Recurrent Genetic Algorithms: Sustaining Evolvability

Adnan Fakeih[1] and Ahmed Kattan[2]

[1] Future Business Development Ltd., Saudi Arabia
[2] Department of Computer Science, Um Al-Qura University, Saudi Arabia
adnan@fbd.net, ajkattan@uqu.edu.sa

Abstract. This paper proposes a new paradigm, referred to as *Recurrent Genetic Algorithms (RGA)*, to sustain Genetic Algorithm (GA) evolvability and effectively improves its ability to find superior solutions. RGA attempts to continually recover evolvability loss caused by the canonical GA iteration process. It borrows the term Recurrent from the taxonomy of Neural Networks (NN), in which a Recurrent NN (RNN) is a special type of network that uses a feedback loop, usually to account for temporal information embedded in the sequence of data points presented to the network. Unlike RNN, the temporal dimension in our algorithm pertains to the sequential nature of the evolution process itself; and not to the data sampled from the problem solution space. Empirical evidence shows that the new algorithm better preserves the population's diversity, higher number of constructive crossovers and mutations. Furthermore, evidence shows that the RGA outperforms the standard GA on two NP problems and does the same on three continuous optimisation problems when aided by problem encoding information.

1 Introduction

The notion of "evolvability" is defined as *"the ability of a population to produce variants fitter than any yet existing"* [1]. Hence, the choice of selection, search operator and representation is vital to the performance of GA because they control the creation of new individuals throughout the evolutionary process. One aim of researchers in the Evolutionary Computation (EC) field is to discover new methods for increasing evolvability of evolutionary systems. The term evolvability does not only refer to how often offspring are fitter than their parents but also to the entire distribution of fitness values among offspring produced by a group of parents [1]. It should noted that even a random search can generate offspring that are fitter than their parents. Thus, to prove that a GA's performance is superior we need to show that the fitness distribution of the entire population is higher than that produced by a random search process. Obviously, for a successful evolvable search process not all parents in the population need to produce fitter offspring. It is usually those parents with higher than average fitness who carry the responsibility of making the search rewarding. This is because selection is naturally biased toward this slice of the population.

To increase evolvability of individuals means to implicitly encourage search operators to produce a high correlation between parents and offspring fitness values in the next generation. This correlation has been explained by building block hypothesis in [4] as the correlation between parents and offspring fitnesses under the crossover operator. The building block is a sequence of genetic materials in a fit parent that is likely to produce fitter offspring upon joining a crossover process with other individuals.

In the standard form of GA, the fundamental idea that moves the search process is gleaned from the famous Darwinian theory of the *"survival of the fittest"* in which individuals that have superior fitness value (in relation to the problem to be solved) are considered fitter than inferior individuals and thus have a better chance to pass their good genetic materials (or more precisely, potentially good genetic materials) into the next generation. In this work, we consider another way of looking at the term *"fittest"* in which we ascribe this description to those individuals who are able to produce fitter offspring. Naturally, these individuals may not necessarily be the fittest in relation to solving the given problem. To this end, we propose a modification to the canonical GA where we evaluate individuals based on their parental abilities (more on this in Section 3). Evolvability refers to the potentiality of evolvement; rather than immediate improvements in fitness. Therefore, our algorithm works best when allowed to evolve for significantly higher number of generations.

2 Related Works

The concept of evolvability has been an active research area in both evolutionary biology and computer science for the past several decades. Hu and Banzhaf in [6] have argued that adopting new knowledge about natural evolution generated in areas such as molecular genetics, cell biology, developmental biology, and evolutionary biology would benefit the field of evolutionary computation. The authors discussed evolvability and methods for accelerating artificial evolution by introducing notions from biology and their potential in designing new algorithms in EC.

It has been recorded that the evolvability property has good effect on the search process. For example, in [2], the authors suggested that evolvability can effectively reduce the bloat in evolutionary algorithms that use variable length representations. In their work, the authors noted the similarity of bloat causes and evolvability theory, thus, they argue that reproductive operators with high evolvability will be less likely to cause bloat.

With the importance of evolvability as a research topic, several measurements have been proposed to quantify it. Wang and Wineberg [11], suggested two measures of evolvability one based on fitness improvement and the other based on the amount of genotypic change. The authors divided the population into three sub-populations, where the size of each sub-population is determined dynamically. The first sub-population uses selection based on fitness directly; the second sub-population is based on the fitness-improvement-ratio; finally, selection for the third sub-population is based on genotypic change. Each sub-population is filled by selecting chromosomes from the parent's generation under its own selection functions. Thereafter, the three sub-populations are merged, and the GA genetic standard operators are applied to form the next generation.

Unlike other works, in this paper, we propose a new paradigm for the evolutionary process to sustain population's evolvability and effectively improves its ability to find superior solutions. The main idea is based on rewarding the parents the fitness of their offspring. This is implemented by introducing an intermediate population (a feedback loop) to measure the ability of parents to reproduce. Upon re-evaluating parents' fitnesses, the algorithm proceeds as a standard GA; until next evaluation is due. (more details in Section 3).

3 Recurrent Genetic Algorithms

The proposed paradigm is broadly outlined in figure1. Firstly, as in standard GA procedures, we randomly generate an initial population t_i where $i \in \{0, 1, \ldots$ $max\_generation\}$ and rank its individuals based on their fitness values. Using standard selection and genetic operators, the system generates population $\hat{t_i}$ from population t_i. Here, population $\hat{t_i}$ is used as feedback intermediate population (between population t_i and t_{i+1}) where it allows the system to discover the ability of individuals to produce fitter offspring. Once $\hat{t_i}$ is available, the algorithm uses its fitness values to reward parents in population t_i, thus, this intermediate population is used to evaluate individuals considering how much they effectively managed to push the search process forward, which, of course, may not coincide with the standard evaluation of individuals based on their fitness values.

Here $\hat{t_i}$ is used as a feedback loop as in the Recurrent Neural Networks (RNN) [5] where connections between units form a directed cycle used to allow it to exhibit dynamic temporal behaviour. Unlike feedforward Neural Networks, RNNs can use their internal memory to process arbitrary sequences of inputs. Although GA applications, in general, do not have temporal dimension as the basic focus lies in finding the best solution in a static "spacial" solution space, we use the the recurrencey concept in the intermediate population $\hat{t_i}$ to capture the temporal effects that the GA undergoes during the process of evolution. In other words, we use the recurrency notion to account for the evolvability of GA from one generation to the next.

Preliminary experiments show that an evaluation of individuals that is solely based on the fitness value of their offspring may not appropriately enhance the evolvability. This may be explained by several factors controlling the creation of any offspring; such as crossover/mutation point, original fitness value of the parent, selection pressure and the fitness value of the other parent in case of crossover. Therefore, if we simply allocate offspring fitness to parents, we would be neglecting all these important factors that contribute to creating the offspring. Also, this raises the question of which offspring to use when rewarding parents?. In [11] the authors used fittest offspring (i.e., the one with the highest fitness value) to measure the evolvability of an individual (assuming that it gives an

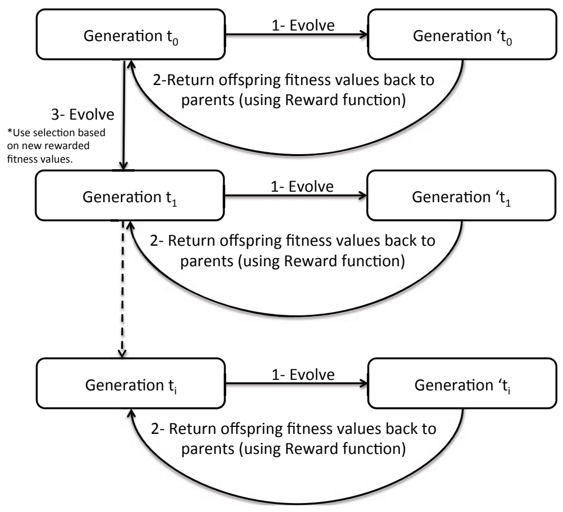
Generation t_0
1- Evolve
Generation 't_0
2-Return offspring fitness values back to parents (using Reward function)
3- Evolve
*Use selection based on new rewarded fitness values.
Generation t_1
1- Evolve
Generation 't_1
2- Return offspring fitness values back to parents (using Reward function)
Generation t_i
1- Evolve
Generation 't_i
2- Return offspring fitness values back to parents (using Reward function)

Fig. 1. RGA process outline

indication of the potential fitness upper limit that an individual can produce). This is not, however, entirely accurate estimation because the circumstances that resulted in creating this offspring are not necessarily to be repeated in future generations. For these reason, in this research, we opted for rewarding the parents based on the average fitness values of their offspring and the amount of genetic materials they passed onto their offspring. Here, we used a *Fitness Reward Function* (FRF) that rewards parents in population t_i based on their offspring's fitness values in the intermediate population $\hat{t_i}$.

For each crossover operator that parents P_x and P_y joined, where $x, y \in \{1, 2, \ldots, population\_size\}$, we use the following FRF:

$$FRF(Parent_x Fitness) = Offspring\_Fitness \times Parent_x\_contribution$$

$$FRF(Parent_y Fitness) = Offspring\_Fitness \times Parent_y\_contribution \quad (1)$$

where, *Offspring_Fitness* is the fitness value of the generated offspring, $Parent$ $_x\_contribution$ and $Parent_y\_contribution$ are real numbers from the interval $(0, 1)$ to represent the proportion of genetic materials that each parent contributed when generating the offspring. Note that $Parent_x\_contribution$ + $Parent_y\_contribution = 1$.

For each mutation operator that parent P_x joined, we use the following FRF:

$$FRF(Parent_x Fitness) = (Offspring\_Fitness \times Parent\_contribution_x) \quad (2)$$

Here, because the mutation operator is based on single parent, $Parent_x\_$ *contribution* is calculated as the amount of genetic materials that passed from the parent into the offspring.

The final rewarded fitness value is calculated as follows:

$$Parent_x\_Fitness = \frac{\sum FRF(Parent_x\_Fitenss)}{Number\_of\_offspring} \tag{3}$$

where, $\sum FRF(Parent_x\_Fitenss)$ is the summation of the FRFs for $Parent_x$ as shown in Equations 1 & 2 (whether it joined crossover and/or mutation operators) and *Number_of_offspring* is the total number of offspring produced by $Parent_x$ in population $\hat{t_i}$.

Note that once the RGA re-evaluates population t_i, it does not consider the original fitness of the parent any longer. Instead, all individuals are ranked based on their ability to produce offspring in relation to the amount of genetic materials they passed onto those offspring.

Naturally, the selection process may decides to leave some individuals unselected. Usually, those individuals have the most inferior fitness values of the whole population and therefore the selection process decides that they are not worthy to be allowed to be part of the next generation. We experimented with several alternatives as to how to evaluate parents that have not produced any offspring. One was to assign them the mean fitness value of the whole population, which resulted in poor performance. The best empirically based treatment was found to be allocating such parents "the least" fitness value allocated to individuals in the same population.

Despite the success of RGA (as will be shown in the experiments section), it suffers from an obvious disadvantage which is the extra computational cost introduced by producing and evaluating the intermediate population. Thus, it is fair to say that RGA has a slower convergence rate than standard GA. However, when comparing the performance of the two algorithms to each others, this disadvantage is mitigated by allowing RGA to evolve for only half the number of generations iterated by the standard GA.

3.1 Elitism

As explained previously, RGA uses FRF to reward parents based on the average fitness values of their offspring; and the amount of genetic materials they passed onto their offspring. Thus, for two parents who joined a crossover operator, the evaluation of their fitness values is dependant on the amount of chromosomes they passed to their offspring. This is both a strength and a weakness, though. On one hand, it is a strength in that the parent who passed more chromosomes into its offspring is more likely to pass a golden building-block of chromosomes that contributes in creating a fitter offspring therefore it receives bigger reward than the second parent. On the other hand, it is a weakness because if this golden building-block of chromosomes is a mixture of both parents (e.g., tail of the first parent concatenated with the head of the second parent) then our reward mechanism will not be fair. This unfairness of reward may divert the algorithm from pursing optimality by discarding already highly fit solutions. There is no practical way of knowing this information unless we have an explicit knowledge about the problem. Therefore, since RGA has already invested in evaluating population $\hat{t_i}$ (the intermediate stage to measure the evolvability of population t_i) we copy the elite individuals from $\hat{t_i}$ into t_i. This has proven to improve the performance in some problems.

4 Experiments

The experiments have been designed to see whether RGA can sustain evolvability, and to see how diversity is closely related its behaviour. Our experiments covered three different problems, namely, NK-Landscape [8] *(unimodal problem)*, Hamming Centres [3], *(multimodal problem)*, and three different continuous optimisation problems.

4.1 NK Landscape

NK-Landscape was established by Stuart Kauffman in [8]. We investigated the performance of RGA under different values of N. Namely, we used $N = 20, 30, 40$ and 50. For each N value we tested three different K values. Thus, $K = \frac{N}{5}$ *(easy problem)*, $\frac{N}{2}$ *(hard problem)*, and $\frac{N}{2} + 5$ *(very hard problem)*.[1] For each N, K combination we tested the system using 20 independent runs. Results have been compared to the standard GA. As stated earlier, to allow fair comparison, we used exactly the same number of evaluations in both systems. Thus, we counted the number of evaluations in the intermediate generations and gave exactly the same to the standard GA. For both RGA and standard GA we used population of 100 individuals and evolved them through 500 generation (this includes the intermediate generations in RGA), crossover rate was 0.9 and mutation 0.1, tournament selection was of size 2. In RGA we applied 5% elitism (defined in sec. 3.1) and 5% standard elitism for the standard GA.

Table 1 summarises the results of 480 independent runs. As can be seen in the table, when the NK problems are easy (as in $NK(20, 5)$, $NK(30, 6)$, $NK(40, 8)$ and $NK(50, 10)$) both RGA and standard GA are even on average (where each system has better average in two out of four cases). Also, in these four easy problems the best solutions (across the 20 runs) achieved by standard GA are better than those achieved by RGA. This is because the problem's landscape is relatively smooth so standard GA search had a good chance to hit solutions near the global optima. Now, if we look at the hard and very hard problems, it is clear that RGA comes on the first place both in terms of average (i.e., average of best solutions in the 20 runs) and best (i.e., best achieved solution across the entire 20 runs). Thanks to the feedback loop, introduced through the intermediate populations RGA has higher evolvability than standard GA search.

To further compare the behaviour of RGA against its competitor, we measured the average of four different criteria for each system under each N, K

[1] In [7] the authors provided an indication of NK-landscape hardness under different settings.

Table 1. Summary of 480 independent runs (20 runs for each N,K combination with each system).

	RGA			GA		
	Mean	*Best*	*Std*	*Mean*	*Best*	*Std*
N20						
K=5	**0.76**	0.77	0.01	0.75	0.77	0.02
K=10	**0.76**	**0.78**	0.02	0.74	0.77	0.02
K=15	**0.75**	**0.79**	0.02	0.74	0.77	0.02
N30						
K=6	0.76	0.80	0.03	**0.78**	0.80	0.02
K=15	**0.72**	**0.77**	0.02	0.71	0.74	0.01
K=20	**0.71**	**0.74**	0.01	0.70	0.73	0.02
N40						
K=8	0.74	0.78	0.02	0.74	**0.79**	0.02
K=20	**0.70**	**0.72**	0.01	0.69	0.70	0.01
K=25	**0.69**	**0.71**	0.01	0.68	0.69	0.01
N50						
K=10	0.72	0.75	0.02	0.72	0.75	0.01
K=25	**0.69**	**0.74**	0.02	0.67	0.69	0.01
K=30	**0.67**	**0.72**	0.02	0.67	0.70	0.01

***Bold** numbers are the highest.

combination across the 20 runs. As can be seen in Figure 3, we compared first the average of best solution (generation by generation) for each system. Note that when the problems are easy both RGA and standard GA have almost the same performance. However, as the problem gets harder RGA gets better as its performance goes beyond the standard GA. What is impressive about these fitness curves is that they maintain an almost linear fitness increase for a prolonged period, and do so in a gradualistic manner, whilst standard GA reaches a plateau and no further improvement in the fitness is observed. Secondly, we compared the diversity of population (diversity was measured as the entropy of the population's fitness). RGA remarkably has higher diversity than standard GA in all experiments. It is worth noticing, though, that the diversity becomes higher as the problem gets harder. Finally, we compared the number of constructive crossovers, and mutations, by 'constructive' we refer to the crossover or mutation operator that resulted in a fitter offspring than its parents. It is clear that RGA has significantly a higher number of constructive crossovers and mutations than its competitor in all runs.

To compare the parents-offspring fitnesses across all generations we used the *fitness cloud* graph introduced by Vanneschi *et al.* in [10] to get a visual rendering of evolvability. Figure 2 illustrates the parent-offspring fitnesses in one of the runs versus their number of occurrence in all generations. As can be seen in standard GA fit parents are not able of producing fitter offspring all times. This

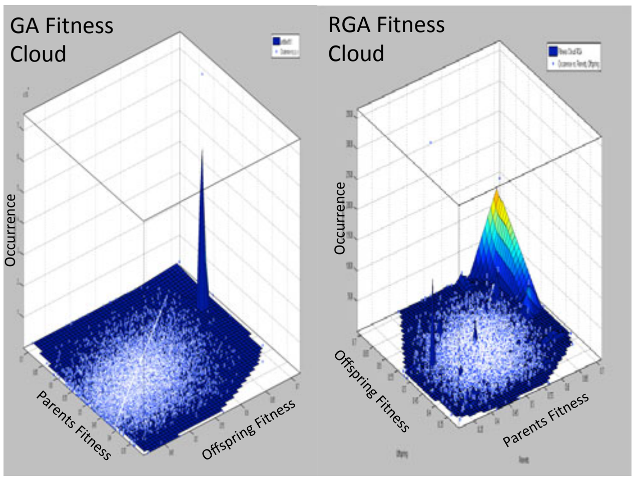

Fig. 2. Fitness Cloud: RGA vs. Standard GA NK(30,15)

is probably due to the destructive nature of the search operators (as it has been illustrated in figure 3 by the declining number of successful (i.e., constructive) crossover/mutations operators). We also noticed that in standard GA the search converges to a single solution dominating the whole population. This is clearly illustrated by the peak appearing in the figure, where a single parents-offspring fitness has high dominating number of occurrences. However, RGA shows that fit parents are able of producing fitter offspring most of the times. Also, the search does not allow a single individual to dominate the whole population as in standard GA, due to the diversity sustained by the RGA.

4.2 Hamming Centres

Hamming Centers is an NP-complete problem defined in [3] as follows. Lets a set S of k_i binary strings, where $i \in \{1, 2...I\}$, each of length n, and r is a positive integer. The objective is to find $n-$bits string y such that for every string k_i in S, the Hamming distance, $H(k_i, y) \leq r$.

We investigated the performance of RGA under different values of $n-$bits ($n =$ 20, 40, 60, 80, 100 and 120). For each n value we performed 20 independent runs. The size of the set S was 20 in all experiments. The fitness value was measured as the number of cases that string y satisfied the condition of $H(k_i, y) \leq r$. Thus, the optimal solution is equal to the size of S (which is 20 in our case). We used the same setting as in Section 4.1, except that the population's size was 500 and the number of generations was set to 1000 for each system.

Table 2 summarises the results of 240 independent runs. As can be seen in the table, RGA performance improves as the problem gets harder. To further compare the behaviour of RGA against its competitor, we measured the average of the four criteria (similar to the NK experiments in Section 4.1) for each system

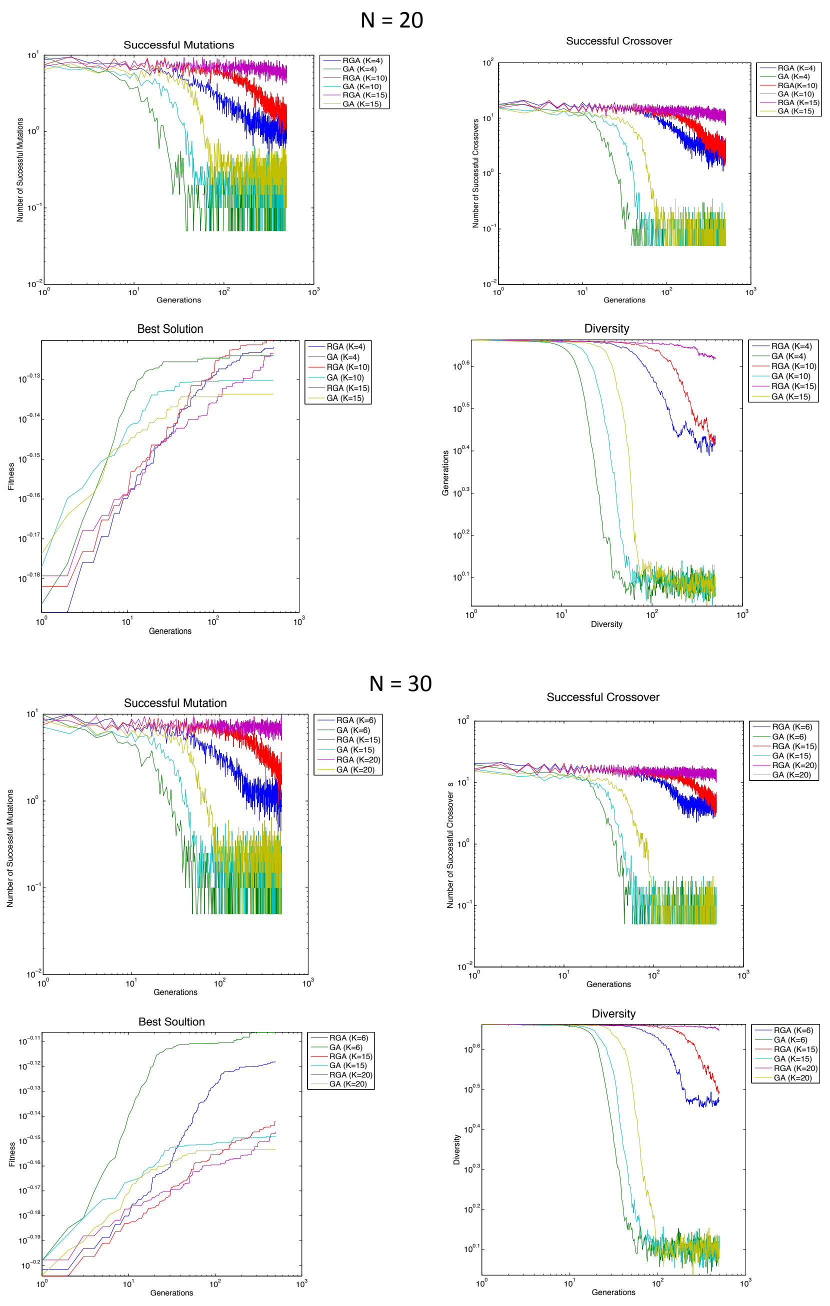

Fig. 3. RGA vs. Standard GA on NK landscape (N = 20, 30)

Table 2. Summary of 240 independent runs (20 runs for each n value)

	RGA			GA		
	Mean	*Best*	*Std*	*Mean*	*Best*	*Std*
n=20	**3**	3	0.00	2.75	3	0.43
n=40	**7.1**	8	0.71	6.95	8	0.70
n=60	**9.75**	**11**	0.73	8.7	10	0.69
n=80	**11.45**	**13**	1.00	10.9	12	0.84
n=100	**12.65**	**15**	1.20	12.1	14	0.90
n=120	12.6	**14**	0.94	**12.65**	13	0.56

***Bold** numbers are the highest.

under each n value across the 20 runs.[2] Looking at the average best solutions in each generation, we noticed that RGA behaves the same way it did in the NK problem. For $n = 20$ and 40 RGA and GA have almost similar performance. The performance gap increases in favour of the RGA in the remaining n values. Also, RGA has a higher diversity and higher number of constructive operators in all runs. Moreover, we noticed that these measures show better values as the problem gets harder. Despite the remarkable results obtained by the RGA it is fair to note that the difference in average best solutions in each generation between it and the standard GA is not as large in this problem as it was in the NK experiments, which may indicate that the RGA does not perform as well in multimodal problems as it does in unimodal ones.

4.3 Continuous Optimisation

We investigated RGA's performance in continuous optimisation problems. Three benchmark functions have been used in our experiments. Namely, Rastrigin function, Dixson & Price function, and Michalewics function. Functions' notations are defined in [9].

Here, we used the same setting as in 4.1, except that the number of generations was set to 5000 and population size was set to 500 for each system. Unlike the other two problems (i.e., NK-landscape and Hamming Centres), here individuals are coded as real numbers from the interval $(0, 5]$. Thus, each function receives n number of parameters and the RGA tries to optimise these parameters in such a way that maximises the function's output. In our experimentation we tried to maximise the test functions using RGA and GA under different number of parameters. Thus, we explored the systems performance at $n = 15$, 20, 25, and 30 for each function. For each n value we tested the systems using 20 independent runs. Table 3 summarises the results. As can be seen in the table, RGA has been outperformed by standard GA in most of the runs. We also noticed that averages of best solutions in each generation, diversity and the number of constructive crossovers/mutations have dropped drastically similar to the standard

[2] Due to the restriction on number of pages we are not able to present the full figures in this paper.

Table 3. Summary of 480 independent runs (20 runs for each n value) using original FRF

	RGA			GA		
	Mean	*Best*	*Std*	*Mean*	*Best*	*Std*
DixonPrice function						
$n = 15$	8227.37	8704.30	303.86	**8524.99**	**9078.43**	408.09
$n = 20$	14157.76	15211.50	591.62	**14954.73**	**16054.00**	779.36
$n = 25$	21715.74	23024.50	1051.07	**22759.18**	**25005.10**	1125.48
$n = 30$	30109.90	34750.80	2117.33	**31761.24**	**35665.00**	1835.55
Michalewics function						
$n = 15$	8.94	10.27	0.75	**9.30**	**10.87**	0.85
$n = 20$	10.57	12.95	1.08	**11.86**	**14.01**	1.16
$n = 25$	12.28	14.10	1.06	**13.77**	**16.22**	1.43
$n = 30$	13.97	16.30	1.19	**15.09**	**18.46**	1.72
Rrastrigin function						
$n = 15$	543.68	548.53	2.88	**545.95**	**554.09**	4.53
$n = 20$	725.72	732.41	3.34	**729.53**	**743.96**	6.28
$n = 25$	905.09	**916.32**	5.21	**907.34**	915.27	5.16
$n = 30$	1087.16	1096.39	4.80	**1090.50**	**1106.41**	7.84

***Bold** numbers are the highest.

GA (unlike the other two problems). However, RGA still maintaining slightly better diversity. These results were surprising given the superior performance by RGA in the previous two problems. We believe that the reason for this performance is largely attributed to the selection of the FRF, which needs to be defined differently for this type of problems. The standard FRF did not manage to reward parents in a favourable manner.

In an attempt to verify our assumption (i.e., degradation of performance in this problem is largely attributed to the selection of the FRF) we have introduced a slight modification in the FRF used for solving continuous optimisation problems. The FRF (defined in Section 3) assumes a linear dependency between impact on fitness and quantity of genetic material passed to the offspring, thus the fitness of the parent has been estimated as the weighted average of the fitness of the offspring with weights the chromosomes proportions of the offspring inherited from the parent. This assumption coincides with the building block hypothesis [4]. In other words, beneficial properties of parents are aggregated in (relatively) small code blocks at various locations within the genome.

This, however, and unlike the previous two test problems, does not work well in continuous optimisation problems where chromosomes are real numbers and not binary digits that can be summed up to represent the parents contribution to the formation of a specific offspring. To account for the different nature of the problem under investigation, we modified the FRF to consider the values contained in the chromosomes contributed by the parents rather than the mere number of chromosomes. So, the contribution of $Parent_x$ now is the sum of the values contained in the chromosomes contributed into the offspring. This

Table 4. Summary of 240 independent runs (20 runs for each n value) using modified FRF

	RGA		
	Mean	*Best*	*Std*
DixonPrice function			
n = *15*	**8944.85**	**9509.38**	272.66
n = *20*	**15645.78**	**16321.30**	475.21
n = *25*	**23859.73**	**25401.10**	975.31
n = *30*	**34229.60**	**36340.1**	1184.77
Michalewics function			
n = *15*	**12.13**	**12.73**	0.44
n = *20*	**16.29**	**17.42**	0.65
n = *25*	**17.67**	**19.22**	0.93
n = *30*	**20.19**	**23.16**	1.39
Rrastrigin function			
n = *15*	**552.36**	**569.30**	6.51
n = *20*	**732.04**	**742.12**	5.12
n = *25*	**910.80**	**925.45**	5.16
n = *30*	**1090.55**	1100.94	5.00

***Bold** numbers are the higher than standard GA.

modification did the trick, and the performance of the RGA was again superior to that of the GA in all test problems (see Table 4).

This modification confirmed our assumption, that the low performance in Table 3 was indeed due to the unsuitability of the FRF. Off course it can be argued that this enhancement is obtained by using additional knowledge about the problem under consideration. This is absolutely true, but it should also be remembered that the issue of "parent contribution" is in the heart of the RGA algorithm. In contrary to standard GA, this allows RGA to benefit from such problem "encoding" knowledge in a very simple and straightforward manner. So, although it is not fair to compare the performance of RGA to that of GA when the former benefits form problem encoding information, while the later does not; it is also unfair to deprive the former form so doing just because the later does not have the means to employ such useful information. In future research we will concentrate on this aspect of the algorithm.

5 Conclusions

This paper proposes a new paradigm, referred to as *Recurrent Genetic Algorithms (RGA)*, that attempts to sustain Genetic Algorithm (GA) evolvability and effectively improves its ability to find superior solutions. The main idea is formalised by simply introducing an intermediate population between subsequent generations. This intermediate population serves as a feedback loop where the system reward individuals based on their abilities to produce better offspring. The idea of the feedback loop accounts for the temporal effects that the GA undergoes during the process of evolution.

As an experimental validation of the new paradigm on a non-trivial space and structured representation, we have considered two well-known NP benchmark problems: the NK-landscape problem, to show the RGA behaviour under unimodal problems and the Hamming Centres, to show RGA behaviour under mutlimodal problems. Moreover, we tested RGA using three continuous optimisation problems. Experimental evidence shows that RGA remarkably maintains higher diversity and increase the population's ability to produce fitter offspring in comparison to standard GA. Furthermore, empirical evidence shows that the new paradigm has outperformed standard GA on two NP problems and does the same on three continuous optimisation problems when aided by problem encoding information. This, indeed, shows that RGA has the potential to work well on real-world problems.

In future work we will test RGA on multi-objective optimisation problems.

References

1. Altenberg, L.: The evolution of evolvability in genetic programming. In: Kinnear Jr., K.E. (ed.) Advances in Genetic Programming, ch.3, pp. 47–74. MIT Press (1994)
2. Bassett, J.K., Coletti, M., De Jong, K.A.: The relationship between evolvability and bloat. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO 2009, pp. 1899–1900. ACM, New York (2009)
3. Frances, M., Litman, A.: On covering problems of codes. Theory of Computing Systems 30, 113–119 (1997), doi:10.1007/BF02679443
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1989)
5. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall, Upper Saddle River (1999)
6. Hu, T., Banzhaf, W.: Evolvability and speed of evolutionary algorithms in light of recent developments in biology. J. Artif. Evol. App. 2010, 1:1–1:28 (2010)
7. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Eshelman, L.J. (ed.) ICGA, pp. 184–192. Morgan Kaufmann (1995)
8. Kauffman, S., Levin, S.: Towards a general theory of adaptive walks on rugged landscapes. J. Theoret. Biol. 128(1), 11–45 (1987)
9. Molga, M., Smutnick, C.: Test functions for optimization needs. Test functions for optimization needs (2005)
10. Vanneschi, L., Clergue, M., Collard, P., Tomassini, M., Vérel, S.: Fitness Clouds and Problem Hardness in Genetic Programming. In: Deb, K., et al. (eds.) GECCO 2004, Part II. LNCS, vol. 3103, pp. 690–701. Springer, Heidelberg (2004)
11. Wang, Y., Wineberg, M.: The estimation of evolvability genetic algorithm. In: The 2005 IEEE Congress on Evolutionary Computation, vol. 3, pp. 2302–2309 (September 2005)