17]	
						Gaussian	Cauchy
f_1	1500	8.79×10^{-2}	4.55×10^{-3}	5.56×10^{-6}	6.22×10^{-6}	2.2×10^{-4}	$5.7\times10^{-4\cancel{3}}$
f_2	2000	6.51×10^{-4}	2.55×10^{-3}	1.61×10^{-6}	1.63×10^{-6}	2.6×10^{-3}	8.1×10^{-3}
f_3	5000	3.05	1.26	53.5	101.6	5.0×10^{-2}	1.6×10^{-2}
f_4	5000	1.14	0.205	2.95×10^{-3}	4.20×10^{-3}	2.0	0.3
f_5	20000	1.59	1.72	22.8	16.3	6.17	5.06
f_6	1400	169.4	0.02	0	0	577.8	0
f_7	3000	0.147	2.64×10^{-2}	1.77×10^{-2}	1.86×10^{-2}	1.8×10^{-2}	7.6×10^{-3}

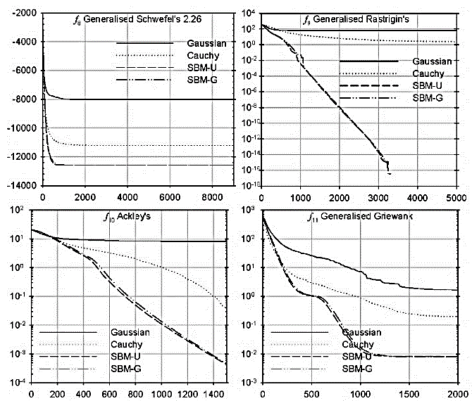

Fig. 4. Average of 50 trials on best fitness on high dimensional multimodal benchmark problems

Table 3. Comparison of best results averaged over 50 trials for high dimensional multimodal benchmark functions

Func-tions	Gen	Gaussian	Cauchy	SBM-U	SBM-G	Yao et al.[17]	
						Gaussian	Cauchy
f_8	9000	-8001.1	-11218.5	-12569.5	-12569.5	-7917.1	-12554.5
f_9	5000	66.9	2.41	0	0	89.0	4.6×10^{-2}
f_{10}	1500	8.17	3.94×10^{-2}	4.33×10^{-2}	4.45×10^{-4}	9.2	1.8×10^{-2}
f_{11}	2000	1.64	0.199	8.03×10^{-3}	7.97×10^{-3}	8.6×10^{-2}	1.6×10^{-2}

Fig. 5 shows the average convergence of the best solutions against generations for the low dimensional multimodal benchmark functions, $f_{12} - f_{16}$. The final averaged best results are summarised in Table 4. All four mutations found the optimal solutions for $f_{14} - f_{16}$. and had similar convergence curves. For test function f_{12}, both SBMs performed better than the two adaptive mutations and found the global optimum. However, the performances of the SBMs were worse than both adaptive mutations for test function f_{13}.

Overall the SBMs performed better than the adaptive mutations while having a linear convergence rate in many of the test functions. This is achieved without the added complexity of having scaling parameters for each objective parameter in every individual. [15] also introduced the improved fast evolutionary programming (IFEP) algorithm in which each parent individual produces two individuals by adaptive Gaussian and by adaptive Cauchy mutations and the better of the two is kept as the offspring. To allow for fair comparison the IFEP algorithm used half the population size and results were published for the IFEP for test functions f_1, f_2, f_{10}, and f_{11} [15].

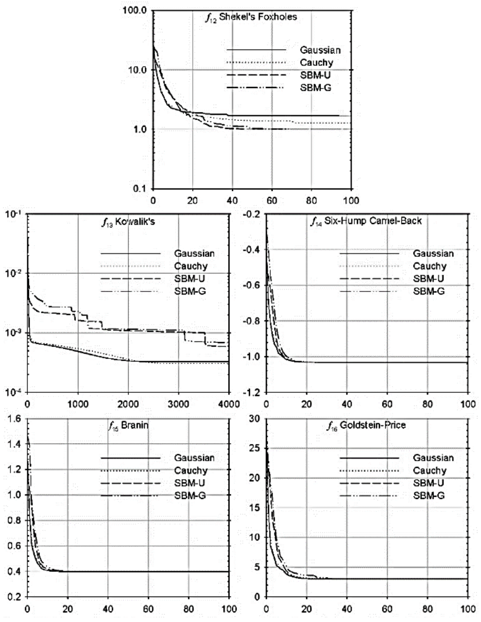

Fig.5. Average of 50 trials on best fitness on low dimensional multimodal benchmark problems

Table 4. Comparison of the best results averaged over 50 trials for the low dimensional multi-modal benchmark functions

Func-tions	Gen	Gaussian	Cauchy	SBM-U	SBM-G	Yao et al.[17]	
						Gaussian	Cauchy
f_{12}	100	1.68	1.27	0.998	0.998	1.66	1.22
f_{13}	4000	3.28×10^{-4}	3.07×10^{-4}	5.84×10^{-4}	6.84×10^{-4}	4.7×10^{-4}	5.0×10^{-4}
f_{14}	100	-1.03	-1.03	-1.03	-1.03	-1.03	-1.03
f_{15}	100	0.398	0.398	0.398	0.398	0.398	0.398
f_{16}	100	3	3	3	3	3	3.02

The results of both of the SBMs were also better than those of the IFEP algorithm. The similarity of the results of SBM-U and SBM-G suggests that the order k has only a small effect on the convergence. There are however three of the test functions that the SBMs performed worse than the adaptive mutations, namely f_3, f_5 and f_{13}. By looking into the landscapes of these test functions and the mechanism of the SBMs, the type of problems that the SBMs perform less well in and why it is so can be found. Contour plots of two dimensional versions of f_3 and f_5 are shown in Fig. 6 The minimum of f_3

is elliptical with its major and minor axis rotated from, and thus unaligned with, the parameter axes. The Rosenbrock function, f_5, is sometimes known as the banana function due to its minimum being located within a curved valley.

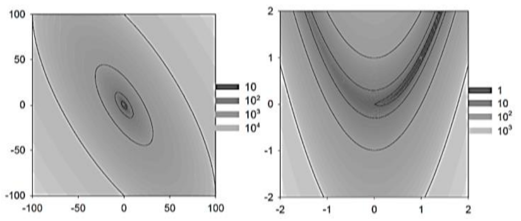

Fig. 6. Contour plots of f_3 and f_5 with $n = 2$, left and right respectively

As mentioned previously, the SBM is derived from binary mutation and hence inherited its mutation rate method. That is, only one of the objective parameters in each individual will undergo mutation, hence the SBM suffers from the same problem as performing line optimisations along axes directions only. In problems like f_3, where the ellipse's principal axis is at an angle to the parameter axis, the algorithm has to zigzag, constantly crossing the ellipse's principal axis, to reach the minimum. In such problems, it is necessary to simultaneously adjust many parameters to be able to converge efficiently and traverse the direction closer to the minimum. The adaptive mutations performed better here because all of the objective parameters undergo mutation and thus individuals can travel diagonally.

Likewise, for f_5, individuals mutated by the SBM must zigzag around the bent valley to reach the optimum which is very inefficient. These types of functions are probably best tackled with the adaptive mutations with correlated mutations, i.e., rotation angles. For the SBM to be more efficient in these types of functions, it may be necessary to abandon the one mutation-per-individual mutation rate and adopt an alternative approach to selecting which parameters to undergo mutation.

6 Conclusions

By factoring out the common denominators of the three originating branches of evolutionary algorithms (EA) and identifying their particularities, a generalised HEA structure has been created, which behaves like any of the three original EA paradigms. The customisability and flexibility of the library makes it a more comprehensive tool in the application and research of EAs.

The mutation of chromosomes in a binary-coded GA is very good in its ability to produce mutations of different magnitudes. The simulated binary mutation has been created to simulate this binary-coded mutation for real-value parameters. SBM is seen to perform very well comparing with adaptive mutations for real-value parameters showing good convergence on many benchmark functions, although it is identified that certain landscapes can cause SBM a difficulty. In these landscapes, the SBM would have to zigzag slowly along a valley. This behaviour is due to the typical one mutation per chromosome inherited from binary-coded mutation of GAs, but increasing the mutation rate would allow individuals to traverse at a higher speed.

References

1. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization. In: SIAM (2009).
2. Powell, M.J.D.: Direct search algorithms for optimisation calculations. In: Acta Numerica, vol. 7, pp. 287-336 (1998). doi:10.1017/S0962492900002841
3. Fogel, D.B.: Evolutionary Computation: Towards a New Philosophy of Machine Intelligence. IEEE Press (1995). doi:10.1002/0471749214
4. Grosan, C., Abraham, A.: Hybrid Evolutionary Algorithms: Methodologies, Architectures, and Reviews. Springer Berlin Heidelberg (2007). doi: 10.1007/978-3-540-73297-6_1
5. Lin, Q., Chen, J., Zhan, Z.H.: A hybrid evolutionary immune algorithm for multiobjective optimization problems. In: IEEE Transactions on Evolutionary Computation vol. 20(5), pp. 711-729 (2016). doi:10.1109/TEVC.2015.2512930
6. Chen, G., Low, C.P., Yang, Z.H.: Preserving and exploiting genetic diversity in evolutionary programming algorithms. In: IEEE Transactions on Evolutionary Computation, vol. 13, pp. 661-673 (2009). doi:10.1109/TEVC.2008.2011742
7. Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd and Oxford University Press (1997).
8. Guo, P., Cheng, W., Wang, Y.: Hybrid evolutionary algorithm with extreme machine learning fitness function evaluation for two-stage capacitated facility location problem. In: Expert Systems with Applications, vol. 71, pp. 57-68 (2016). doi:10.1016/j.eswa.2016.11.025
9. Cheng, T.C.E., Peng, B., Lü, Z.: A hybrid evolutionary algorithm to solve the job shop scheduling problem. In: Annals of Operations Research, vol. 242(2), pp. 223-237 (2016). doi:10.1007/s10479-013-1332-5
10. Zhu, Q., Lin, Q., Du, Z.: A novel adaptive hybrid crossover operator for multiobjective evolutionary algorithm. In: Information Sciences, vol. 345, pp. 177-198 (2016). doi:10.1016/j.ins.2016.01.046
11. Anderson, R.W.: The Baldwin effect. In: Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, C3.4:1-C3.4:7. Institute of Physics Publishing Ltd and Oxford University Press (1997).
12. Li, L., Zhang, C.Z., Li, Z.N., Li, Y.: Particle filter with Lamarckian inheritance for nonlinear filtering. In: 2016 IEEE Congress on Evolutionary Computation, pp.2852-2857, IEEE, Vancouver, Canada (2016). doi:10.1109/CEC.2016.7744149
13. Li, L., Li, Y.: Particle filter track-before-detect algorithm with Lamarckian inheritance for improved dim target tracking. In: 2017 IEEE Congress on Evolutionary Computation, pp. 1158-1164, IEEE, San Sebastián, Spain (2017). doi:10.1109/CEC.2017.7969437
14. Yao, X., Liu, Y.: Fast evolution strategies. In: Control and Cybernetics, vol. 26(3), pp. 467-496 (1997). doi:10.1007/BFb0014808
15. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. In: IEEE Transactions on Evolutionary Computation, vol. 3(2), pp. 82-102 (1999). doi:10.1109/4235.771163
16. Bäck, T.: Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press (1996).