Mathematical and Computer Modelling 49 (2009) 709–720

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

Trait-based heterogeneous populations plus (TbHP+) genetic algorithm

Gokmen Tayfur [a,*], Hakki Erhan Sevil [b], Erkin Gezgin [b], Serhan Ozdemir [b]

[a] Department of Civil Engineering, Izmir Institute of Technology, Gulbahce Kampus, Urla, Izmir 35340, Turkey

[b] Department of Mechanical Engineering, Izmir Institute of Technology, Gulbahce Kampus, Urla, Izmir 35340, Turkey

ARTICLE INFO

Article history:
Received 27 November 2007
Received in revised form 6 August 2008
Accepted 7 August 2008

Keywords:
Genetic algorithm
Memory concept
Immunity
Instinct
Character fitness
Trait
Heterogeneous population

ABSTRACT

This study developed a variant of genetic algorithm (GA) model called the trait-based heterogeneous populations plus (TbHP+). The developed TbHP+ model employs a memory concept in the form of immunity and instinct to provide the populations with a more efficient guidance. Also, it has an ability to vary the number of individuals during the search process, thus allowing an automatic determination of the size of the population based on the individual qualities such as character fitness and credit for immunity. The algorithm was tested against the classical GA model in convergence and minimum error performance. For this purpose, 5 different mathematical functions from the literature were employed. The selected functions have different topological characteristics, ranging from simple convex curves with 2 variables to complex trigonometric ones having several hilly shapes with more than 2 variables. The developed model and the classical GA model were applied to finding the global minima of the functions. The comparison of the results revealed that the developed TbHP+ model outperformed the classical GA in faster convergence and minimum errors, which may be explained by the adaptive nature of the new paradigm.

1. Introduction

The classical GA model has been commonly employed in solutions of many engineering problems. It roughly involves the following steps: Creation of a randomly formed gene pool, selection, cross-over, mutation, and fitness evaluation. Fig. 1 shows a schematic diagram of a single cycle without the mutation operation. The gene pool represents the space of solutions. Through a proper selection process of the individuals, segments of chromosomes are swapped to generate a new set of solutions. To avoid getting trapped in local minima due to premature convergence, a proper amount of timed mutation is applied by flipping alleles or bits on the chromosome strings. Finally, the fitness values of each solution set are evaluated. This evaluation decides whether another cycle or generation is needed or fitness levels of a certain set are sufficient. More on the basics of GAs could be found in detail in [1,2], and even in [3].

This classical GA method although solves many problems, it would often, for some complex problems, take a large number of iterations to reach the optimal solution. Also, in some cases, it gets trapped in a local minimum. In addition, due to the perturbation effect, the solution may get unstable such that instead of going in the right direction it may totally diverge from the optimal solution.

This study intends to develop a variant of GA that would be faster and more efficient in reaching an optimal solution. The goal is to create an adaptive error-driven algorithm. The idea is to develop a simple GA code that would consider two features. Firstly, only the best parent, instead of two best, is to be used for reproduction. Secondly, cross-over is to be replaced by an error-driven mutation rate. The primitive version of this idea was put to test by Ozdemir and Karakurt [4] and Alpay et al. [5].

* Corresponding author.
E-mail addresses: gokmentayfur@iyte.edu.tr (G. Tayfur), serhanozdemir@iyte.edu.tr (S. Ozdemir).

0895-7177/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2008.08.016

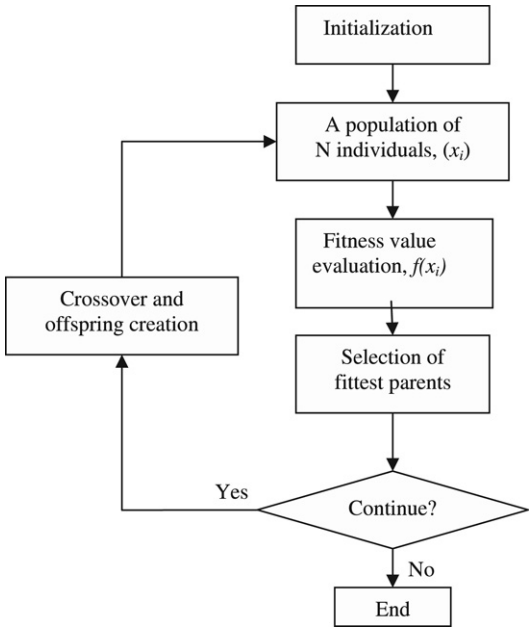

Fig. 1. Flow chart of conventional GA.

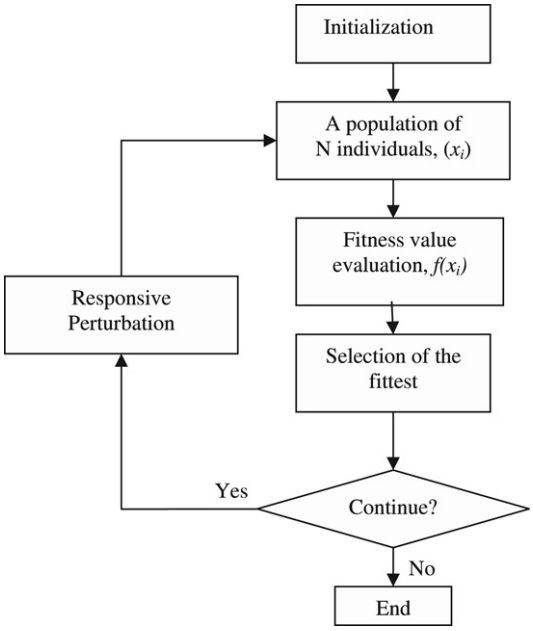

Fig. 2. Flow chart of RPA.

In [4], this paradigm was called the "Responsive Perturbation Algorithm (RPA)", which was population based and adaptive (Fig. 2). Basically, RPA uses the general philosophy of combining genetic algorithms (GAs) with simulated annealing (SA) in the very basic principles. When a population is created, individuals are not converted to binary strings, since no crossing occurs. Instead of selecting the best performing individuals, only the best is kept. Furthermore, perturbation replaces the crossing. A new family of individuals is created by perturbing the best solution with a radius that is a function of both character and the time (temperature in SA sense, [6,7]). This guarantees that all the offsprings are the variations of the best parent, not the offsprings of the best and the second best. Also as more iterations are run, solutions are 'tempered' as search radii get smaller and smaller, resulting in narrower search circles. Nevertheless, RPA had no advanced ageing and search schemes. RPA had an error-driven perturbation but its amplitude was driven more by a scheduled reduction than the reduction in error. Also, the lack of characters meant that the population distribution was not economically dispersed.

Table 1
Population distribution

Trait	Step 1	Step 2	Step 3
Greedy	0.4*N*	0.3*N*	0.05*N*
Curious	0.2*N*	0.1*N*	0.05*N*
Normal	0.1*N*	0.2*N*	0.25*N*
Conservative	0.1*N*	0.3*N*	0.6*N*
Fickle	0.1*N*	0.05*N*	0.03*N*
Erratic	0.1*N*	0.05*N*	0.02*N*

N: Total number of individuals.

It must be noted that variants of GAs are not the only alternatives to numeric problems, since numeric techniques are also well-established. One typical work that may show the historical perspective is [8].

A more advanced version of RPA was developed by Alpay et al. [5] under the name of "Trait-based Heterogeneous Populations (TbHP)". The most important feature of this new GA code was the implementation of the six characters that are explained in the following paragraphs in detail. This ensured a relatively economical distribution of the solutions where and when they are needed. This was achieved by spontaneous ageing that was triggered when one of the selected traits was not *the* dominant character during the running of the program. All the six traits were preserved in the new TbHP+ version.

RPA and TbHP were applied for training feed forward back propagation and the radial basis function neural networks with relatively fast convergence [5]. Alpay et al. [5] have shown the superiority of TbHP over RPA on training of RBFNN [radial basis function neural networks] on a specific machining data [9]. A similar study may be found in [7].

Although TbHP is quite robust and more advanced compared to conventional GA and RPA, it lacks the means to use the resources more efficiently. The objective of this study is to develop more advanced version of TbHP. This advance version [here called TbHP+] is equipped with more features that would overcome the shortcomings of TbHP. The details of TbHP+ are given below.

2. TbHP+ genetic algorithm

Instead of a population of homogeneous individuals, as it is the case for generic Genetic Algorithms (GAs) and RPA, a population of heterogeneous individuals has been set to compete in TbHP+. Here homogeneous refers to alike individuals. The term heterogeneous, on the other hand, describes dissimilar ones, varying in a hierarchy of radii and perturbation amplitude. The novelties are manifold. First of all, every individual is made to differ by assuming a character or trait. Individuals with the highest fitness become the current dominant character and yield offsprings that carry predominantly the traits of the winner parent. Initially, the majority of the population is deliberately made *greedy* (one of the group with large search radii) so as to put more individuals on the look out for better neighborhoods. Then, as the *ageing* (the reduction in the search radii of all traits and the increase in the number of individuals of relatively more *conservative* traits) starts, gradually more settled (conservative) traits make up the majority.

Human characteristics were imitated in creating the traits (*greedy, curious, normal conservative, fickle,* and *erratic*) (Table 1). Greedy is the trait with the most extensive search radius among the main traits. Curious is meant to cover an area hierarchically only second to greedy. Normal and conservative traits are poised at the bottom of the character hierarchy. The trait conservative has the lowest perturbation percentage among all the others. In other words, conservative solutions are the closest ones to the parent. Normal characters are second to conservative solutions. Fickle and especially erratic traits were meant to find the searched domain faster and keep the population out of the local stagnation zones. Since, as their names suggest, they are at the fringes of the population distribution, only a small percentage of the resources are committed to sustain these individuals (Table 1). However, erratic ones are the most unstable characters at the beginning. This situation changes considerably as they get older.

Ageing helps zoom in on the solution with an ever increasing precision (Fig. 3). Fig. 3a shows starting point and target location. Figs. 3b and c demonstrate the progress of the program at the initial stage and in time, respectively. At this stage, the erratic trait becomes dominant and allows fast advance towards the target. Fig. 3d depicts the final stage when ageing starts. Another aspect of ageing is that a greater percentage of the population is concentrated in and around the center. Not only the perturbation amplitude of the individuals is dampened to pinpoint the solution set but also more individuals are put to work close to it. In simple, nonscientific terms, the population debuts with a young greedy population, and ends up with an older population who are mostly conservative in nature. As individuals get older, the characteristics that are dominant at the beginning become progressively recessive.

In TbHP+, on the contrary to TbHP, more than one ageing policies have been implemented. The first one is based on a schedule, and the ageing is performed stepwise in a predetermined way. This has not turned out to be quite such a satisfactory technique. In the second method, called the spontaneous ageing, as long as greedy is the dominant character, no ageing takes place. Once the curious (second to greedy trait in search radius-wise) becomes successively the dominant character, ageing commences. Whenever greedy is not the dominant character, the optimal solution [target] falls within reach. In other words, at this instant, the target is closer to the center than the greedy individuals. Once the target is found, its exact coordinates must be determined. Determination of the coordinates is attained through the start of ageing which

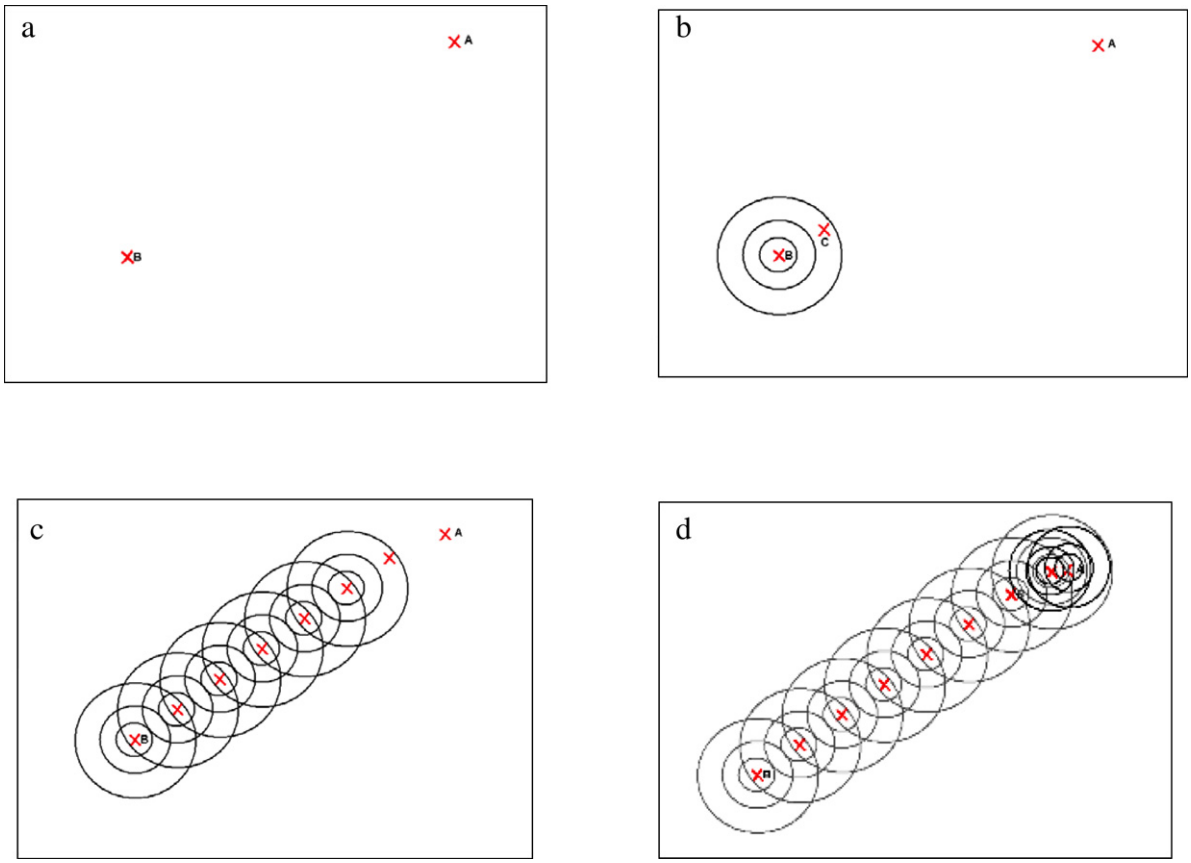

Fig. 3. Schematic representation of ageing process: (a) Starting point (point B) and target location (point A); (b) initial progress; (c) progress in time; and (d) final stage where ageing starts.

involves taking of two steps. The first is to deploy more individuals at the vicinity of the target and keep pulling them towards the center in phase with the target. The second is to reduce all the search radii simultaneously, so that all the populations are pulled inward. Aging process and traits are thus applied to accelerate convergence. Individuals with different characters are used to cover a search area and as the iterations continue their search area is reduced to find a neighbor-optimal solution set.

Table 1 expounds the population [traits] distribution in TbHP+. The distribution of the population is divided into three phases in the duration of the code. For example, the greedy trait assumes 40% of the population in the 1st, 30% in the 2nd, and 5% in the 3rd phase. The decision over the population distribution is completely experiential. The breakdown of population given in Table 1 is directed towards one goal only, tracking down the optimal solution. The first phase starts with the current choice of percentage where a majority of the candidate solutions are spread out towards the perimeter of the maximum search radius. Few are left inside this circle, in case, at the start, optimal solution vector may happen to lie already in the neighborhood. Since this probability is rather small, so is the percentage of the population near the center. Again referring to the Table 1, normal and conservative traits, which happen to be located close to the center of the search radii, constitute only the 20% of the whole. In other words, the belief that initially the solution lies close to the center is only 20%. The second phase is switched on when a candidate solution is detected at the perimeter. This leads to the start of the ageing process. Contraction of the population towards the center takes place. As it might be noticed from the same table, the "look-outs" at the perimeter are not called back. Traits such as greedy and fickle still keep searching for possible better solutions away from the center. However, this time, their perturbation amplitude and the distributions are reduced. Our belief, at this second phase, that the solution might be close to the center, namely, the population sums of normal and conservative characters is 50%. The last phase sets out with a strong belief that the solution is almost definitely in the vicinity of the center. The breakdown of the last stage from the table reveals now that this belief is around 85% with the final and conclusive reduction in the amplitudes of the perturbations. In this last stage, by the nature of ageing, traits come close to their lower neighbors in the traits hierarchy. For example, the normal trait in the last stage bears the features of the conservative trait in the first stage (see Table 1).

When the program is initiated, the solution is, in general, far from the starting point. Initially sparse and expanded populations congregate at the center. Hence, at the beginning, most of the population is around the perimeter of the searching area. As the iterations advance, populations try to find the best solution in the multiple search radii. Populations

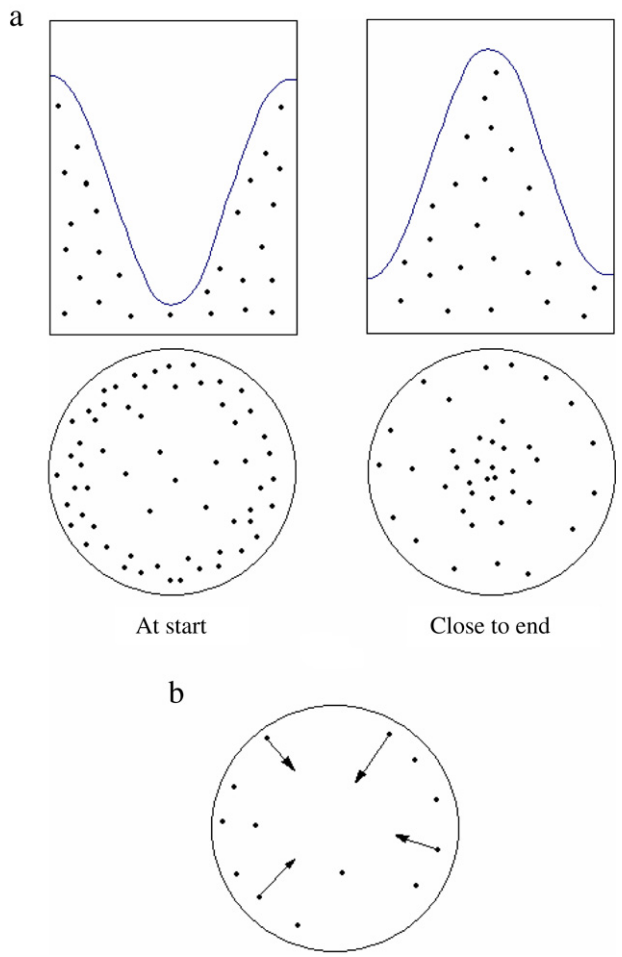

Fig. 4. Population migration (a) early stage; (b) central migration.

Table 2
Perturbation percentage

Trait	Step 1	Step 2	Step 3
Greedy	±20%–30%	±15%–20%	±10%–15%
Curious	±10%–20%	±7%–15%	±5%–10%
Normal	±2%–10%	±1%–7%	±0.5%–5%
Conservative	±0%–2%	±0%–1%	±0%–0.5%
Fickle	±30%–100%	±20%–50%	±15%–25%
Erratic	±100%–1000%	±50%–500%	±25%–100%

are not allowed to stay fixed in the domain but rather move around in an optimal fashion (called *population migration*), Fig. 4. At the early stages, the optimum solution falls in the search area of the erratic individuals, Fig. 4a. After a number of iterations, the best solution tends to move towards the center of the searching area triggering the *aging* process, Fig. 4b. This phenomenon is the by-product of the ageing process.

The search radius of each character is given in Table 2. Phases in this table designate stages of the running of the code. Phase 1 represents early stages until phase 2 which starts at around middle of the cycle, and the third phase takes place just about in the remaining 10% of the iterations. According to Table 2, for example, the individuals in the greedy trait are perturbed around the optimal solution by 20%–30% in the 1st, 15%–20% in the 2nd, and finally 10%–15% in the 3rd phase. The perturbation here is obtaining a set of neighboring solutions that are found by multiplying a predetermined amplification factor by the optimal solution.

In addition to the above mentioned characteristics, the TbHP+ contains features such as instinct (exploitation) and immunity (credit) (Fig. 5). Fig. 5a shows how TbHP+ would have proceeded in the absence of instinct, from a random starting point to optimal solution. Fig. 5b depicts the phase that the code "learns" the optimal direction. If there is no deviation from that course, to save the resources, only the solutions within a slice in the forward (learned) direction are computed, and

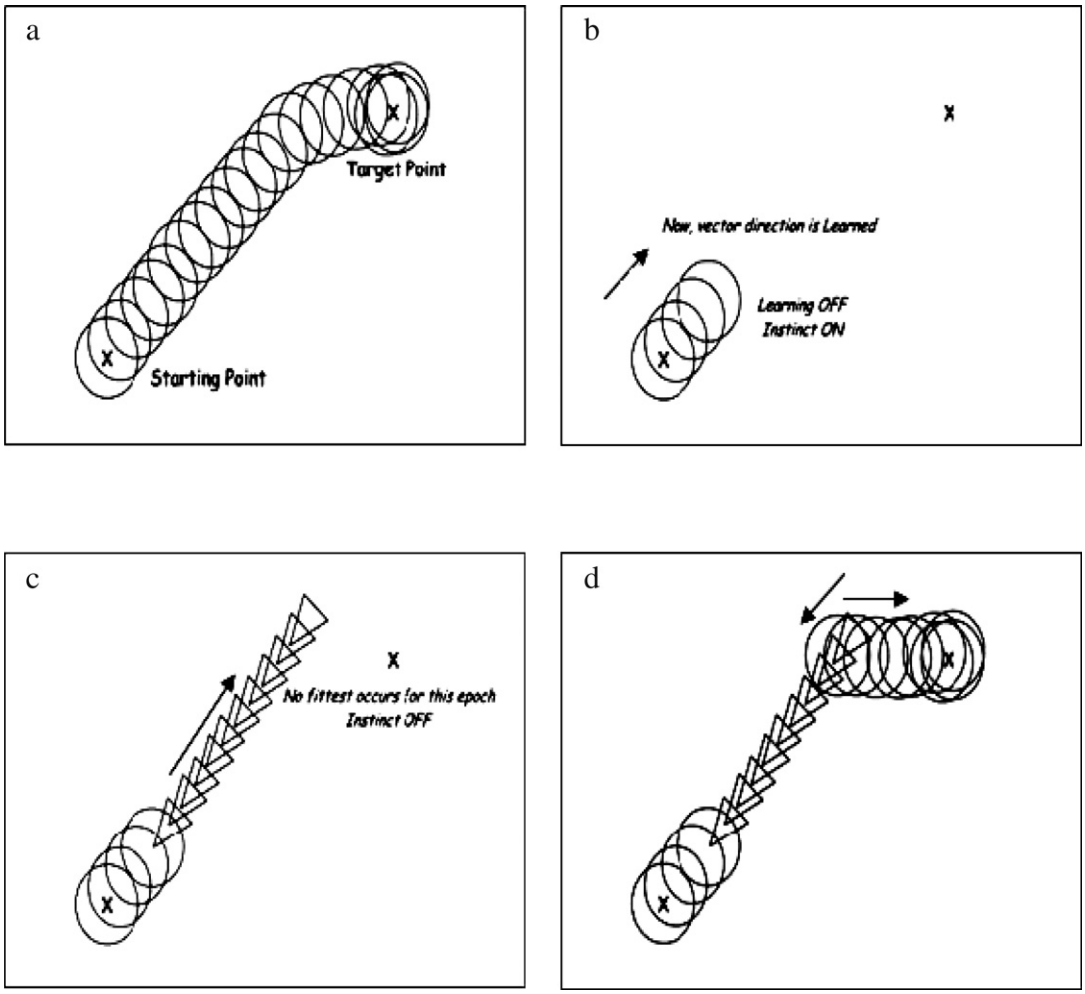

Fig. 5. Simulation of instinct: (a) phase of absence of instinct; (b) phase of optimal direction; (c) instinct mode phase; (d) switch of instinct mode phase.

the rest are discarded (Fig. 5c). This phase is called the instinct mode. When instinct mode gradually gives rise to increasing errors, this points to the fact that a new direction must be learned, and instinct mode must be switched off (Fig. 5d). The exploration versus exploitation dilemma has been articulated by various researchers in different forms. This is equally valid in computing and simulation arena. Exploiting, as it is already known, is cheaper than the effort to widen the current pool of information. This concern has been the basis to the newly developed concept of instinct in populations in this study. Instead of a life-long learning, either learning (exploration) or instinct (exploitation) is turned on and off while monitoring their performances. When the simulation is in the instinct mode, if the performance (fitness) worsens, it is switched off, and learning resumes. When the immunity is added to the algorithm, an important decrease in the successive error percentage and CPU time is achieved. During aging process, if a vectorial direction of the winner individuals is detected, by virtue of the instinct, the program continues by the half of the population in that direction until the results start deviating from optimal solution.

Fig. 6 roughly summarizes the algorithm for TbHP+. Note that code details were omitted for clarity.

3. Test Functions

We tested the performance of the developed TbHP+ model against 5 different functions that are summarized below.

1. **De Jong's 1st Test Function** [3]: It is continuous and has a convex shape (Fig. 7). It is commonly used in performance evaluation. The mathematical expression of the function, its global minimum, and the solution space are given as:

$$f(x) = \sum_{i=!}^{n} x_i^2, \quad -5.12 \le x_i \le 5.12 \text{ (solution space)}$$

$$f(x) = 0, \quad \text{when } x_i = 0 \text{ (global minimum point)}$$

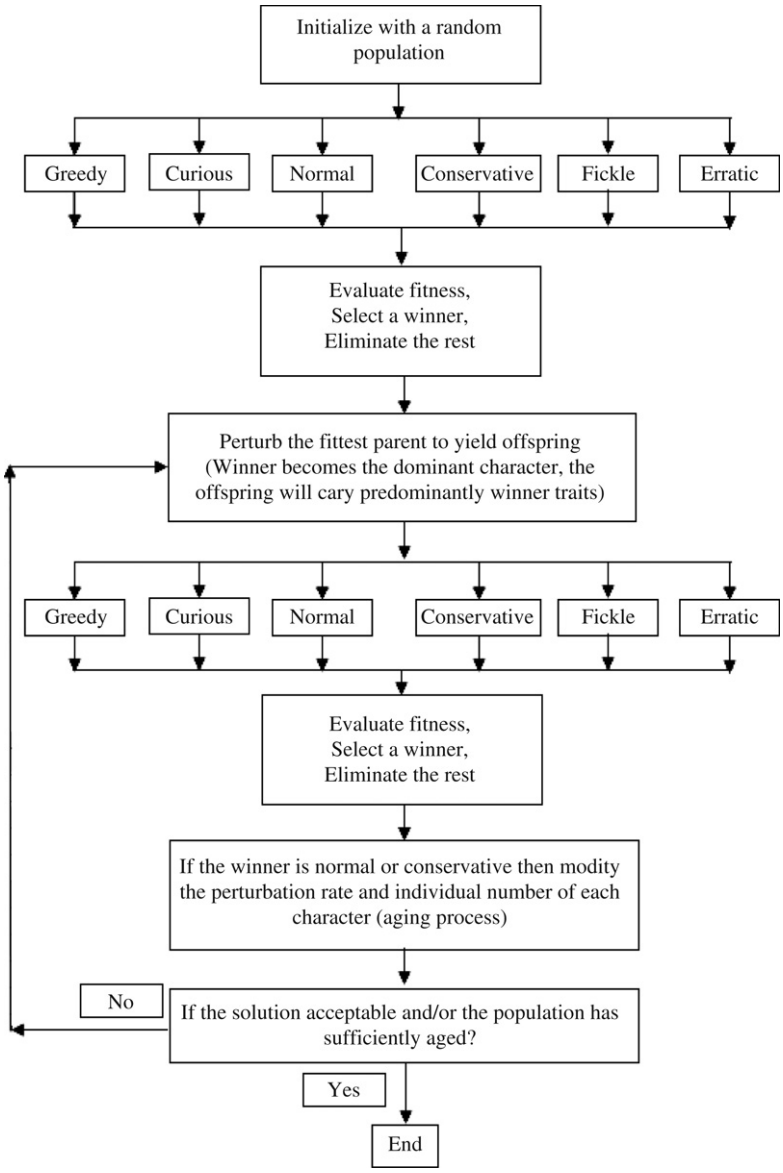

Fig. 6. Flowchart of the TbHP+ program.

when the number of variables is accepted as $n = 2$, the graph of the function is presented in Fig. 7 where one can see that the global minimum is at $x_1 = 0$ and $x_2 = 0$.

2. **Rastrigin's 6th Test Function** [10]: The mathematical form of the function, with the solution space and global minimum, is given below as:

$$f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), \quad -5.12 \le x_i \le 5.12$$

$$f(x) = 0, \quad \text{when } x_i = 0.$$

Assuming $n = 2$, the graph of the function is presented in Fig. 8 where one can see that the function has many local minimums.

3. **Michalewicz's 12th Test Function** [2]: It contains local minima as many as the factorial of variables, i.e. if the number of variables is 3, it would contain 6 local minima. In this study, we assumed 5 variables and thus there are 120 local minima. The mathematical expression of the function, with the specified solution space and the global minimum when $n = 5$, is

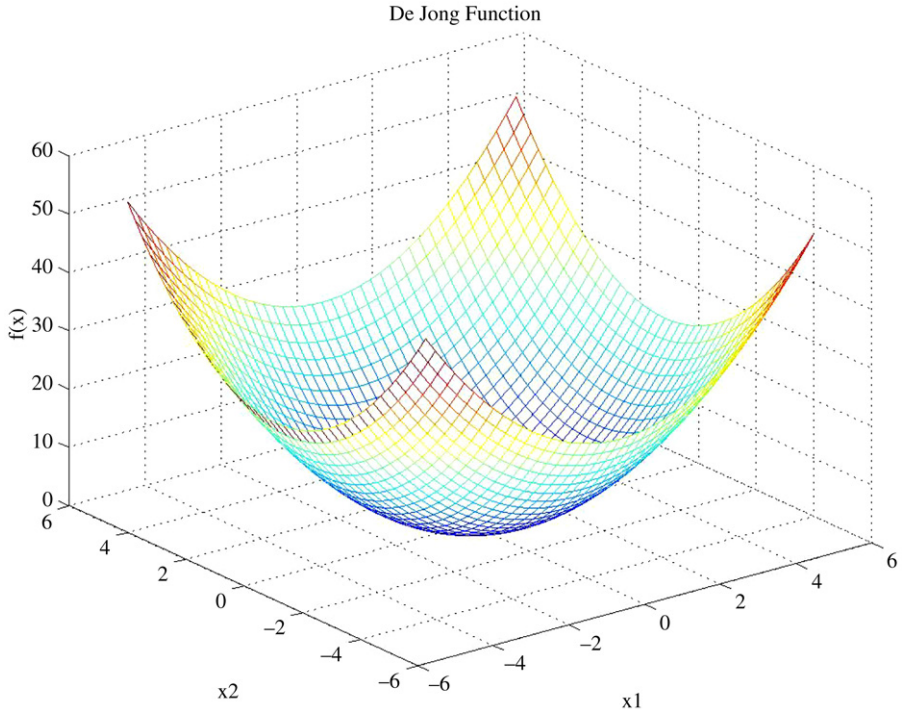

Fig. 7. De Jong's 1st Test Function and its global minimum [for $n = 2$ variables].

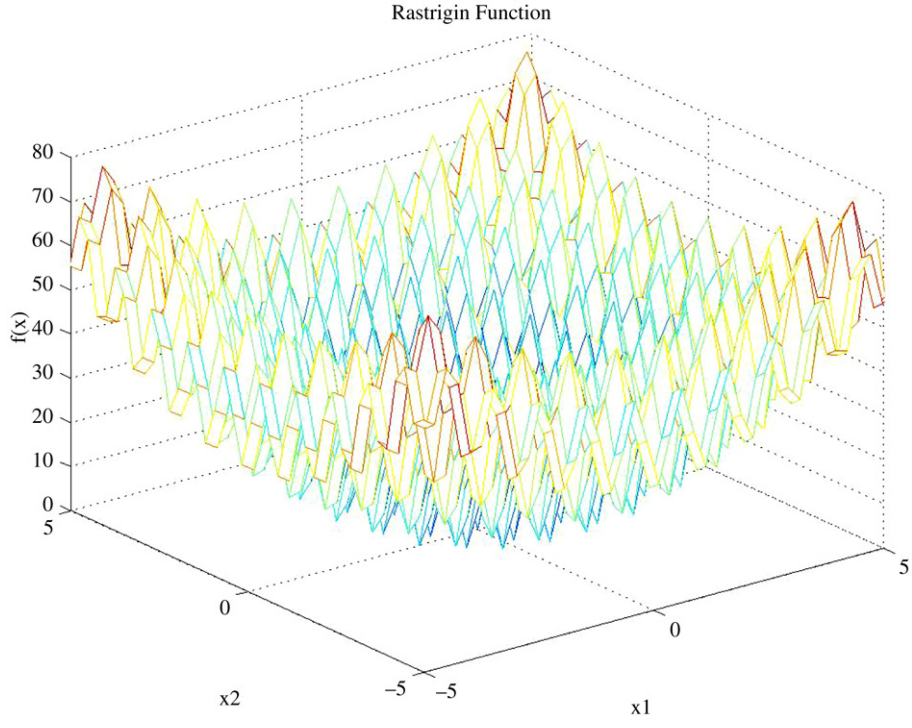

Fig. 8. Rastrigin's 6th Test Function and its global minimum [for $n = 2$ variables].

summarized as:

$$f(x) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin\left(\frac{i * x_i^2}{\pi} \right) \right]^{2m}, \quad 0 \le x_i \le \pi$$

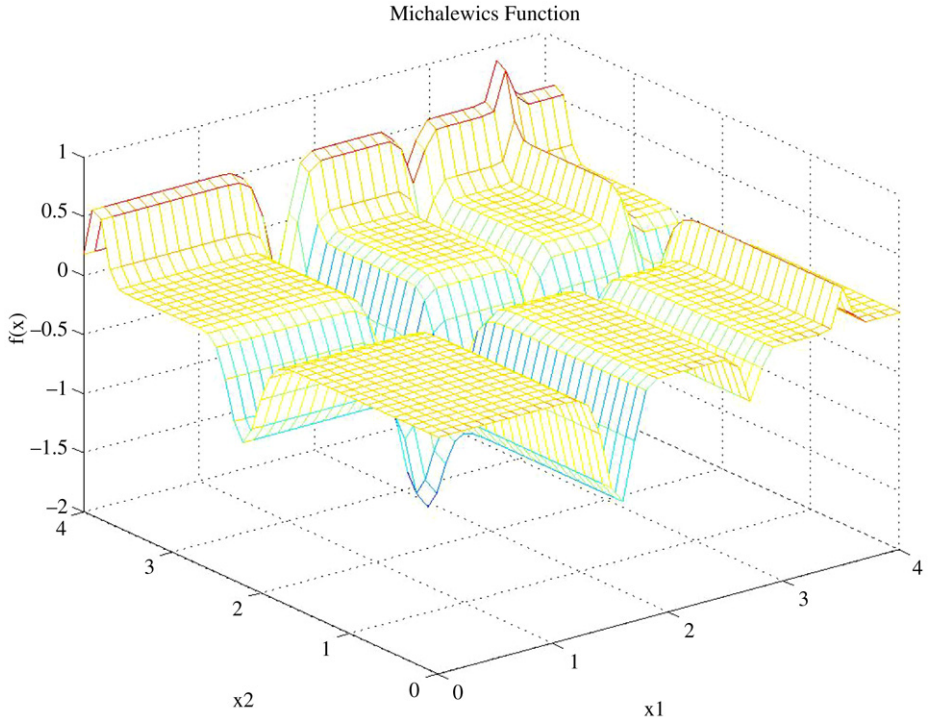

Fig. 9. Michalewicz's 12th Test Function and its global minimum [for $n = 2$ variables].

$f(x) = -4.687$, when $m = 10$ and $n = 5$ and $x_1 = 2.20895, x_2 = 1.57153, x_3 = 1.27608, x_4 = 1.91898, x_5 = 0.99552$.

Note that m shows the steepness of the hills in the solution space and therefore, for larger m values, it is harder to reach the global optimal solution. Assuming $n = 2$, the graph of the equation is presented in Fig. 9 where one can also see the global minimum position [for the sake of clarity, we showed $n = 2$ *variable* case in Fig. 9].

4. **Goldstein–Price's Test Function** [11]: This function has two variables. The mathematical form of the function, its specified solution space, and its global minimum location are given as:

$$f(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \cdot \left[30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]$$

$$-2 \le x_i \le 2, \quad i = 1, 2$$

$f(x_1, x_2) = 3 \quad x_1 = 0 \quad x_2 = -1$ [Global minimum position].

Fig. 10 shows the graph of the function having 2 variables where one can also see the global minimum position.

5. **Schwefel's 7th Test Function** [8]: The function, its specified solution space, and global minimum are given as:

$$f(x) = \sum_{i=1}^{n} -x_i \cdot \sin(\sqrt{|x_i|}), \quad -500 \le x_i \le 500$$

$f(x) = -837.9658 \quad x_i = 420.9687, \quad i = 1, 2.$

The graph of the function for $n = 2$ is presented in Fig. 11 where one can also see the global minimum position of the function within the specified solution space.

4. Model testing

The developed TbHP+ and the classical GA models, for performance comparison purpose, were applied to find the global minimums of the functions summarized above. Initially random values in between [0, 1] were assigned for the variables. Mean absolute error measure (MAE) is used to test the performance of the models. For each test function, the models were run 10 times and the mean of the mean absolute error of each run was computed for each model. Table 3 shows the summary of 10 runs by the models for each test function. For each run, 1000 iterations were performed. While the computer code for TbHP+ was developed by the authors, the package program Evolver was employed for the classical GA [12]. The algorithm employs the *Recipe Solving Method* to minimize (or maximize) any objective function under specified constraints [12].

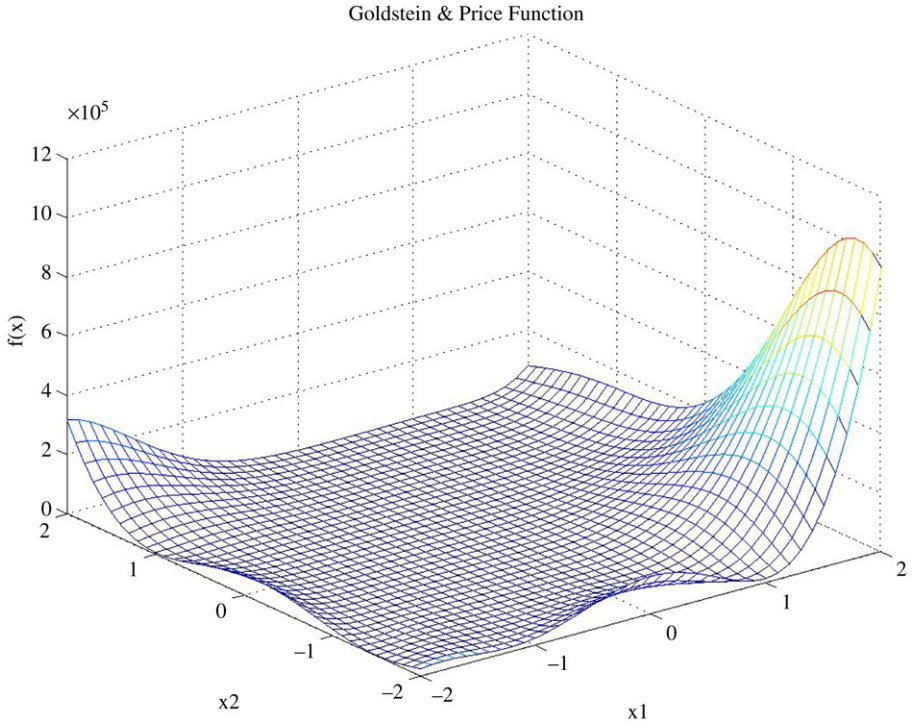

Fig. 10. Goldstein–Price's Test Function and its global minimum [for $x = 2$ variables].

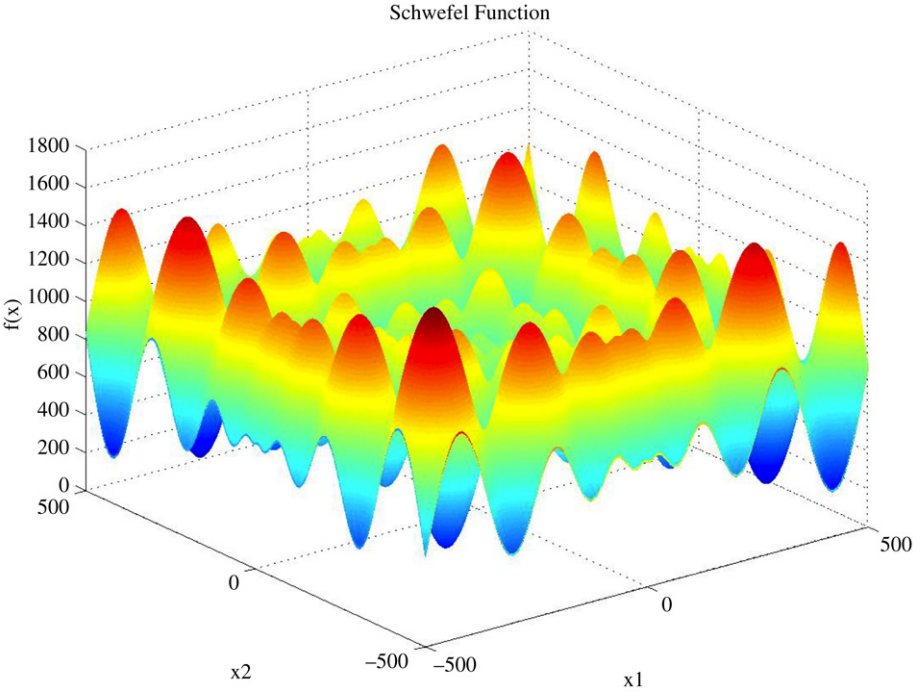

Fig. 11. Schwefel's 7th Test Function and its global minimum [for $n = 2$ variables].

As can be seen in Table 3, for each run for the first test function, TbHP+ model significantly outperformed the classical GA. For this case, the MAE = 96.7E−326 which is practically equal to zero (no error) for TbHP+ model versus MAE = 0.01317 for the classical GA model. For the second test model, TbHP+ also showed superior performance with MAE = 1.54E−18 [which means practically zero error] versus MAE = 0.8056 in the case of classical GA. For the 3rd model, TbHP+ had almost 50% less

Table 3
MAE for TbHP+ and the classical GA in finding global minimums of five different mathematical functions

Run	De Jong's 1st Test Function		Rastrigin's 6th Test Function		Michalewicz's 12th Test Function		Goldstein–Price's Test Function		Schwefel's 7th Test Function	
	MAE		MAE		MAE		MAE		MAE	
	TbHP+	Classical GA	TbHP+	Classical GA	TbHP+	Classical GA	TbHP+	Classical GA	TbHP+	Classical GA
1	3.34E−326	2.80E−02	4.70E−19	1.13E+00	2.77E−06	2.18E−01	5.58E−02	9.85E+01	3.12E−05	2.80E+00
2	1.77E−326	4.66E−02	6.02E−19	4.41E−01	1.47E−07	6.53E−02	1.54E−01	8.26E+01	2.76E−05	2.80E−01
3	1.24E−326	3.09E−03	2.46E−19	1.35E+00	1.49E−01	5.36E−02	2.58E−01	8.93E+01	2.70E−05	2.06E+00
4	2.53E−326	3.11E−03	2.83E−18	3.62E−01	1.63E−01	1.03E−01	6.91E−03	9.20E+01	4.55E−05	3.40E−01
5	1.89E−326	1.15E−02	2.69E−18	9.19E−02	1.49E−01	7.51E−02	3.83E−01	3.71E−01	3.85E−05	5.28E−01
6	1.44E−326	1.34E−02	2.69E−18	1.82E+00	3.55E−01	2.75E−02	2.43E−02	1.05E+02	2.81E−05	2.50E+00
7	2.84E−326	5.36E−03	3.41E−20	1.05E+00	1.62E−01	6.96E−01	2.46E−02	2.60E+00	2.79E−05	9.95E−02
8	2.08E−326	5.29E−04	2.53E−18	1.30E+00	1.91E−01	2.42E−01	3.97E−03	5.13E+00	3.48E−05	1.56E+00
9	2.05E−326	6.79E−04	8.66E−19	1.77E−01	4.12E−02	1.98E−01	1.95E+00	2.62E+00	2.91E−05	1.86E−01
10	1.78E−326	1.95E−02	2.42E−18	3.32E−01	3.26E−01	3.34E−01	2.75E−01	9.03E+01	3.66E−05	3.94E+00
Mean MAE	96.7E−326	1.32E−02	1.54E−18	8.06E−01	1.54E−01	2.01E−01	3.14E−01	5.69E+01	3.26E−05	1.43E+00

Table 4
Comparison of models with respect to number of iterations and corresponding errors

Trials	De Jong's 1st Test Function			Rastrigin's 6th Test Function			Michalewicz's 12th Test Function		
	MAE			MAE			MAE		
	Iteration #	TbHP+	Classical GA	Iteration #	TbHP+	Classical GA	Iteration #	TbHP+	Classical GA
1	143	96.7E−326	1.88E+00	109	1.54E−18	1.04E+01	221	1.54E−01	1.02E+00
2	154	96.7E−326	2.03E+00	97	1.54E−18	2.49E+01	540	1.54E−01	3.76E−01
3	152	96.7E−326	1.48E−01	24	1.54E−18	3.04E+01	548	1.54E−01	4.54E−01
4	147	96.7E−326	9.70E−01	170	1.54E−18	5.81E+00	618	1.54E−01	2.97E−01
5	153	96.7E−326	1.59E+00	23	1.54E−18	7.89E+00	533	1.54E−01	4.39E−01
6	147	96.7E−326	4.88E−01	457	1.54E−18	2.16E+00	316	1.54E−01	1.32E+00
7	151	96.7E−326	8.65E−01	51	1.54E−18	1.86E+01	601	1.54E−01	5.66E−01
8	153	96.7E−326	6.13E−01	89	1.54E−18	2.29E+00	807	1.54E−01	2.11E−01
9	149	96.7E−326	5.89E−02	54	1.54E−18	8.50E+00	478	1.54E−01	2.84E−01
10	147	96.7E−326	4.73E+00	245	1.54E−18	1.47E+00	424	1.54E−01	8.26E−01

Trials	Goldstein–Price's Test Function			Schwefel's 7th Test Function		
	MAE			MAE		
	iteration #	TbHP+	Classical GA	iteration #	TbHP+	Classical GA
1	692	3.14E−02	3.22E+01	693	3.26E−05	1.56E+00
2	751	3.14E−02	8.97E+01	382	3.26E−05	8.84E−01
3	732	3.14E−02	8.21E+01	353	3.26E−05	1.96E+00
4	78	3.14E−02	4.68E+00	306	3.26E−05	3.19E+00
5	805	3.14E−02	9.25E+01	798	3.26E−05	5.12E+00
6	984	3.14E−02	8.45E+01	794	3.26E−05	1.21E−01
7	871	3.14E−02	2.35E+00	780	3.26E−05	1.40E−01
8	727	3.14E−02	1.10E+02	709	3.26E−05	5.97E−02
9	188	3.14E−02	1.15E+02	657	3.26E−05	1.84E+00
10	961	3.14E−02	5.47E−01	335	3.26E−05	9.43E−01

error than classical GA for each run. For the 4th test function, almost 200 times less error is produced by the developed TbHP+ model with MAE = 3.14E−01 against MAE = 56.87 of classical GA model. Similar performance was also observed when TbHP+ is applied to the last test function. The computed mean MAE = 3.263E−05 for TbHP+ while mean MAE = 1.4289 for the classical GA. The ratio is almost 44 000.

As noted above, the mean of MAE of 10 runs for each test function is given in Table 3. Table 4 shows the number of iterations to reach this mean error by the TbHP+ model for each test function. For this purpose, we also did 10 different runs. The table, at the same time, shows the error produced by the classical GA for the same number of iterations. Note that, the classical GA, as presented in Table 3, is never able to reach the minimum error level captured by the TbHP+. For example, the mean of MAE of the 10 runs for the first test function is 96.7E−326 (see Table 3). Table 4 shows that it took only 143 iterations for the TbHP+ in the first run to reach that error. For the same number of iterations however the classical GA produced error of 1.883. Considering the results of 10 runs for the first test function, we can conclude that it took, on the average, 150 iterations for the TbHP+ to reach 97.7E−326 [practically zero] error level. For the same number of iterations, the classical GA, on the average, produced 1.34. This shows the superiority of TbHP+ which can produce practically zero error in a few iterations. Similar performance was observed for the other test functions (Table 4). According to Table 4, it took at most 457 iterations for TbHP+ to reach the minimum error of 1.54E−18 for the second test function. For the same number of iterations, the classical GA produced error of 2.158. For the third test function, it took minimum 221 and maximum 807

iterations for the TbHP+ to reach the error of 0.154. For the same number of iterations, the classical GA produced 1.02 and 0.211 mean absolute errors. As the functions became more complicated, as it was the case for the 4th and 5th test functions, it took more iteration [still less than 1000] for the TbHP+ to capture the low level errors. The classical GA for the same number of iterations produced significantly large errors.

Note that, the criteria of acceptance in the checkpoint of the TbHP+ can be adjusted in percent error to minimize the computation time or computation precision.

The reasons behind the better performance of this newly developed algorithm are manifold. With respect to time, the feature *instinct* is very efficient by slashing down the majority of the populations along the optimal path, or local optimal paths. This way, only a slice of the candidate solutions is retained as long as error is decreased progressively. Learning of the optimal new path resumes when the decrease in error stops, and an increase is observed. Even until ageing, overall population and the population of the traits are allowed to vary with respect to the feats of the traits. This is made possible through a very simple scheme of credit-assignment. After five successive wins of a certain trait, the individuals of other traits are culled randomly so as to reduce inactive population. This feature is called the immunity, since successive feats by a trait make this character immune to random culling.

5. Summary and conclusions

In the developed TbHP+ model, populations of homogenous individuals are transformed into populations of heterogeneous individuals. Non-similar individuals introduce their respective traits in the competition. The set of winner parents is replaced by the fittest individual who is made parent to yield offspring. Another concept that is implemented in the model is called the spontaneous ageing which is triggered by the solution neighborhood. As long as greedy is the dominant character, no ageing takes place. However, once the curious has emerged as the dominant character successively, i.e. solution neighborhood lies within the search radii, ageing commence. Simulations have shown that ageing helps zooming in on the solution with an ever increasing precision.

The model testing results showed that the developed TbHP+ model significantly outperforms the classical GA model. It produced minimum errors in small number of iterations [fewer than 200 for a simple test function and fewer than 1000 for very complex test function] in reaching the global minimums. These results imply that the TbHP+ model can be employed in a variety of disciplines in solving very complex problems.

Creating characters of various radii is the strongest part of this code compared to the classical GA. The existence of traits such as *fickle* and *erratic* almost always guarantees that this algorithm never gets trapped in local minima. Keeping these two traits well and alive even towards the final few cycles of the whole program reflects the strategy of this algorithm. While a fine-tuning is in progress in it's final cycles, a few sceptics keep searching for better solutions. This advantage, when combined with the adaptive nature of the code, gives the upper hand over GAs.

References

[1] T. Munakata, Fundamentals of the New Artificial Intelligence Beyond Traditional Paradigms, Springer-Verlag, New York, 1988.
[2] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, New York, 1992.
[3] K.D. De Jong, An analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor., 1975.
[4] S. Ozdemir, M. Karakurt, A combined simulated annealing-genetic algorithm optimization variant for networks, in: Proceeding of MDP-8, Cairo University Conference on Mechanical Design and Production, Cairo, Egypt, 2004.
[5] S. Alpay, L. Bilir, S. Ozdemir, Study of heterogeneous individuals, in: Proceedings of 4th International Symposium on Intelligent Manufacturing Systems, September 6–8, Sakarya, Turkey, 2004, pp. 767–773.
[6] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.
[7] Z. Khan, B. Prasad, T. Singh, Machining condition optimization by genetic algorithms and simulated annealing, Comput. Oper. Res. 24 (7) (1997) 647–657.
[8] H.P. Schewefel, Numerical Optimization of Computer Models, Wiley & Sons, Chichester, 1981.
[9] T. Ozel, Development of a predictive machining simulator for orthogonal metal cutting process, in: 4th International Conference on Engineering Design and Automation, July 30–August 2, Orlando, Florida, USA, 2000.
[10] L.A. Rastrigin, Extremal Control Systems, in: Theorethical Foundations of Engineering Series, Nauka, Moscow, 1974 (in Russian).
[11] A.A. Goldstein, I.F. Price, On descent from local minima, Math. Comput. 25 (115) (1971).
[12] Palisade Corporation, Evolver, the Genetic Algorithm Solver for Microsoft Excel, Newfield, New York, USA, 2001.