Selection of Building Blocks for Adaptive Grammatical Evolution in PSO

Jyotheesh Gaddam[1*], Jan Carlo Barca[1†], Richard Dazeley[1†] and Maia Angelova[1†]

[1]Data to Intelligence (D2I) Research Centre, School of Information Technology, Deakin University, Burwood, Melbourne, 3125, VIC, Australia.

*Corresponding author(s). E-mail(s): jgaddam@deakin.edu.au;
Contributing authors: jan.barca@deakin.edu.au;
richard.dazeley@deakin.edu.au; maia.a@deakin.edu.au;
†These authors contributed equally to this work.

Abstract

In this study, we used grammatical evolution to develop a customised particle swarm optimiser by incorporating adaptive building blocks. This makes the algorithm self-adaptable to the problem instance. Our objective is to provide the means to automatically generate novel population-based meta-heuristics by scoring the building blocks. We propose a new self-adapting algorithm by adaptive selection and scoring of the building blocks to solve multiple problem instances by reducing computation time and iteration count. To achieve our objective, we ranked building blocks that were extracted from a broad set of existing particle swarm optimisers and scored these during the evolutionary process. These scores were provided as an input to the evolutionary process that enabled the replacement of blocks of evolved solutions in cases where they were unable to improve the overall fitness. Our numerical experiments demonstrated that the proposed algorithm with adaptive building blocks reduced the iteration count and computation time with respect to PSO.

Keywords: Particle Swarm Optimisation, Swarm Intelligence, Evolution Strategies

1 Introduction

Particle Swarm Optimisation (PSO) is one of the most popular nature-inspired meta-heuristic optimisation algorithms developed by [1]. Since its development, PSO is one of the widely used optimisation technique in solving continuous, discrete, constrained and unconstrained problems. Optimisation problems in real-world situations can be divided into a few categories. The first type is discrete or continuous. In discrete optimisation, a problem may consist of a finite number of points to be ordered in the most efficient way. An example is finding an optimal path between a number of points. On the other hand, a continuous optimisation problem deals with a set of real values. The second type occurs if the problem were single- or multi-objective. The third type of problem can be unconstrained or constrained. In addition, problems can be either static (don't change over time) or dynamic, which means they have at least one component that is likely to change over time.

Research on dynamic optimisation has become a challenging research topic. The objective function, constraints, or both change over time in dynamic optimisation. To solve dynamic problems, various approaches have been used, including Genetic Programming, Mathematical Optimisation, and Bayesian Optimisation. Researchers also developed hybrid approaches to further improve PSO by introducing combinations with Genetic Algorithms [2], Simulated Annealing [3], Ant Colony Optimisation [4], Cuckoo Search [5], Artificial Bee Colony [6], Artificial Immune Systems [7], Bat Algorithm [8], Firefly Algorithm [9] and Glow Worm Swarm Optimisation [10]. With the hybridisation of the PSO algorithms, they overcome the issues of exploration and exploitation (means a decision to take when to explore and when to exploit), trapping local minima, convergence rate, uni-modal and multi-modal optimisation. But still these algorithms lack of continuous self-adaption capability to improve and solve different problem instances.

Among these, Evolutionary and Swarm algorithms are two important algorithms belonging to the nature-inspired meta-heuristics known as an evolutionary computation [11]. Popular hybridisation approaches in PSO are with Genetic Algorithms by sharing the common framework. These meta-heuristics shares the following two characteristics: population-based presentation for the candidate solutions, and an iterative procedure with a stochastic exploration [11]. Grammatical Evolution (GE) is an evolutionary search technique similar to genetic programming [12], inspired by the process of biological evolution. Several research works on PSO and GE use Building Blocks (BBs) to solve different problem instances.

The algorithm usually involves BBs in designing a search process suitable for a situation depending on the constraints and decision variables [13]. The BBs is a set of constraints or rules. In what follows, the BBs will be identified and provided by us to the system to build an algorithm. In many algorithms, we observed that the random selection of BBs is made every time in initialising the search process. This random initialisation may consume more time in the

search process. Moreover, we cannot guarantee that the provided blocks will be efficient in finding the best solution [14].

In dynamic environments, changes occur with data, inputs and objective function (e.g., stock market). The search process must adapt to the changes and come up with the best possible solution. Heuristic selection, using previous best-performed blocks to solve new problems, is not very useful. Moreover, selecting the existing low-level heuristics reduces the performance in applied scenarios and raises the Red Queen Effect. The ever-changing fitness landscape caused by the species competition makes the system dynamic more complex and is referred to as "Red Queen Effect" [15]. Using the same process of solving in different situations does not yield positive results and even misguides the search process.

The aim of the study is to develop a new algorithm with continuous self-adaption and improving the algorithm to solve different problem instances, GE with Adaptive BBs (GEABB), with the integration of PSO. Based on the existing works, we designed a hybrid model that self-adapts to different problem instances without changing its initially provided BBs set. We address the following research questions:

- How can GEABB dynamically change and control BBs in solving different problem instances?
- Can GEABB adapt to different problem instances without changing the initial set of BBs?
- Does GEABB reduce the iteration count and computational time compared to standard PSO?

The paper is structured as follows: Preliminary works are discussed in Section 2, the integration of PSO and GE is described in Section 3, while the BBs selection method is described in Section 4. A detail description of the proposed method is given in Section 5. The experimental setup is given in Section 6 and the performance of the algorithm measured as obtained from the conducted experiments is discussed in Section 7. Section 8 concludes the paper as well as the possible future research work that can be done.

2 Preliminary Works

Many variants of PSO have been developed for solving different kind of problem instances. One such variant is focusing on BBs [16]. As mentioned above, BBs are the sets of rules or processes that are the primary source for building the algorithm involved in the search process. In PSO, BBs such as inertia weight, constriction coefficient, acceleration coefficient, topologies and population size have a great influence on the performance [17].

Researchers have used various BBs and implemented different strategies to improve the PSO and evolutionary algorithms [18]. This shows the importance of BBs in solving different problem instances. Parameter tuning and

parameter control are the two types of BBs settings in the literature of evolutionary algorithms [19]. During parameter tuning, the BBs are configured before implementing the optimisation process. These tuned values of the BBs can be used in a variety of optimisation processes. Some of the recent parameter tuning methods are: Iterated Racing for Automatic Algorithm Configuration (IRACE) [20], ANOVA - a racing algorithm for configuring meta-heuristics [21], and the F-Race Algorithm [22]

As the optimal value may differ at different stages of a run, static parameter tuning may result in poor performance. The optimisation process starts with initial configuration of BBs and then the values change dynamically during the iteration process. This leads to parameter control, introduced by [19]. It is based on what is changes in BBs and when is the change made and how is the change made. BBs can be controlled by applying rules or adaptively. The deterministic rule is used to control the BBs value in the iterations. This gives better performance of the optimisation process for some cases but not in all cases.

In [14], have proved that a basic PSO gives satisfactory performance when its BBs are tuned rightly. BBs plays a prominent role in the algorithm performance and distinct settings are require for a problem instance as well as for different optimisation problem instances. Various PSO variants focusing on BBs is presented and discussed by [17]. The following BBs were observed as perspectives of development in PSO algorithm. The inertia weight is the initialisation of swarm particles with velocity [23]. The constriction factor ensures the trade-off between exploration and exploitation [24]. Cognition and social velocity models of the swarm indicate the attraction of the particles towards the global best in the feasible neighbourhood and usually converge faster with predominantly exploratory behaviour [25]. Cognitive and social acceleration coefficients are weights that capture how much a particle should weigh moving towards its cognitive attractor or its social attractor [26]. The swarm topologies establish swarm particles connectivity of its members to the others. Analysis of convergence shows that particles converge to a single point which is a cognitive attractor. Velocity and position update the mechanisms of the swarm particles. Multiple swarms can deal with multi-modal problems where multiple optima exist. Quantum particles are used as a guaranteed search method for finding a global convergence and artificial best particles [13].

The research gap among all these approaches is that they are not continuously self-adapting and improving the algorithm to solve different problem instances. Based on the identified BBs in PSO and the popular hybridisation method with GE, we build on these existing works and propose a new approach focusing on continuous self-adaption. With this implementation, the PSO with GE will be capable of self-adapt its own parameter settings in solving different problem instances without any manual interpretations. As best of our knowledge, this is the first approach introducing with the concept of scoring and ranking BBs in-order to help the algorithm self-adaptable to different problem instances.

3 PSO with GE

GE is a grammar-based form of genetic programming. It can operate on different data types, and grammar is selectable to make GE beneficial over traditional genetic programming [27]. GE is governed by the Backus-Naur Form (BNF) grammar [28], a notation technique that is used to define the syntax for communicating languages in computing, computer programming, instruction sets, and communication protocols. This grammar makes GE flexible across several problem domains with robust performance [29]. GE adopts a population of either linear genotypic integer strings or binary strings, which are transformed into functional phenotype through a genotype-to-phenotype mapping using grammar - a set of rules that describes the syntax of the program, a fitness function - assess the quality (the cost or fitness) of a program, and a search engine - to find the optimal solution for the generated program [29].

3.1 Grammar

GE uses the BNF grammar to generate programs (algorithms) to solve a problem [28]. The BNF grammar is represented by a tuple containing four sets, $< T, N, S, P >$, where T is the set of terminals, N is the set of non-terminals, S is the start symbol (a member of N), and P is a set of production rules [30]. The swarm particles consist of a vector of integer numbers $\mathbf{C}$ (named codons) which are decoded using the grammar defined in a tuple as stated above and passed to a mapper in the GE.

3.2 Search Engine

PSO imitates the social behaviour of a flock of flying birds to solve complex scientific problems [31] and is widely used to solve engineering problems [32]. The swarm particles (candidate solutions) fly in an dimensional search space, $n \geq 1$. Each particle has two associated properties: a current position and a velocity. Each particle remembers the best location in the search space found so far (*Pbest*), and knows the best location found to date by all particles in the population (*Gbest*). Therefore, at each step, the velocity of each particle is a function of its own history and the social influence of its peer group [13]. For each particle, the velocity vector v_i and position vector x_i is updated by Eq. (1) and Eq. (2) respectively.

$$\mathbf{v}_{ij}(t+1) = w\mathbf{v}_{ij}(t) + \mathbf{r}_1(t)C_1(Pbest_{ij}(t) - \mathbf{x}_{ij}(t)) + \mathbf{r}_2(t)C_2(Gbest(t) - \mathbf{x}_{ij}(t)) \tag{1}$$

$$\mathbf{x}_{ij}(t+1) = \mathbf{x}_{ij}(t) + \mathbf{v}_{ij}(t+1), \tag{2}$$

where r_1 and r_2 are random vectors with magnitudes, $r_1 \geq 0, r_2 \leq 1$, regenerated after every velocity update. C_1 and C_2 are the weights associated with *Pbest* and *Gbest*. w is an inertia coefficient, usually between 0.8 and 1.2. $\mathbf{v}_{ij}(t)$ is the velocity of the particle at time t, and $\mathbf{x}_{ij}(t)$ is the position of the particle

at time t, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$. $Pbest_{ij}(t)$ is the particle's individual best solution found at time t, and $Gbest(t)$ is the swarm's best solution found at time t.

Pbest and *Gbest* fitnesses and positions are updated by comparing the newly evaluated fitness against the previous *Pbest* and *Gbest* fitnesses as necessary. The objective function can be selected as maximisation or minimisation depending on the objective of the search engine. If the objective is minimisation, the objective function, min *Pbest* is updated by Eq. (3),

$$Cost(\mathbf{x}_i(t+1)) < Cost(Pbest_i(t)) \rightarrow Pbest_i(t+1) = \mathbf{x}_i(t+1). \quad (3)$$

In case of maximisation the objective function, max *Pbest* is updated by Eq. (4),

$$Cost(\mathbf{x}_i(t+1)) > Cost(Pbest_i(t)) \rightarrow Pbest_i(t+1) = \mathbf{x}_i(t+1). \quad (4)$$

3.3 Mapper

The mapper function converts a genotype (individual solutions) into a phenotype (evaluable solution to the problem) [33]. The conversion from genotype to phenotype is done using Eq. (5).

$$Rule = C\%R, \quad (5)$$

where C is the codon integer value, R is the number of rule choices for the current non-terminal symbol, and % is the modulus rule [34].

Initially, the mapper function begins by mapping the starting symbol into terminals. It converts each codon to its corresponding integer value. To use it, one first converts all codon values to integers, then, starting from the starting symbol, applies the mapping rule to convert the leftmost non-terminal into a terminal until all non-terminals have been converted into terminals.

4 Hybrid Approach for BBs Selection

For the BBs selection we make use of two selection approaches. The first, elitist selection, is a selection technique that selects a small number of BBs with the highest scores that avoids the crossover and mutation operations in the evolutionary process. The proposed BBs set is provided in Table 1, the ranking process is further explained in Section 5.

The second approach, fitness proportionate selection, also known as roulette wheel selection, is a genetic operator for choosing potentially useful recombination solutions in evolutionary algorithm. In the Roulette Wheel selection the blocks are given a probability P_i of being selected, calculated by

Eq. (6),

$$P_i = \frac{1}{N-1} * \left(1 - \frac{S_i}{\sum_{j \in BBs} S_j}\right), \tag{6}$$

where S_i is the value of scores for the individual block, N is the number of BBs, $i = 1, .., N$. In each evolution, the algorithm calculates the probability for each block based on the scores. Because the scores change after each iteration, the algorithm calculates the probability of each block in each evolution. The BBs are represented as a circular wheel using the probability values (percentages), and a fixed point is chosen on the wheel's circumference. After rotating, the wheel stops at a point, which determines the BB chosen for the evolutionary process.

In order to achieve better performance with GEABB algorithm, we developed a hybrid approach, by combining the elitist selection and the roulette wheel selection in one.

The hybrid approach selects BBs based on high ranking (from the elitist selection), *Parent 1* and probability (from the roulette wheel selection), *Parent 2*. Crossover operation is applied on *Parent i, i=1,2*, with 0.75 probability to generate five children. This probability and number of children were found by tuning using IRACE. The generated children will carry out the evolutionary process in finding the optimal solution based on the objective function defined as either minimisation or maximisation.

5 Proposed Framework

The proposed system architecture in Figure 1 gives an overview of GE integrated with PSO and scoring function for BB recommendation. The pseudo-code for the proposed framework, is shown in Algorithm 1.

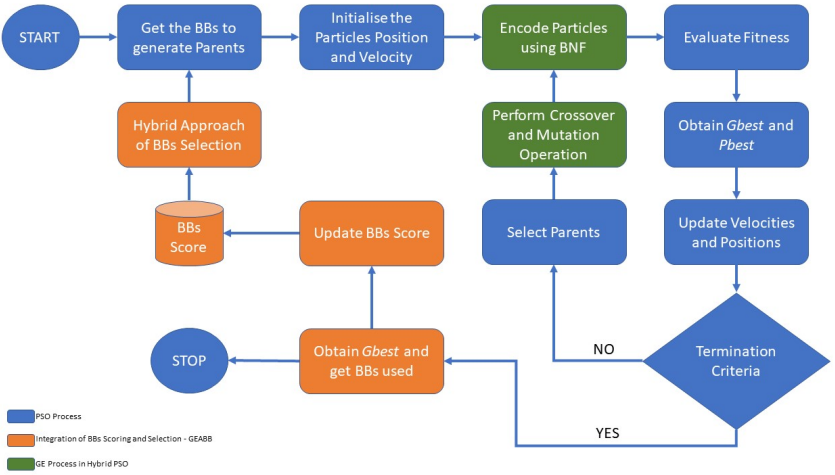

Fig. 1 Integration of GE with PSO for providing adaptive BBs. The bright orange blocks in the flowchart represent the new elements introduced in our novel GEABB framework.

The inputs for GEABB algorithm are the BNF grammar, and BBs score. The BBs used for the evolutionary process are shown in Table 1. After getting the inputs, the algorithm initialises the particles P as a population with random position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, personal best $Pbest$, and an iteration count of $n = 0$.

Table 1 BBs structure in BNF

Building Blocks	<Population Size>| <Iterations>| <N_Swarms>| <Velocity>| <Acceleration>| <Update Method> | <Search Mode>| <Topologies>| <Inertia Weight>
<Population Size>::=	100 | 200 | 300 | 400 | 500
<Iterations>::=	25 | 50 | 75 | 100
<N_Swarms>::=	1 | 2 | 3 | 4 | 5
<Velocity>::=	1.5
<Acceleration>::=	<C1>| <C2>
<C1>::=	0 | 2.05 | 2
<C2>::=	0 | 2.05 | 2 | 2.8
<Update Method>::=	<Synchronous>| <Asynchronous>
<Inertia Weight>::=	<Random>| <Linear Time Varying>| <Non Linear Time Varying>| <Fuzzy Adaptive>
<Search Mode>::=	<Point Based>| <Reference Based>
<Topologies>::=	<Pyramid>| <Random>| <Ring>| <Star>| <Von Neumann>

This is followed by particle encoding for all particles using the mapper Eq. (5) with the grammar by the Mod (or modulus) rule. Given the non-terminal

Algorithm 1 GEABB Algorithm

Require: Grammar, $\text{BBs}_{\text{score}}$

Hybrid Approach of BBs Selection
Get the population size ← $\text{BBs}_{\text{PopulationSize}}$
Generate parents based on selected BBs
for Every particle i=1...n **do**
    Initialise the position x_i
    Initialise the velocity v_i
end for
for Every particle in i=1...n **do**
    Encode particles according Eq. (5) using *grammar*
    Evaluate fitness according Eq. (7)
end for
while ¬ Stopcondition ($n \geq n_{max}$) **do**
    for Every particle i=1...n **do**
        Update position according Eq. (2)
        Update velocity according Eq. (1)
        if Objective function is minimisation **then**
            Update *Pbest* according Eq. (3)
        else
            Update *Pbest* according Eq. (4)
        end if
        Select the best performing parents
        Perform crossover to generate new children's
    end for
end while
Get the *Gbest*

<*op*>, which describes the set of mathematical operators that can be used elsewhere in the grammar, there are four production rules to select from. The choices are effectively labelled with integers counting from zero.

```
<op>    ::= +       (0)
          - & (1)
          * & (2)
          / & (3)
```

If we assume the codon produces the integer 6, then: *6 % 4 = 2*, would select rule *(2) **. Therefore, the non-terminal <*op*> is replaced with the terminal * in the derivation string. Each time a production rule is selected to transform a non-terminal, another codon is read. In this way the system traverses the genome.

This is followed by evaluation of the fitness of the particles with Eq. (7),

$$Fitness = E(P), \tag{7}$$

where E is the evolution function and P is the individual phenotype. The fitness gradient ranks the fitness of solutions in binary terms of better or worse on any single given objective. The determination of how one single objective fitness value is considered superior to another is done through maximisation or minimisation.

In the case of maximisation, *Pbest* is updated by Eq. (4), while for minimisation by Eq. (3). At the same time, particles update their position and velocity using Eqs. (2) and (1) respectively.

If the Termination Criteria $n \geq n_{max}$ is met, the algorithm decodes *Gbest* and the output is the best solution found so far. Otherwise, $n = n + 1$, and the algorithm selects the best-performing parents and performs a crossover using the crossover probability value of 75%. This means that it generates a new population set that has a 75% probability of having a population set the same as the previous one, after which it goes back to the encoding process step described above.

This process continues until the optimal solution is found or $n \geq n_{max}$. It ends with the final optimal solution found. Along with that, the algorithm obtains the BBs used in the current evolutionary process that provided the optimal solution and increments the scores by one value for those blocks.

6 Experiments

The objective of the experiments serves two purposes. The first is to identify the significance level of GEABB when compared to PSO. The second is to compare the run-time and iteration count of GEABB and PSO. Each set of experiments was repeated 30 times.

The proposed algorithm GEABB was tested with CEC'2013 [35] benchmark problems which are designed with the aim of providing a suitable evaluation platform for testing and comparing large-scale global optimization algorithms. It has three fully-separable functions ($f_1 - f_3$) , eight partially additively separable function; four functions with a separable sub component ($f_4 - f_7$) and four functions with no separable sub components ($f_8 - f_{11}$), three overlapping functions ($f_{12} - f_{14}$) and one non-separable function (f_{15}).

The description of these benchmark problems, Definition and Range are presented in Table 2. [35] gives the detailed description of these global benchmark problems.

In order to compare GEABB with PSO, we tuned the parameters of both algorithms using the IRACE package [36]. The tuned parameter settings for these experiments are shown in Table 3.

The PC Configuration for conducting these experiments is with Windows 10 operating system with AMD Ryzen 7 2700 Eight Core 3.20 GHz processor with 32GB of ram. The application used is Spyder 4.2.0.

Table 2 CEC'2013 state of the art benchmark problems

Description	Definition	Range
Shifted Elliptic	$f_1(Z) = \sum_{i=1}^{D} 10^6 \frac{i-1}{D-1} z_i^2$	[-100,100]
Shifted Rastrigin	$f_2(Z) = \sum_{i=1}^{D} [z_i^2 - 10\cos(2\pi z_i) + 10]$	[-5,5]
Shifted Ackley	$f_3(Z) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right)$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right) + 20 + e$	[-32,32]
7-nonseparable, 1-separable Shifted and Rotated Elliptic	$f_4(Z) = \sum_{i=1}^{\lvert S\rvert-1} w_i f_{elliptic}(Z_i)$ $+f_{elliptic}(Z_{\lvert S\rvert})$	[-100,100]
7-nonseparable, 1-separable Shifted and Rotated Rastrigin's	$f_5(Z) = \sum_{i=1}^{\lvert S\rvert-1} w_i f_{rastrigin}(Z_i)$ $+f_{rastrigin}(Z_{\lvert S\rvert})$	[-5,5]
7-nonseparable, 1-separable Shifted and Rotated Ackley's	$f_6(Z) = \sum_{i=1}^{\lvert S\rvert-1} w_i f_{ackley}(Z_i)$ $+f_{ackley}(Z_{\lvert S\rvert})$	[-32,32]
7-nonseparable, 1-separable Shifted Schwefel's	$f_7(Z) = \sum_{i=1}^{\lvert S\rvert-1} w_i f_{schwefel}(Z_i)$ $+f_{schwefel}(Z_{\lvert S\rvert})$	[-100,100]
20-nonseparable Shifted and Rotated Elliptic	$f_8(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{elliptic}(Z_i)$	[-100,100]
20-nonseparable Shifted and Rotated Rastrigin's	$f_9(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{rastrigin}(Z_i)$	[-5,5]
20-nonseparable Shifted and Rotated Ackley's	$f_{10}(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{ackley}(Z_i)$	[-32,32]
20-nonseparable Shifted Schwefel's	$f_{11}(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{schwefel}(Z_i)$	[-100,100]
Shifted Rosenbrock's	$f_{12}(Z) = \sum_{i=1}^{D-1} [100(z_i^2 - z_{i+1})_2$ $+(z_i - 1)^2]$	[-100,100]
Shifted Schwefel's Function with Conforming Overlapping Subcomponents	$f_{13}(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{schwefel}(Z_i)$	[-100,100]
Shifted Schwefel's Function with Conflicting Overlapping Subcomponents	$f_{14}(Z) = \sum_{i=1}^{\lvert S\rvert} w_i f_{schwefel}(Z_i)$	[-100,100]
Shifted Schwefel's	$f_{15}(Z) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	[-100,100]

7 Results and Discussion

The results obtained when testing with benchmark problems are shown in Table 4. The outcomes obtained by GEABB and PSO using 15 benchmark problems are presented. To evaluate the performance, we used the two tail t-test, a hypothesis testing used to determine the GEABB significance over PSO.

The mean, standard deviation and level of significance of both algorithms are calculated with most commonly used significance level p as 0.05 for 15 benchmark problems to determine the significance of GEABB. On all tested problems, GEABB performed better than PSO with significance greater than 95%.

Table 3 Tuned parameter settings for the experiments

Parameter	GEABB	PSO
P	-	500
n_{max}	-	75
Selection Proportion	0.5	-
Crossover Probability	0.75	-
C1	-	1.55
C2	-	1.98
w	-	0.73
Tournament Size	2	-
Initial Genome Length	150	-
Number of runs for each problem	30	30

Figure 2 and 3 shows the mean run time and iteration count of GEABB and PSO for 15 benchmark problems. For each problem, 30 runs of GEABB and PSO were performed and the collected data for run time and iteration count were averaged.

The results show, that using adaptive BBs in GEABB reduced significantly the run time in most cases compared to PSO. In a few cases, f_3, f_6, f_{10}, f_{12}, a small difference in run time is observed, however overall GEABB run time is better than the PSO. GEABB took much less iterations in getting the optimal solution compared to PSO.

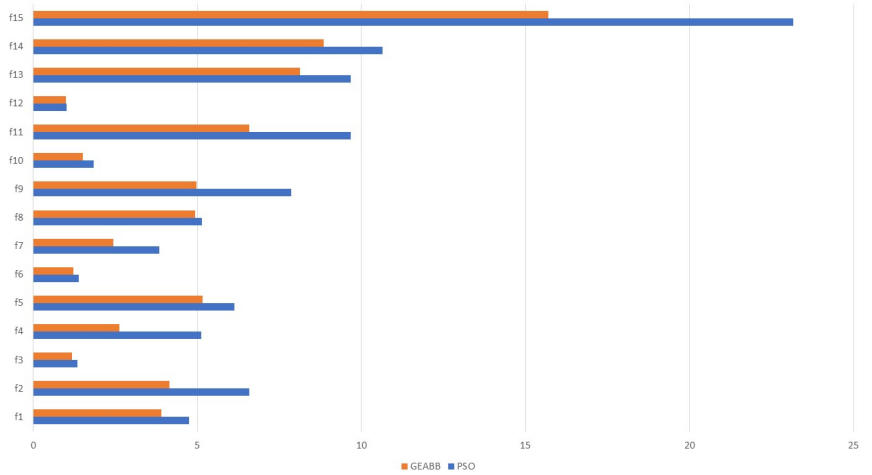

Fig. 2 The comparison of mean run time (in sec's) of PSO and GEABB with CEC'2013 benchmark functions. Shorter bar length indicates less computational time, thus more computationally efficient algorithm.

Table 4 Two-Tail t-Test significance level calculation and performance comparison of PSO and GEABB in CEC'2013 benchmark problems

Function	Approach	Mean	SD	Significance
f_1	PSO	8.46E-04	5.83E-05	
	GEABB	1.11E-04	1.96E-04	>95%
f_2	PSO	8.63E-04	5.86E-05	
	GEABB	2.96E-05	0.96E-05	>95%
f_3	PSO	6.11E-03	6.83E-01	
	GEABB	2.79E-05	1.11E-04	>95%
f_4	PSO	25.86	19.45	
	GEABB	7.86E-05	2.09E-04	>95%
f_5	PSO	9.56E-04	3.53E-04	
	GEABB	5.98E-05	1.86E-04	>95%
f_6	PSO	11.73	5.23	
	GEABB	4.11E-05	1.23E-04	>95%
f_7	PSO	3156.89	1742.63	
	GEABB	2986.21	1232.47	>95%
f_8	PSO	6.23E-04	7.87E-05	
	GEABB	1.03E-04	1.97E-04	>95%
f_9	PSO	5.23E-04	4.12E-05	
	GEABB	2.36E-05	1.89E-05	>95%
f_{10}	PSO	10.78E-03	6.59E-03	
	GEABB	3.54E-04	1.26E-04	>95%
f_{11}	PSO	15.63E-05	11.71E-04	
	GEABB	7.39E-05	3.91E-04	>95%
f_{12}	PSO	7.68E-04	6.89E-04	
	GEABB	1.73E-04	0.63E-04	>95%
f_{13}	PSO	11.27E-04	8.98E-04	
	GEABB	4.71E-04	1.44E-04	>95%
f_{14}	PSO	9.68E-03	6.34E-03	
	GEABB	2.56E-05	1.36E-05	>95%
f_{15}	PSO	7.42E-04	5.63E-04	
	GEABB	3.48E-04	1.98E-04	>95%

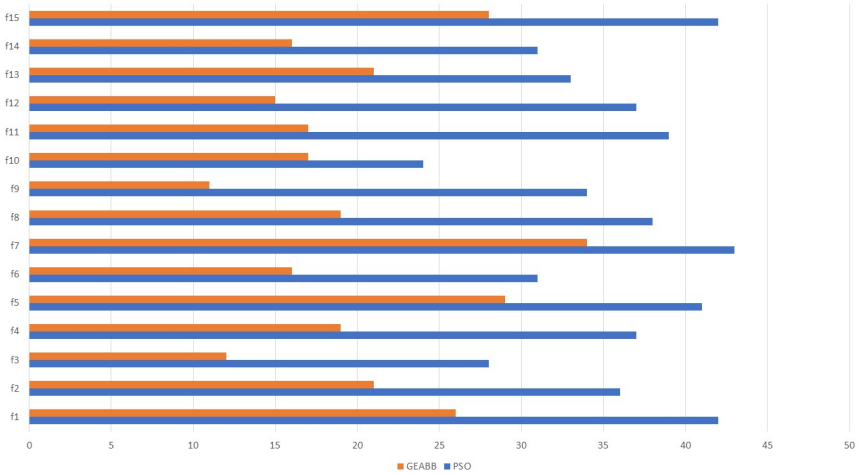

Fig. 3 Comparison of iteration count between PSO and GEABB with CEC'2013 benchmark functions. The shorter bar length indicates the better efficiency of the algorithm as it takes less iterations in finding optimal fitness. For example, *f3* and *f9*.

The results in Table 4, Figure 2 and 3, show that GEABB performed well on all tested 15 benchmark problems. Selecting adaptive BBs significantly improved the performance of the algorithm and reduced the computational time and iteration count. Furthermore, as the algorithm uses the previous best performed BBs, it increases the performance level (measured by computational time and iteration count) which in turn reduces the risk of Red Queen Effect.

8 Conclusion

In this paper, we propose a novel hybrid PSO approach, GEABB, providing adaptive BBs with the integration of GE. We implemented a hybrid approach for selecting BBs, combining elitist selection and roulette wheel selection. To make GEABB algorithm self-adaptable, we implemented a scoring function to score and select the BBs.

We tested the proposed algorithm GEABB on 15 global optimisation problems. In all tested scenarios, GEABB outperformed PSO by using less iterations and was computationally more efficient in finding the optimal solution.

The main contributions of the study are:

- A novel approach, GEABB, is proposed with scoring and recommending BBs making the algorithm self-adaptable to the problem instance in order to find the optimal solution.
- A hybrid approach is used to provide adaptive BBs for solving different problem instances without any manual interpretations to the blocks set.
- GEABB outperforms the PSO by taking fewer iterations and less computational time in all tested benchmark problems.

As an initial approach, the proposed model functioned well. Future work will address the self-adaptability of the algorithm by dynamically changing the blocks during the search process. This will make GEABB more robust and efficient in solving online and real-world problem instances.

Acknowledgments. JG thanks Deakin University for Deakin University Postgraduate Research Scholarship (DUPRS).

References

[1] Eberhart, R., Kennedy, J.: New optimizer using particle swarm theory. Proceedings of the International Symposium on Micro Machine and Human Science, 39–43 (1995). https://doi.org/10.1109/MHS.1995.494215

[2] Ghosh, M., Guha, R., Alam, I., Lohariwal, P., Jalan, D., Sarkar, R.: Binary genetic swarm optimization: A combination of ga and pso for feature selection. Journal of Intelligent Systems **29**(1), 1598–1610 (2020). https://doi.org/10.1515/jisys-2019-0062

[3] Zhao, F., Zhang, Q., Yu, D., Chen, X., Yang, Y.: A hybrid algorithm based on PSO and simulated annealing and its applications for partner selection in virtual enterprise. In: Lecture Notes in Computer Science, vol. 3644, pp. 380–389 (2005). https://doi.org/10.1007/11538059_40

[4] Shelokar, P.S., Siarry, P., Jayaraman, V.K., Kulkarni, B.D.: Particle swarm and ant colony algorithms hybridized for improved continuous optimization. Elsevier **188**(1), 129–142 (2007). https://doi.org/10.1016/j.amc.2006.09.098

[5] Ghodrati, A., Lotfi, S.: A hybrid CS/GA algorithm for global optimization. In: Advances in Intelligent and Soft Computing, vol. 130 AISC, pp. 397–404 (2012). https://doi.org/10.1007/978-81-322-0487-9_38

[6] Gao, Y.: An improved hybrid group intelligent algorithm based on artificial bee colony and particle swarm optimization. In: Proceedings - 2018 International Conference on Virtual Reality and Intelligent Systems, ICVRIS 2018, pp. 160–163 (2018). https://doi.org/10.1109/ICVRIS.2018.00046

[7] Zhao, F., Li, G., Yang, C., Abraham, A., Liu, H.: A human-computer cooperative particle swarm optimization based immune algorithm for layout design. Neurocomputing **132**, 68–78 (2014). https://doi.org/10.1016/j.neucom.2013.03.062

[8] Pan, T.S., Dao, T.K., Nguyen, T.T., Chu, S.C.: Hybrid particle swarm optimization with bat algorithm. Advances in Intelligent Systems and Computing **329**, 37–47 (2015). https://doi.org/10.1007/978-3-319-12286-1_5

[9] Wei, B., Xia, X., Gui, L., He, G., Xie, C., Xing, Y., Wu, R., Tang, Y.: A hybrid optimizer based on firefly algorithm and particle swarm optimization algorithm. Article in Journal of Computational Science **26**, 488–500 (2017). https://doi.org/10.1016/j.jocs.2017.07.009

[10] Shi, Y., Wang, Q., Zhang, H.: Hybrid ensemble PSO-GSO algorithm. In: Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012, vol. 1, pp. 114–117 (2013). https://doi.org/10.1109/CCIS.2012.6664379

[11] Nakane, T., Bold, N., Sun, H., Lu, X., Akashi, T., Zhang, C.: Application of evolutionary and swarm optimization in computer vision: a literature survey. IPSJ Transactions on Computer Vision and Applications **12**, 1–34 (2020). https://doi.org/10.1186/S41074-020-00065-9/TABLES/16

[12] Ryan, C., Collins, J.J., O'Neill, M.: Grammatical evolution: Evolving programs for an arbitrary language. Lecture Notes in Computer Science

(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **1391**, 83–96 (1998). https://doi.org/10.1007/BFB0055930

[13] Sengupta, S., Basak, S., Peters, R.A.: Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives. arXiv (2018) arXiv:1804.05319. https://doi.org/10.3390/make1010010

[14] Pedersen, M.E.H., Chipperfield, A.J.: Simplifying particle swarm optimization. Applied Soft Computing **10**, 618–628 (2010). https://doi.org/10.1016/J.ASOC.2009.08.029

[15] Floreano, D., Nolfi, S.: God save the red queen! competition in co-evolutionary robotics. Genetic Programming 1997: Proceedings of the Second Annual Conference, 398–406 (1997). https://doi.org/10.1.1.126.1780

[16] Jian, M.C., Chen, Y.P.: Introducing recombination with dynamic linkage discovery to particle swarm optimization. In: GECCO 2006 - Genetic and Evolutionary Computation Conference, vol. 1, pp. 85–86 (2006). https://doi.org/10.1145/1143997.1144010

[17] Shandilya, S.K., Shandilya, S., Deep, K., Nagar, A.K.: Handbook of research on soft computing and nature-inspired algorithms. Handbook of Research on Soft Computing and Nature-Inspired Algorithms, 101–132 (2017). https://doi.org/10.4018/978-1-5225-2128-0

[18] Sipper, M., Fu, W., Ahuja, K., Moore, J.H.: Investigating the parameter space of evolutionary algorithms. BioData Mining **11**, 1–14 (2018). https://doi.org/10.1186/S13040-018-0164-X/FIGURES/5

[19] Ágoston Endre Eiben, Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation **3**, 124–141 (1999). https://doi.org/10.1109/4235.771166

[20] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002

[21] Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. GECCO'02, pp. 11–18. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)

[22] Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. Experimental Methods for the Analysis of Optimization Algorithms, 311–336 (2010). https://doi.org/10.1007/978-3-642-02538-9_13

[23] Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S., Abraham, A.: Inertia weight strategies in particle swarm optimization. In: Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011, pp. 633–640 (2011). https://doi.org/10.1109/NaBIC.2011.6089659

[24] Eberhart, R.C., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation, CEC 2000, vol. 1, pp. 84–88 (2000). https://doi.org/10.1109/CEC.2000.870279

[25] Sousa-Ferreira, I., Sousa, D.: A review of velocity-type PSO variants. Journal of Algorithms and Computational Technology **11**(1), 23–30 (2017). https://doi.org/10.1177/1748301816665021

[26] Wang, H.C., Yang, C.T.: Enhanced Particle Swarm Optimization With Self-Adaptation Based On Fitness-Weighted Acceleration Coefficients. Intelligent Automation and Soft Computing **22**(1), 97–110 (2016). https://doi.org/10.1080/10798587.2015.1057956

[27] Sloss, A.N., Gustafson, S.: 2019 Evolutionary Algorithms Review (2019). https://doi.org/10.1007/978-3-030-39958-0_16

[28] O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation **5**(4), 349–358 (2001). https://doi.org/10.1109/4235.942529

[29] Ryan, C., O'Neill, M., Collins, J.J.: Handbook of Grammatical Evolution, pp. 1–497 (2018). https://doi.org/10.1007/978-3-319-78717-6

[30] Nicolau, M., Agapitos, A.: Understanding grammatical evolution: Grammar design. Handbook of Grammatical Evolution, 23–53 (2018). https://doi.org/10.1007/978-3-319-78717-6_2

[31] Reynolds, C.W.: FLOCKS, HERDS, AND SCHOOLS: A DISTRIBUTED BEHAVIORAL MODEL. Computer Graphics (ACM) **21**(4), 25–34 (1987). https://doi.org/10.1145/37402.37406

[32] de Almeida, B.S.G., Leite, V.C.: Particle swarm optimization: A powerful technique for solving engineering problems. Swarm Intelligence - Recent Advances, New Perspectives and Applications (2019). https://doi.org/10.5772/INTECHOPEN.89633

[33] Fagan, D., Murphy, E.: Mapping in grammatical evolution. Handbook of Grammatical Evolution, 79–108 (2018). https://doi.org/10.1007/978-3-319-78717-6_4

[34] Fenton, M., Mcdermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., O'neill, M.: Ponyge2: Grammatical evolution in python (2017). https://doi.org/10.1145/3067695.3082469

[35] Li, X., Tang, K., Omidvar, M., Yang, Z., Qin, K., China, H.: Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. gene **7**(33), 8 (2013)

[36] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002

Statements and Declarations

8.1 Funding

This research received no external funding. JG thanks Deakin University for Deakin University Postgraduate Research Scholarship (DUPRS).

8.2 Competing Interests

The authors declare no conflicts of interest.

8.3 Author Contributions

JG wrote the manuscript, developed algorithm, wrote the software and performed the experiments. JCB, RD and MA wrote the manuscripts and evaluated the experiments. JCB and MA designed the study. All authors have read and agree to the published version of the manuscript.

8.4 Data Availability

The CEC'2013 benchmark functions used in the current study are available in the mikeagn/CEC2013 repository, https://github.com/mikeagn/CEC2013