Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

Analysis of frequency-based compact genetic algorithm (fb-cGA)

Sunisa Rimcharoen, Srichol Phiromlap and Nutthanon Leelathakul *

Faculty of Informatics, Burapha University, Chon Buri, 20131, Thailand

* Corresponding author, e-mail: nutthanon@buu.ac.th

Received: 19 June 2014 / Accepted: 30 March 2015 / Published: 9 April 2015

Abstract: A behaviour analysis of frequency-based compact genetic algorithm (fb-cGA) is proposed. The fb-cGA is a version of compact genetic algorithm (cGA) enhanced by the use of a new updating strategy. The algorithm counts the number of probability updates and the continuities of probability-update directions and uses them to adaptively update the algorithm's step sizes. This method requires fewer function evaluations and achieves solutions that are more accurate than those from the conventional cGA. It has been shown that fb-cGA can reduce the number of function evaluations to only one ninth of the number obtained from cGA on ten copies of a 3-bit trap function using a tournament size of 2. We conduct parameter studies and show that the use of one fourth of the population size (*psize/4*) as the algorithm's starting threshold can improve the overall efficiency of fb-cGA. The behaviour of fb-cGA on various problems is also examined. The results of the analysis show that information from the algorithm's past experience (i.e. the numbers of probability updates and continuities) can help the fb-cGA to update the probability vector towards a more promising direction, requiring fewer function evaluations.

Keywords: compact genetic algorithm, updating strategy, update frequency, update continuity

INTRODUCTION

Compact genetic algorithm (cGA) was proposed by Harik et al. [1]. It has been widely applied to various fields such as pipe network optimisation [2], parameter optimisation [3, 4], inventory planning [5], image recognition [6], traffic transportation management [7], communication [8-10], container loading [11], grid computing [12] and biology [13, 14]. The main contribution of this algorithm is to replace a whole set of candidate solutions (the so-called population) used by simple genetic algorithm (sGA) with a probability distribution. cGA requires

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

much less memory, as it does not need to maintain the population throughout the evolution process. The concept of cGA can be easily translated to hardware implementation by using the common very-large-scale integration [15-17]. Therefore, it opens up the application of genetic algorithm to new fields such as embedded systems. For example, Timmerman [18] used cGA to develop an insect-sized flapping-wing micro air vehicle.

However, for more difficult problems, cGA does not provide acceptable solutions. There have been many attempts to modify and improve cGA's probability updating strategy. Zhou et al. [19] proposed an improved cGA using mutation and named the algorithm mutated-by-bit-compact genetic algorithm (MBBCGA). At each generation, MBBCGA generates only one individual and then mutates this individual bit by bit. Ha et al. [20] proposed the use of more than one probability vector (PV) to enhance the exploration properties of the algorithm. Rimcharoen et al. [21] improved the updating strategy of cGA by using a moving average technique (mcGA). Ahn and Ramakrishna [22] adopted 'elitism', i.e. the idea of reserving the best solution in each generation. They proposed two variants: a persistent elitist compact genetic algorithm (pe-cGA) and a non-persistent elitist compact genetic algorithm (ne-cGA). The former stores the current best solution until a better solution is found, while the latter keeps the best solution just for a certain lifetime. In 2008 Lee et al. [23] introduced a new update strategy using augmented Bayesian networks. A few years later, they proposed compact genetic algorithm using a belief vector (cGABV) [24]. The new technique uses a belief vector (BV) instead of a probability vector. The difference between BV and PV is that each element of the BV stores a probability distribution (represented by associated mean and variance), whereas each of the PV keeps a probability value.

In our previous work [25], we proposed the usage of a frequency-based updating technique as the updating strategy of cGA. The technique collects and utilises information from the algorithm's past experience. Specifically, for each probability in the PV, the number of probability updates (in both up and down directions) are counted and used to adjust probability-updating step sizes, turning the vector towards the promising direction faster. Comparison results show that the frequency-based compact genetic algorithm (fb-cGA) requires substantially (up to nine times) fewer function evaluations when compared with traditional cGA. However, in-depth explanation and analysis of why this algorithm outperforms others remained lacking. Accordingly, in this paper, we conduct parameter studies and analyse how the algorithm behaves while solving various problems.

FREQUENCY-BASED COMPACT GENETIC ALGORITHM (fb-cGA)

cGA is one of various evolutionary algorithms. Instead of evolving the population for searching solutions, it employs a probabilistic model, PV, which requires relatively small amount of memory. Furthermore, the algorithm eliminates genetic operators such as crossover and mutation.

cGA keeps a PV over a chromosome to represent the population. The number of probabilities in the vector is equal to the chromosome length. Each probability is defined as the probability with the associated bit being equal to 1. The pseudo-code of cGA is shown in Figure 1. The two parameters are the chromosome length (l) and the population size (*psize*), which are used to further specify an updating step size (i.e. step size defined as 1 / *psize*). (Note that the relation between *psize* and the updating size in the cGA is analogous to the one between population size and evolving speed in sGA.)

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

```
initialise(p)
while (p does not converge) do
    individual1 := generate(p)
    individual2 := generate(p)
    evaluate(individual1, individual2)
    winner, loser := compete(individual1,
                             individual2)
    for i:=1 to l
    begin
        if winner[i] ≠ loser[i] then
            if winner[i] = 1 then
                p[i] := p[i] + 1/psize
            else
                p[i] := p[i] – 1/psize
    endfor
endwhile
```

Figure 1. Pseudo-code of cGA

```
s := tournament size
initialise(p)
while (p does not converge) do
    create(p, S[], s)
    evaluate(S[])
    rearrange(S[]) // S[1] is the best individual
    for i := 2 to s
    begin
        winner, loser := compete(S[1], S[i])
        for i:=1 to l
        begin
            if winner[i] ≠ loser[i] then
                if winner[i] = 1 then
                    p[i] := p[i] + 1/psize
                else
                    p[i] := p[i] – 1/psize
        endfor
    endfor
endwhile
```

Figure 2. Pseudo-code of tournament cGA

First, the cGA initially sets each of the probabilities in the vector to 0.5. According to the PV, the algorithm randomly generates two candidate solutions, denoted as *individual1* and *individual2*. Next, the solutions are evaluated, i.e. assigned fitness values. The winner, the one with the greater fitness value, is selected. In step 5, the PV is then updated towards the winner. The value of each probability changes if the winner's associated bit is not equal to the loser's: either increasing when the winner's bit is one, or decreasing otherwise. The loop continues to run until the PV converges, meaning that each probability in the vector is either zero or one.

Harik et al. [1] also modified cGA by adding more candidates, called tournament cGA, shown in Figure 2. The modified version randomly generates a set of s candidate solutions, denoted by an array *S* in the pseudo-code, and uses a tournament selection to choose the winner, which will be stored in *S*[1]. The PV is then updated by comparing *S*[1] with *S*[i] (for all *i* not equal to 1) in the same manner as the original cGA.

Both of the cGAs update each probability in the vector towards either one or zero. Some probabilities gradually increase while others drop. However, some might fluctuate, reflecting uncertainty in updating the PV. It is known that the PV fluctuates during the beginning period and converges to a certain direction at the end. The algorithms seem to work well in the case where problems have consistent information, leading the algorithms to turn the vector towards only one direction. However, if the problems are deceptive, they might delude the algorithms into searching for solutions in the wrong directions. Consequently, the cGAs could not provide the desired solution quality in spite of spending much of searching time. There has been much research aimed at modifying and improving the cGAs in such case.

In our previous work [25], we applied a frequency-based technique to update the PV, using the numbers of updates and the continuities of preceding updates as criteria. (The update continuity is

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

defined as the number of consecutive updates moving towards the same direction). We measured the uncertainty by observing the direction of each probability in the vector: if the direction is the same for a long time (high continuity), the uncertainty is low. The monitored continuities serve as a guideline or a promising trend that quickly leads to vector convergence.

Specifically, for each probability in the vector, the frequencies of two types of updates: stepping-up (increasing the probability towards 1) and stepping-down (decreasing the probability towards 0), were counted. Likewise, two types of update continuities were collected. The stepping-up continuity is reset to zero if the current update moves towards 0 and the stepping-down continuity is reset to zero if the current update moves towards 1. The fb-cGA technique is shown in Figure 3.

Updating strategy of fb-cGA

```
1:   for i := 1 to l
2:   begin
3:       if winner[i] ≠ loser[i] then
4:           if winner[i] = 1 then
5:               Ufreq[i] := Ufreq[i] + 1;
6:               Ucon[i] := Ucon[i] + 1;
7:               Dcon [i] := 0;
8:               if (Ufreq[i] > Dfreq[i] AND Gen > (psize/3)) then
9:                   p[i] := p[i] + ((1/psize)+(p[i] * (Ucon[i]/100));
10:              else
11:                  p[i] := p[i] + (1/ psize);
12:          else
13:              Dfreq[i] := Dfreq[i] + 1;
14:              Dcon [i] := Dcon [i] + 1;
15:              Ucon[i] := 0;
16:              if (Dfreq[i] > Ufreq[i] AND Gen > (psize/3)) then
17:                  p[i] := p[i] - ((1/psize) + (p[i]*(Dcon [i] / 100));
18:              else
19:                  p[i] := p[i] - (1/psize);
20:  endfor
```

Parameters:

Ufreq : number of stepping-up updates
Dfreq : number of stepping-down updates
Ucon : number of consecutive stepping-up updates
Dcon : number of consecutive stepping-down updates
Gen : generation number (incremented in Step 6)

Figure 3. Pseudo-code of fb-cGA

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

Figure 3 presents the pseudo-code of the frequency-based updating strategy in fb-cGA. *Ufreq* denotes the number of probability updates towards one (i.e. stepping-up updates). *Dfreq* denotes the number of probability updates towards zero (i.e. stepping-down updates). *Gen* denotes the generation number whose value is increased incrementally in step 6. The proposed updating strategy is performed when *Gen* is greater than 1/3 of the population size (*psize*). For the first third of the generations, fb-cGA works like the original method to explore solutions and find the right direction. It waits until the generation number reaches *psize*/3 because it needs time to gather sufficient information to see the trend. For the last two-thirds of the generations, the i^{th} probability is updated when the i^{th} bit of the winner (*winner*[*i*]) and the one of the loser (*loser*[*i*]) are not equal. If *winner*[*i*] is 1, the algorithm checks whether, from past experience, this probability is updated towards 1 most of the time (i.e. *Ufreq* greater than *Dfreq*). If so, the probability vector should be updated according to the majority with a larger step size. The step size can be determined by adding the term *Ucon*/100 multiplied by the previous value of i^{th} probability, where *Ucon* denotes the number of consecutive stepping-up updates. In contrast, when *winner*[*i*] is 0, the algorithm performs in a similar manner but considers *Dfreq* and *Dcon* instead. The i^{th} probability is updated by decreasing towards zero.

In this paper, we study the effects of the *psize* parameter and show that using *psize*/4 can improve the efficiency of fb-cGA. Thus, we use *psize*/4 instead of the previously proposed *psize*/3 [25] throughout the experiments conducted and presented in this paper.

PARAMETER STUDIES

As mentioned earlier, the proposed method performs the new updating strategy when the number of generations is greater than *psize*/4. The reason behind this strategy is that the statistics obtained during the beginning period are not reliable enough to capture the trend. In this section, empirical experiments are presented to explain why we set this parameter as *psize*/4. The algorithm on 4 benchmark problems, viz. 100-bits One-Max, 100-bits Random Max, 64-bits Royal Road and ten copies of 3-bits Trap problems, were tested. The characteristics of the four problems are explained below.

The One-Max problem is quite simple. The objective is to find the solution which is a bit string whose bits are all one. The fitness value is equal to the number of 1-bits in the bit string. The Random Max problem is similar to the One-Max problem in finding a bit-string solution whose bit pattern is exactly the same as the one of the target. However, instead of being all one, the target bit pattern is selected randomly. Obtained by comparing bit by bit, the fitness value is the number of bits equal to the associated ones of the target. Notice that this problem is designed to determine whether an algorithm is biased against one or zero.

The Royal Road is a group of bit patterns built up from sequences of short bit patterns. The bit pattern is called schema. There are 15 schemas for 64-bits royal road as shown in Figure 4. After comparing the bit string with each schema, the fitness value is calculated by summing up the numbers of bits equal to those of s_i for all i. For example, a fitness value of a bit string that contains all one (the optimum solution) is $(8 \times 8) + (4 \times 16) + (2 \times 32) + 64 = 256$.

The Trap problem is one of many difficult problems used for testing GAs. It is designed to fool gradient-based optimisers that favour zeroes, but the optimal solution is composed of all 1-bits. We can create a $k \times m$ Trap problem by combining the m groups of a k-bits trap. The fitness value is calculated by summing up the scores associated with all groups. For instance, a 3-bit Trap problem

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

gives a score of 3, 0, 1 and 2 for a group of three, two, one and zero 1-bits respectively. For example, a candidate solution '111 001 110 000 100' has a fitness value of 3 + 1 + 0 + 2 + 1 = 7.

Schema 1 = 11111111**********; $s1 = 8$
Schema 2 = ********11111111**********; $s2 = 8$
Schema 3 = ****************11111111**********; $s3 = 8$
Schema 4 = ************************11111111**********; $s4 = 8$
Schema 5 = ********************************11111111**********; $s5 = 8$
Schema 6 = **11111111**********; $s6 = 8$
Schema 7 = **11111111********; $s7 = 8$
Schema 8 = **11111111; $s8 = 8$
Schema 9 = 111111111111111**********; $s9 = 16$
Schema10 =****************111111111111111**********; $s10 = 16$
Schema11 =********************************111111111111111**********; $s11 = 16$
Schema12 =**111111111111111; $s12 = 16$
Schema13 =11111111111111111111111111111111**********; $s13 = 32$
Schema14 =********************************1111111111111111111111111111111; $s14 = 32$
Schema15 =11111111111111111111111111111111111111111111111111111111111111; $s15 = 64$

Figure 4. Royal Road problem

We ran the proposed algorithm with all benchmark problems described above. For each tournament size of 2, 4 and 8, the parameter was varied among *psize*/2, *psize*/3, *psize*/4 and *psize*/5. The results shown in Table 1 are efficiency ratios [= (solution quality / number of evaluations) × 1000]. The efficiency ratio is used as a quantitative measurement to quantify a quality rate: the higher the rate, the better the efficiency. When the value of *n* is varied from 2 to 4, the efficiency ratio is better when *n* is large (4 or 5) in the case of solving the easy problems (i.e. One-Max and Random Max). For the harder but non-deceptive problem (i.e. Royal Road), a small value of *n* (2 or

Table 1. Efficiency ratio of varying tournament and population sizes in One-Max, Random Max, Royal Road and Trap problems

Problem		Efficiency ratio				Average
		One-Max	Random Max	Royal Road	Trap	
Tournament Size 2	*psize*/2	56.63	32.51	4.97	0.37	23.62
	psize/3	53.14	32.23	5.65	0.42	22.86
	psize/4	56.79	33.68	5.35	0.50	24.08
	psize/5	54.48	34.00	5.35	0.52	23.59
Tournament Size 4	*psize*/2	69.08	41.56	10.03	0.80	30.37
	psize/3	78.88	41.61	10.53	0.95	32.99
	psize/4	89.20	44.76	9.87	0.95	36.20
	psize/5	80.36	49.07	9.65	1.03	35.03
Tournament Size 8	*psize*/2	86.95	42.38	17.51	0.69	36.89
	psize/3	83.52	40.06	14.47	0.91	34.74
	psize/4	87.17	47.69	16.01	1.06	37.99
	psize/5	91.19	47.28	14.03	1.15	38.42

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

3) yields a slightly better ratio. This can be interpreted that the proposed algorithm needs more time to collect more diverse and higher fitness-valued samples before increasing its updating step size. For the deceptive problem (i.e. Trap), the efficiency ratio tends to be relatively high when n is large. This is because in the Trap problem the fb-cGA cannot find a good solution no matter what parameters are ─ the fitness value might remain similar. Therefore, the efficiency depends on the number of fitness evaluations more than the fitness value. In terms of tournament size, a larger size tends to provide a larger efficiency ratio. Overall, almost all of the best quality rates come from *psize*/4 and *psize*/5 (highlighted in Table 1). The average rate of *psize*/4 from all problems and all sizes of the tournament is 32.76, and that of *psize*/5 is 32.35. The *psize*/4 is therefore more desirable in terms of efficiency.

As shown in Figure 5, the convergence graphs, obtained from the One-Max problem experiments (*psize* = 100), reflect the algorithm behaviour. The dash lines are plotted at the generation numbers equal to *psize*/*n* (x = *psize*/*n*), showing when the proposed updating strategy is triggered. If n is larger, the proposed strategy starts sooner. Passing this line, the algorithm updates the PV with a larger step size when the winner's bit conforms to the majority direction (i.e. meeting the condition on line 8 or 16 in Figure 3). As the graphs show, the fitness values gradually improve in the early generations (generation number < *psize*/*n*) but increase abruptly after the trigger. This behaviour explains why the algorithm's PV converges to a solution using fewer function evaluations.

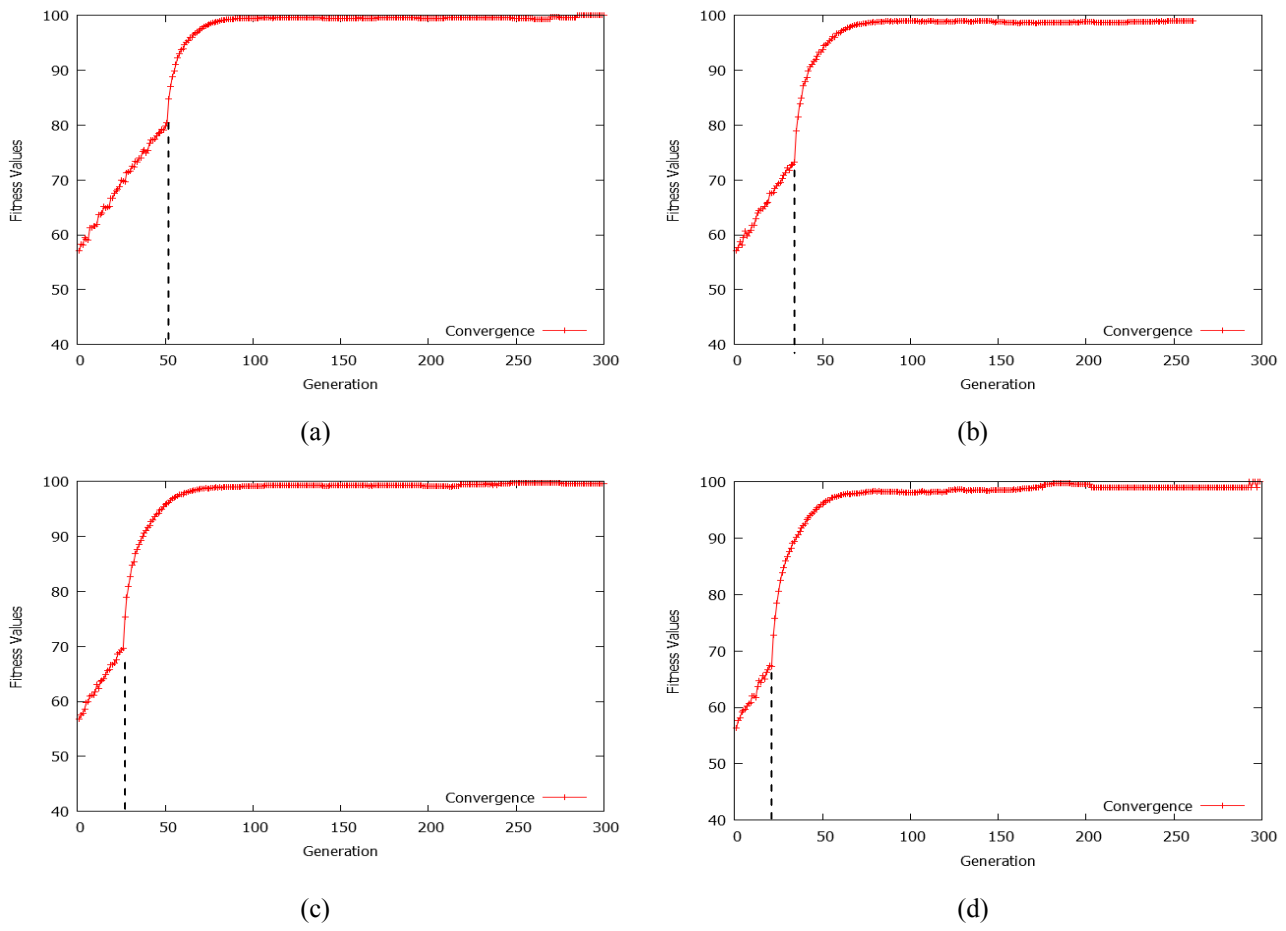


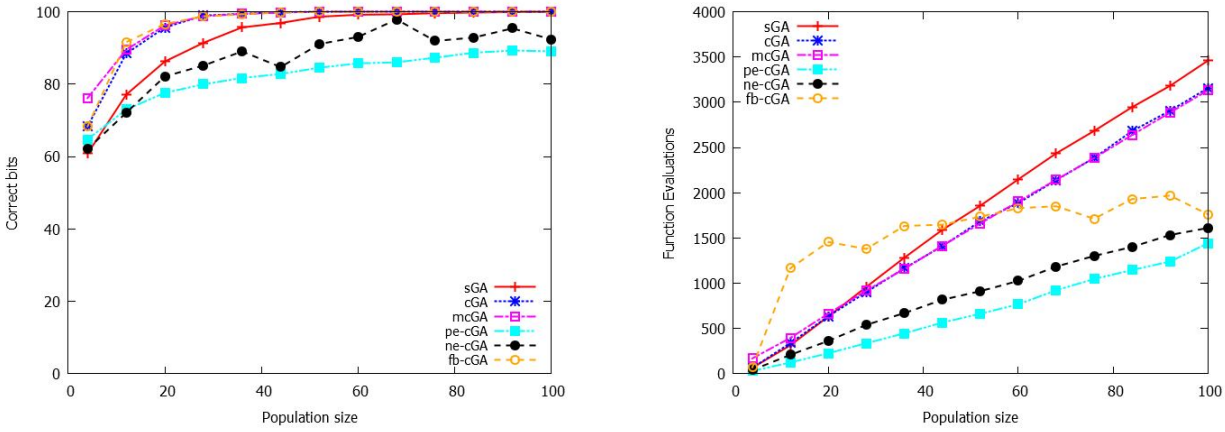
Figure 5. Convergence graphs of experiments with parameters: (a) psize/2, (b) psize/3, (c) psize/4 and (d) psize/5

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

PERFORMANCE COMPARISONS

At first, we tested all of the algorithms – sGA, cGA, mcGA, pe-cGA, ne-cGA and fb-cGA – with the 100-bit One-Max problem. Each graph in Figure 6 shows the results when the parameter *psize* (population size) varies between 4-100 with a step value of 8. All algorithms used the tournament size of 2. Each line shows an average result from 50 runs. In general, when the population size becomes larger, GAs take more function evaluations but yield better solutions.

Figure 6a shows that the solution quality (the numbers of correct bits) obtained from fb-cGA is comparable with those obtained from sGA, cGA and mcGA, while pe-cGA and ne-cGA have lower solution qualities. In terms of the number of function evaluations, Figure 6b shows that fb-cGA outperforms sGA, cGA and mcGA if the population size is large. When it is small (*psize* < 40), fb-cGA needs more function evaluations than do others. The example where *psize* is equal to 4 is used to explain this situation; the algorithm collects the statistics merely from one generation (*psize*/4 = 1) as a guide to update the PV with a large step size. The triggering time may be too early, leading the vector to a wrong direction. Consequently, the algorithm would spend more time (i.e. a larger number of function evaluations) searching before coming back to the right direction. Nevertheless, the number of fitness evaluations of fb-cGA does not increase as much as the population size and is comparable to those of pe-cGA and ne-cGA when the population size is 100.



(a) Solution quality (b) Number of function evaluations

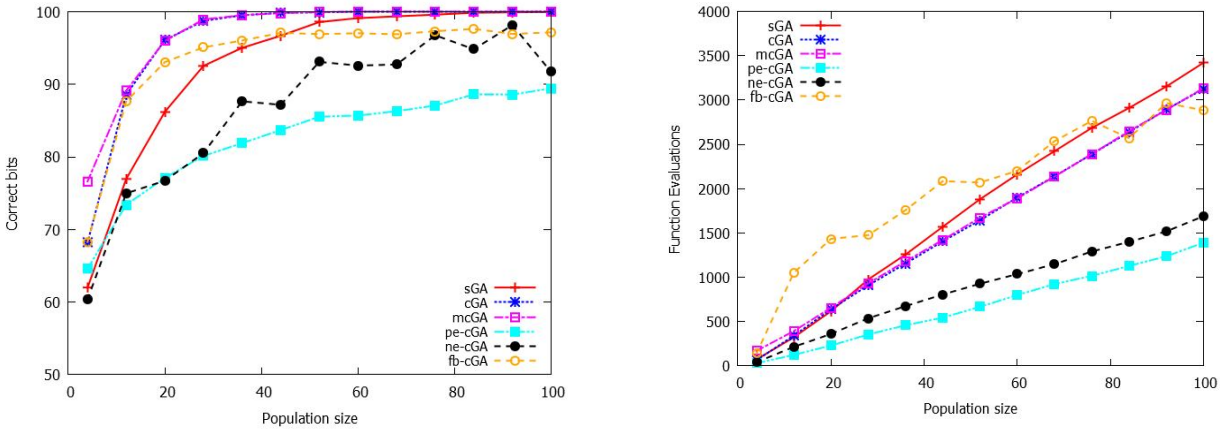
Figure 6. Correct bits and function evaluations (of all the algorithms) in One-Max problem

Figure 7 shows the performance of all the algorithms on the Random Max problem. The fb-cGA performance is moderate when compared with other techniques. It requires a large number of function evaluations to find the solution in the case of a small population, but when the population size increases the numbers of function evaluations tend to be comparable to those in cGA and mcGA.

Figure 8 shows the performance of all the algorithms on the Royal Road problem using tournament sizes of 2, 4 and 8. The number of population sizes varies between 4-100 with a step value of 8. Figures 8a, 8c and 8e show the solution quality in terms of fitness value. Figures 8b, 8d and 8f show the numbers of function evaluations. The fb-cGA yields comparable results in terms of solution quality with those from sGA, cGA and mcGA while requiring a much smaller number of function evaluations. When compared with ne-cGA in the case of tournament size of 2, fb-cGA has a higher fitness value than that of ne-cGA but requires more function evaluations. However, for

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

tournament sizes of 4 and 8, fb-cGA yields comparable fitness values with those of ne-cGA and needs fewer fitness evaluations. In this problem, pe-cGA requires the smallest number of function evaluations but yields lowest fitness values.



(a) Solution quality

(b) Number of function evaluations

Figure 7. Correct bits and function evaluations (of all the algorithms) in Random Max problem

Figure 9 shows the algorithms' performance on the 3-Trap problem – the Trap problem with a group of 3 bits (k=3) – using tournament sizes of 2, 4 and 8, and population sizes of 8, 500, 1000, 1500, 2000, 2500 and 3000. Figures 9a, 9c and 9e show the solution quality in terms of the number of correct building blocks (the number of 3-bits blocks containing all 1-bits). Figures 9b, 9d and 9f show the numbers of function evaluations taken to find the solution. Figure 9a shows that the solution quality of fb-cGA is higher than that of sGA, cGA and mcGA, but lower than that of pe-cGA and ne-cGA. The fb-cGA requires the smallest number of function evaluations, using them approximately 14, 9, 12, 3 and 2 times fewer than do sGA, original cGA, mcGA, pe-cGA and ne-cGA respectively. This cofirms the efficiency of the proposed method in terms of the number of function evaluations saved.

ANALYSIS OF ALGORITHMS' CONVERGENCE AND BEHAVIOUR

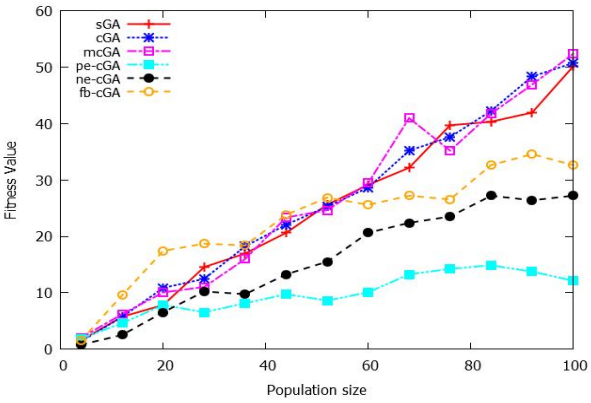
The convergence analysis was carried out by using plots of the fitness values over time (generations). The behaviour analysis was performed through the graphic representation of all probability values in the PV from the first to the last generation to track how the probabilities change.

Figure 10 shows the fitness values and the probability values of cGA, mcGA, pe-cGA, ne-cGA and fb-cGA for the One-Max problem with tournament sizes of 2 and *psize* of 100. The graphs on the right show probability values in density of greyscale. Bearing in mind that the objective of the One-Max problem is to find a solution in which all bits are 1, all the shades representing probability values shown in the graphs on the right should fade to white (probability = 1) in the final generation.

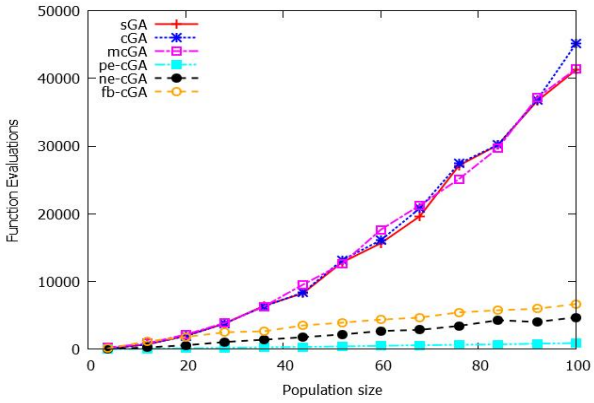
The cGA, mcGA and fb-cGA can find the optimal solution (fitness value = 100), the ne-cGA yields a result very close to the optimal, while the pe-cGA's PV converges to one far from the optimal. The convergence graphs of cGA and mcGA are very similar. Their PV converges to the solution at nearly the same generation. However, the way in which the probabilities change is slightly different. The shades of mcGA fade quicker and more smoothly than do those of cGA. The

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135

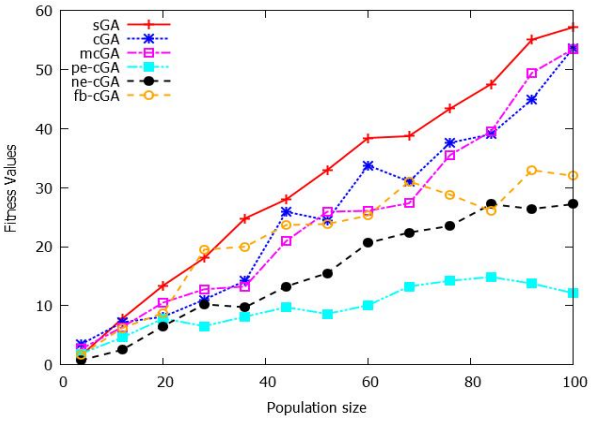
smooth change in the probability values of mcGA is in accordance with its updating rule in that the moving average approach waits to see the trend, thus slowing down the increase or decrease in the probability values.



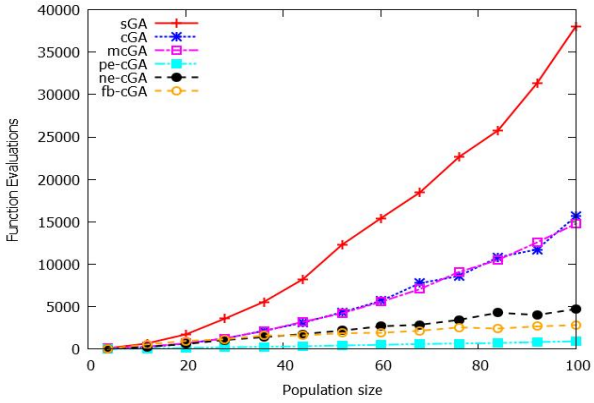
(a) Solution quality (tournament size 2)



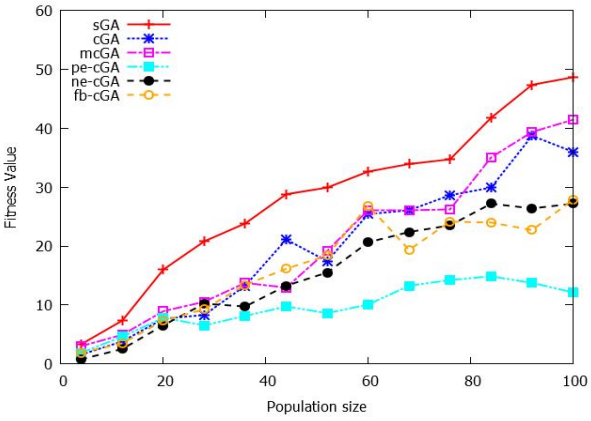
(b) Number of function evaluations (tournament size 2)



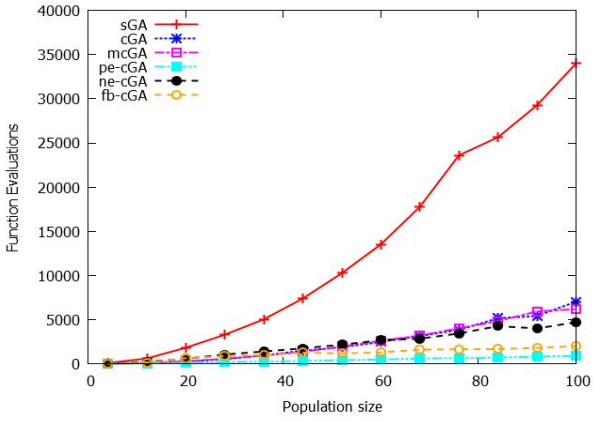
(c) Solution quality (tournament size 4)



(d) Number of function evaluations (tournament size 4)



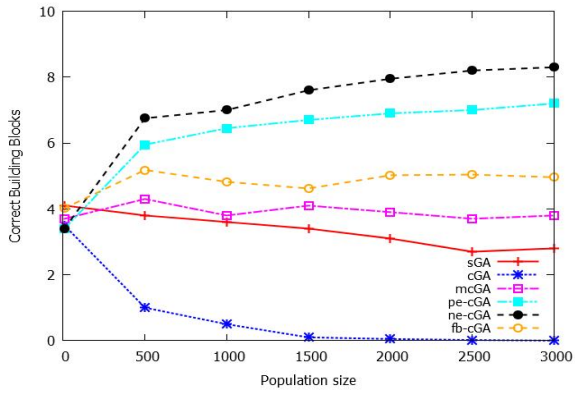
(e) Solution quality (tournament size 8)



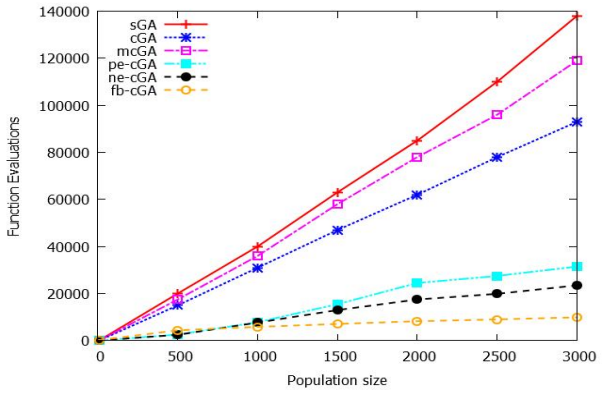
(f) Number of function evaluations (tournament size 8)

Figure 8. Fitness values and function evaluations (of all the algorithms) in Royal Road problem

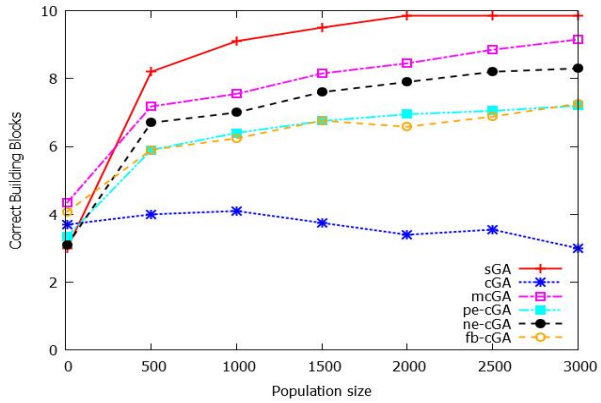
Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135



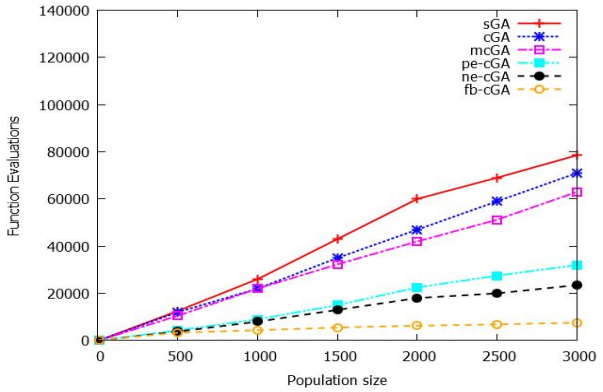
(a) Solution quality (tournament size 2)



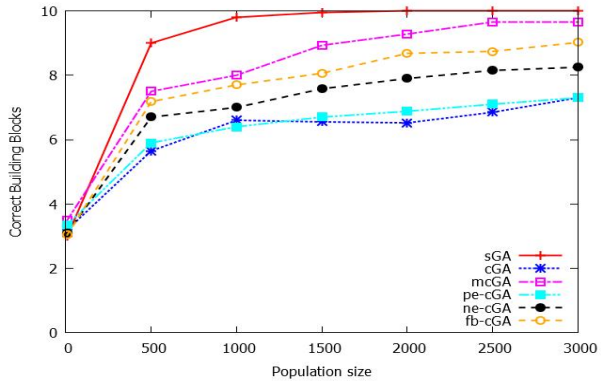
(b) Number of function evaluations (tournament size 2)



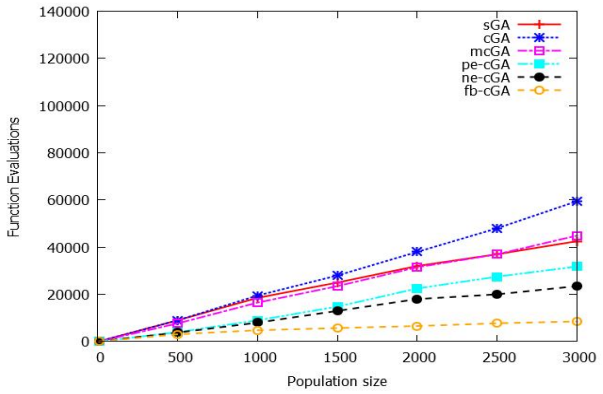
(c) Solution quality (tournament size 4)



(d) Number of function evaluations (tournament size 4)



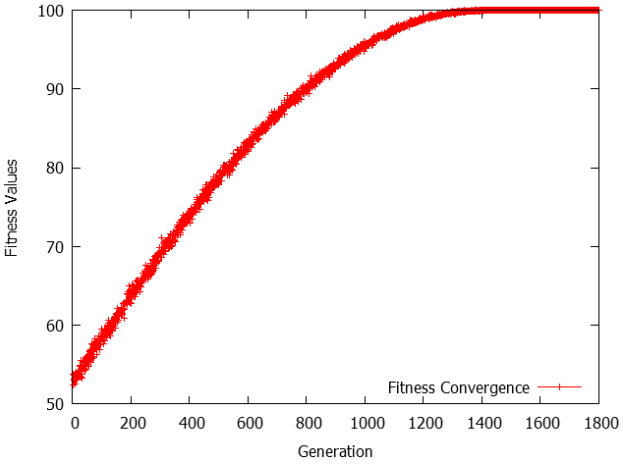
(e) Solution quality (tournament size 8)



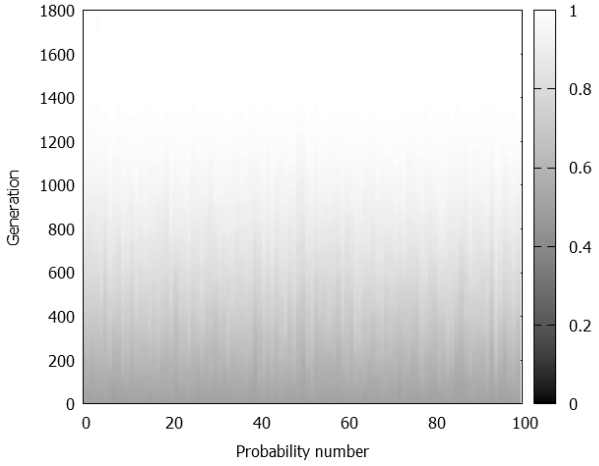
(f) Number of function evaluations (tournament size 8)

Figure 9. Correct building blocks and function evaluations (of all the algorithms) in Trap problem

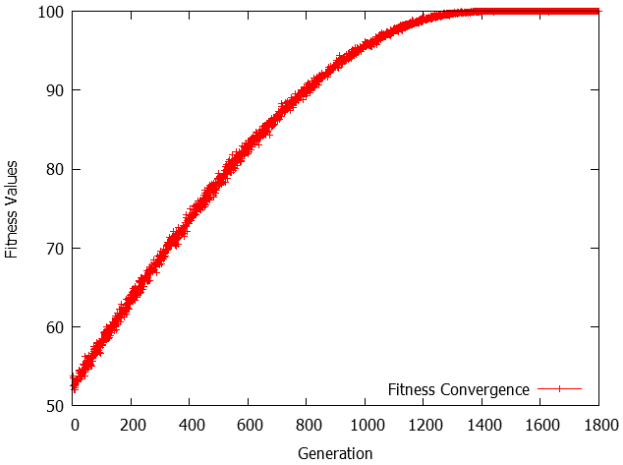
Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135



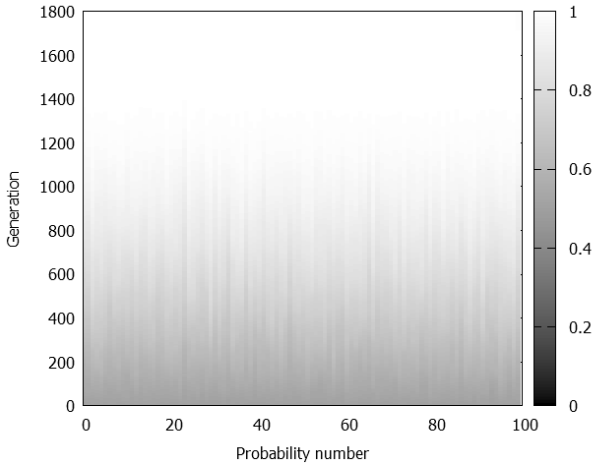
(a) Convergence of fitness values (cGA)



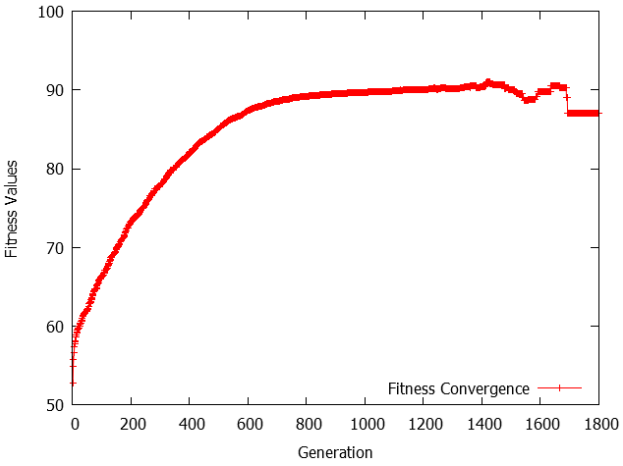
(b) 100 probability values of probability vector (cGA)



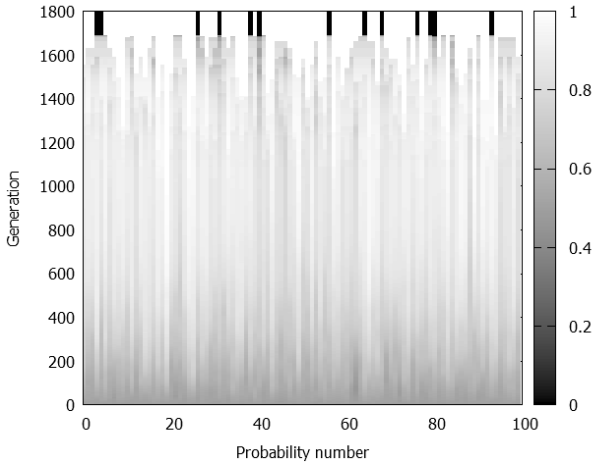
(c) Convergence of fitness values (mcGA)



(d) 100 probability values of probability vector (mcGA)



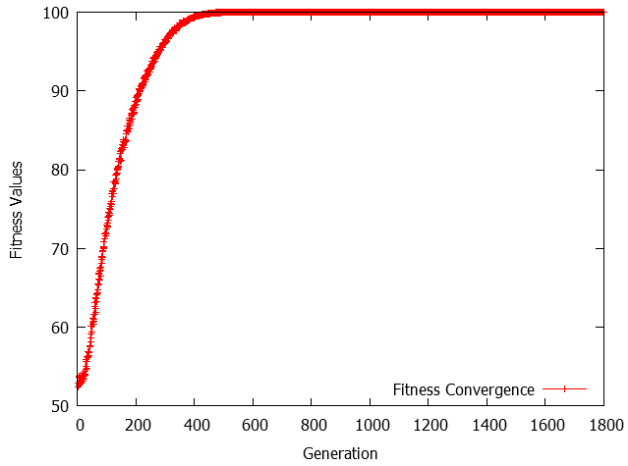
(e) Convergence of fitness values (pe-cGA)



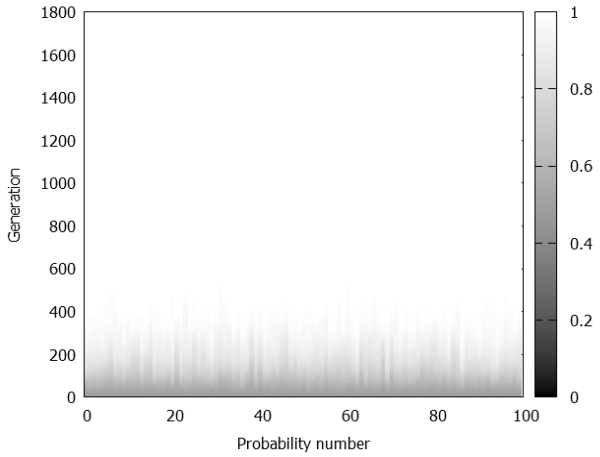
(f) 100 probability values of probability vector (pe-cGA)

Figure 10. Convergence of fitness values and change in 100 probability values in the probability vectors at each generation. Darker shading represents the probability closer to 0 while white represents the probability of 1. All probabilities are initialised to 0.5. All plots show the average values from 50 runs.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 121-135



(i) Convergence (fb-cGA)



(j) Probability (fb-cGA)

Figure 10 (continued). The convergence and probability analysis

Both the pe-cGA's and ne-cGA's shades turn light grey faster than do the cGA's and mcGA's, as shown in the early generations of Figures 10f and 10h. However, the pe-cGA's final PV is unfavourable and the ne-cGA's approaches, but does not quite reach, the optimum. This is characteristic of elitism. The elite (the best solution so far) often contains zero bits in the chromosome, which deceives the algorithm into updating the probability towards zero. In the case where newly generated candidate solutions are worse than the elite, the PV is updated towards the elite again and the probability at the associated position may come closer to zero. This situation may lead the elitism-based algorithms to update the PV towards the wrong direction. Elitism affects the pe-cGA's performance more than the ne-cGA's due to its everlasting elite.

The fb-cGA's PV converges to the solution faster than the other algorithms (Figure 10i). The shade representing probability values in Figure 10j turns white in a small number of generations. To efficiently apply our proposed technique to a real-world problem, the chromosome should have all of its bits uncorrelated with one another. It is important to realise that the fb-cGA evolves its PV by updating all associated probabilities of all bits. The update is done only one bit at a time without taking into account the information of the other bits. The results of One-Max problem shown in Figure 10 serve as an example that supports this claim: fb-cGA can solve the problem using far fewer number of generations (approximately 400, instead of about 1300 generations required by the traditional cGA). This significant outperformance stems from the new dynamic updating strategy: the proposed technique decides to update the probabilities with a larger step size based on the collected statistics. However, if the chromosome bits of the real-world problem are correlated with one another, the proposed algorithm might not find the best solution, as shown in the Royal Road and the Trap problem.

For a real-world optimisation problem that has multiple local optima, there is a higher chance that the fb-cGA may get stuck at a local optimum. For example, in the field of computational vision, efficient algorithms such as cGA may be used to recognise objects in an image. However, if the input image is complex, it might introduce various local optima in the search space. Because fb-cGA uses a trend in early generations to quickly decide to update PV with a larger updating step size, it might prematurely decide to search towards a seemingly promising direction at a certain time. Once the proposed algorithm gets stuck, it is hard to escape from the local optimum because it already has a strong bias in favour of either zero or one. To handle this kind of problem, we should wait longer to

see a correct trend before using a large step size. In addition, the step size should be incrementally increased during the evolution.

CONCLUSIONS

This paper presents a behaviour analysis of fb-cGA. To update the PV, the fb-cGA collects and utilises the update number of each probability in both up and down directions. The numbers of updates are used to adjust probability-updating step sizes, turning the vector towards the promising direction faster. When the effect of parameter *psize*/*n* on the algorithm performance was investigated, the results suggested that the newly proposed updating strategy should be used when the generation number is greater than *psize*/4. The analysis, through graphic representation of all probabilities from the first to the last generation, shows that the fb-cGA updates the PVs towards the solution quicker than the other algorithms and also requires fewer function evaluations.

ACKNOWLEDGEMENTS

This work was funded by Thailand Research Fund, the Office of Higher Education Commission and the Faculty of Informatics, Burapha University (Grant No. TRG5680073). We also thank our mentor, Prof. Prabhas Chongstitvatana, for his guidance, support and encouragement.

REFERENCES

1. G. R. Harik, F. G. Lobo and D. E. Goldberg, "The compact genetic algorithm", *IEEE Trans. Evol. Comput.*, **1999**, *3*, 287-297.
2. M. H. Afshar, "Application of a compact genetic algorithm to pipe network optimization problems", *Transact. A*: *Civil Eng.*, **2009**, *16*, 264-271.
3. R. D. Al-Dabbagh, M. S. Baba, S. Mekhilef and A. Kinsheel, "The compact genetic algorithm for likelihood estimator of first order moving average model", Proceedings of 2nd International Conference on Digital Information and Communication Technology and Its Applications, **2012**, Bangkok, Thailand, pp.474-481.
4. R. D. Al-Dabbagh, A. Kinsheel, M. S. Baba and S. Mekhilef, "An integration of compact genetic algorithm and local search method for optimizing ARMA (1, 1) model of likelihood estimator", Proceedings of 2nd International Conference on Computer Science and Computational Mathematics, **2013**, Kuala Lumpur, Malaysia, pp.60-67.
5. C. F. M. Toledo, M. S. Arantes, R. R. R. Oliveira and A. C. B. Delbem, "A hybrid compact genetic algorithm applied to the multi-level capacitated lot sizing problem", Proceedings of 28th Annual ACM Symposium on Applied Computing, **2013**, Coimbra, Portugal, pp.200-205.
6. R. R. Silva, H. S. Lopes and C. R. E. Lima, "A compact genetic algorithm with elitism and mutation applied to image recognition", *Lect. Notes Comput. Sci.*, **2008**, *5227*, 1109-1116.
7. P. Olarthichachart, S. Kaitwanidvilai and S. Karnprachar, "Trip frequency scheduling for traffic transportation management based on compact genetic algorithm", Proceedings of International MultiConference of Engineers and Computer Scientists, **2010**, Hong Kong, pp.1072-1074.
8. R. D. H. Al-Dabbagh, "Compact genetic algorithm for cryptanalysis trapdoor 0-1 knapsack cipher", *J. Al-Nahrain Univ.*, **2009**, *12*, 137-145.
9. A. Azouaoui, A. Berkani and M. Belkasmi, "An efficient soft decoder of block codes based on compact genetic algorithm", *Int. J. Comput. Sci. Iss.*, **2012**, *9*, 431-438.
10. H. Xing and R. Qu, "A compact genetic algorithm for the network coding based resource minimization problem", *Appl. Intell.*, **2012**, *36*, 809-823.

11. P. Gupta and R. Tiwari, "Solving three dimensional bin packing problem using elitism based genetic algorithm", *Int. J. Adv. Res. Comput. Eng. Technol.*, **2012**, *1*, 471-475.
12. P. K. Singh and N. Sahu, "Task scheduling in grid computing environment using compact genetic algorithm", *Int. J. Sci. Eng. Technol. Res.*, **2014**, *3*, 107-110.
13. Y. C. Huang, C. F. Chang, C. H. Chan, T. J. Yeh, Y. C. Chang, C. C. Chen and C. Y. Kao, "Integrated minimum-set primers and unique probe design algorithms for differential detection on symptom-related pathogens", *Bioinformatics*, **2005**, *21*, 4330-4337.
14. A. Bade, I. M. Aref, B. M. Hussien and Y. Eman, "Solving protein folding problem using elitism-based compact genetic algorithm", *J. Comput. Sci.*, **2008**, *4*, 525-529.
15. C. Aporntewan and P. Chongstitvatana, "A hardware implementation of the compact genetic algorithm", Proceedings of IEEE Congress on Evolutionary Computation, **2001**, Seoul, Korea, pp.624-629.
16. J. C. Gallagher and S. Vigraham, "A modified compact genetic algorithm for the intrinsic evolution of continuous time recurrent neural networks", Proceedings of Genetic and Evolutionary Computation Conference, **2002**, New York, USA, pp.163-170.
17. J. C. Gallagher, S. Vigraham and G. Kramer, "A family of compact genetic algorithms for intrinsic evolvable hardware", *IEEE Trans. Evol. Comput.*, **2004**, *8*, 111-126.
18. K. M. Timmerman, "A hardware compact genetic algorithm for hover improvement in an insect-scale flapping-wing micro air vehicle", *Master Thesis*, **2012**, Wright State University, USA.
19. C. Zhou, K. Meng and Z. Qiu, "Compact genetic algorithm mutated by bit", Proceedings of 4th World Congress on Intelligent Control and Automation, **2002**, Shanghai, China, pp.1836-1839.
20. B. V. Ha, R. E. Zich, M. Mussetta, P. Pirinoli and C. N. Dao, "Improved compact genetic algorithm for EM complex system design", Proceedings of 4th International Conference on Communications and Electronics, **2012**, Hue, Vietnam, pp.381-392.
21. S. Rimcharoen, D. Sutivong and P. Chongstitvatana, "Updating strategy in compact genetic algorithm using moving average approach", Proceedings of IEEE Conference on Cybernetics and Intelligent Systems, **2006**, Bangkok, Thailand, pp.690-695.
22. C. W. Ahn and R. S. Ramakrishna, "Elitism-based compact genetic algorithms", *IEEE Trans. Evol. Comput.*, **2003**, *7*, 367-385.
23. J. Y. Lee, S. M. Im and J. J. Lee, "Bayesian network-based non-parametric compact genetic algorithm", Proceedings of 6th IEEE International Conference on Industrial Informatics, **2008**, Daejeon, Korea, pp.359-364.
24. J. Y. Lee, M. S. Kim and J. J. Lee, "Compact genetic algorithms using belief vectors", *Appl. Soft Comput.*, **2011**, *11*, 3385-3401.
25. S. Phiromlap and S. Rimcharoen, "A frequency-based updating strategy in compact genetic algorithm", Proceedings of International Computer Science and Engineering Conference, **2013**, Nakorn Pathom, Thailand, pp.207-211.