PAPER

A Timed-Based Approach for Genetic Algorithm: Theory and Applications

Amir MEHRAFSA[†], ***Member***, **Alireza SOKHANDAN**[†], ***Nonmember***, ***and*** **Ghader KARIMIAN**[†a)], ***Member***

SUMMARY In this paper, a new algorithm called TGA is introduced which defines the concept of time more naturally for the first time. A parameter called TimeToLive is considered for each chromosome, which is a time duration in which it could participate in the process of the algorithm. This will lead to keeping the dynamism of algorithm in addition to maintaining its convergence sufficiently and stably. Thus, the TGA guarantees not to result in premature convergence or stagnation providing necessary convergence to achieve optimal answer. Moreover, the mutation operator is used more meaningfully in the TGA. Mutation probability has direct relation with parent similarity. This kind of mutation will decrease ineffective mating percent which does not make any improvement in offspring individuals and also it is more natural. Simulation results show that one run of the TGA is enough to reach the optimum answer and the TGA outperforms the standard genetic algorithm.
key words: *genetic algorithms, time unit, time to live, population, generator, crossover probability, GAVaPS, premature convergence, random search algorithm.*

1. Introduction

Genetic algorithms (GAs) are a family of probabilistic search algorithms inspired by the biological processes of genetics and evolution. GA developed by Holland [1] and were applied to many practical problems by Goldberg [2]. These algorithms became the most popular evolutionary algorithm due to their successful application in optimization and search problems, particularly in those problems in which the size or complexity of the search space renders infeasible the use of other optimization techniques [3].

A GA allows a population of individuals represented by chromosomes to evolve under specified selection rules for crossover and mutation to a state that minimizes the cost function, i.e. maximizes the "fitness" [4].

Behavior and performance of genetic algorithms are directly affected by the values of their input parameters. Poor parameter settings usually lead to several problems such as the premature convergence [5].

The most important parameters are the strategy or control parameters: population size, mutation rate, and crossover rate [6]. Finding robust methods for determining the optimum values of these parameters is probably impossible, since the optimal values are problem-dependent and the GA parameters interact with each other in a complex way [7]. Moreover, the optimal values can be different in different phases of one single run on a given problem.

Several features have recently been added to the basic genetic algorithm in order to emulate the natural evolution process as precisely as possible. The size of the population is one of the most important parameters of GAs and may be critical in many applications. If the population size is too little, the algorithm may converge too quickly to a local optimum. However, if the population size is too big, the algorithm may become a random search wasting computational resource [8], [9].

Arabas et al. [8] propose the Genetic Algorithm with Variable Population Size (GAVaPS) by introducing the age and maximum lifetime properties for individuals. The maximum lifetime depends on the fitness of the corresponding individual and it is assigned only once to each individual at the moment they are born and remains constant during their evolution, while the age (initialized to zero at birth) is incremented at each generation by one. Individuals are removed from the population when their ages reach the value of their predefined maximal lifetime [10]–[12].

GAVaPS was very sensitive to the reproductive ratio parameter, and the algorithm frequently increased the size of the population over several thousand individuals, which resulted in unreliable performance [13]. Back et al. proposed a variant of GAVaPS, namely the Adaptive Population size GA (APGA) by leaving the best individual unchanged when individuals grow older [6]. It was shown that the APGA is not capable of properly adapting the population size, and that its newly introduced parameters act as the actual population size parameter of a traditional GA [14]. Harik and Lobo [15] introduce parameter-less GA, where several populations with different sizes evolve in parallel starting with small population sizes. By inspecting the average fitness of these populations, less fit undersized populations are replaced by larger ones. Eiben et al. [13] suggest using the pace of fitness improvements as a signal to control population size in Population Resizing on Fitness Improvement GA (PRoFIGA).

Last and Eyal [5] developed a GA with varying population size, where chromosomes are classified into young, middle-age and old according to their age and lifetime. In this GA, crossover probability is a function of parent's age-type (young, middle-aged, old, etc.) and is obtained using a fuzzy rule base and fuzzy possibility theory. Roy et al. [16] proposed a fuzzy genetic algorithm (FGA). Suitable membership function Parameters, as well as the settings of the "expert knowledge", requires further research and experi-

Manuscript received August 12, 2010.
Manuscript revised February 24, 2011.
[†]The authors are with Electrical and Computer Engineering Department, University of Tabriz, Tabriz, Iran.
a) E-mail: karimian@tabrizu.ac.ir
DOI: 10.1587/transinf.E94.D.1306

mentation.

In this paper, a new algorithm called TGA (Timed-GA) is introduced which defines the concept of time more naturally for the first time. The mutation operator is used more meaningfully in the TGA. Simulation results show that the TGA outperforms standard genetic algorithm.

This paper is organized as follows: Sect. 2 describes the proposed algorithm. Section 3 discusses results of few experiments, and some simulation results are provided and compared to standard GA and GAVaPS. The final section is the conclusions.

2. The Proposed Algorithm

2.1 General Description of the Proposed Algorithm

More practical problems are problems with large or infinite state space and mostly non-systematic, or goal-based algorithms are used for solving these problems. For an ideal non-systematic algorithm, three features could be considered.

First, the cycle of algorithm execution could lead to a more optimal answer. In other words, the continuation of algorithm execution should guarantee the chance of searching spaces which are not searched yet. Most of the random search algorithms implement this feature.

Second, the algorithm should start exploring the state space with large steps. And when a relative convergence around the optimal answer appears, it continue to explore the state space with smaller steps. This feature of genetic algorithm, like Stochastic Beam Search, combines uphill tendency using random exploration with exchange of information between parallel strings [18].

Third, The movement of the algorithm through state space should always put the algorithm in a more compatible state with the desired goal, i.e. the movement of the algorithm should provide necessary convergence to expedite the process of reaching the optimal answer. The GA provides the third feature by applying the law of "Survival of the fittest" or "Elitism" to the population of chromosomes.

One problem of genetic algorithm is the extreme use of survival of the fittest law, to an extent that almost all chromosomes with higher finesses than average fitness are more likely to acquire eternal life. Therefore, the individuals will become more similar to each other resulting in a premature convergence or algorithm stagnation. This, in turn, will raise the necessity of algorithm execution for several times from the beginning to make sure that the most efficient answer is found. Therefore, the first feature is sacrificed for the sake of the third one.

This problem of GA could be solved by applying the law of survival of the fittest in a more normal and natural way, as mentioned in this article. We use a parameter called "TimeToLive" to solve this problem. The TimeToLive parameter for a chromosome is a time that it participates in the process of the algorithm. The parameter has a direct relationship with chromosome fitness and inverse relationship with the population fitness range, i.e. the interval between the best and worst fitness of population. There is an algorithm called GAVaPS which utilizes a parameter called "lifetime". This parameter assigns a time concept to individuals in a similar manner with our TimeToLive parameter. Yet, GAVaPS has a completely different purpose of using this parameter which will be explained in more details in Sect. 3.1.

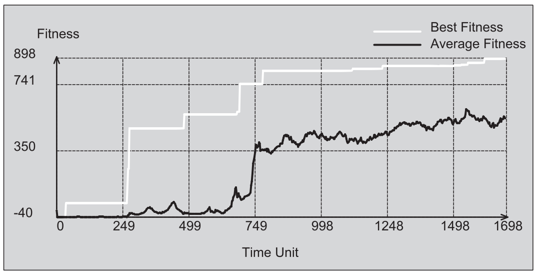
The other problem with genetic algorithm is that it applies mutation only to temporarily avoid premature convergence, which seems inefficient. In the proposed algorithm, always the whole population is in the mating pool but individuals with higher fitness have more chance of being selected for mating. Mutation is only applies to offsprings and its probability has direct relation with the parent similarity, i.e. that is to say that higher the similarity of the parent chromosome, the higher mutation probability. This kind of mutation will decrease ineffective mating percentage that does not make any improvement in offsprings and also is closer to the natural mutation. This will be explained in more details in Sect. 2.6.

The other point is that in the proposed algorithm, even the chromosomes placed in the infeasible space of the problem do not lose the chance of participation, although their participation is low. The proposed algorithm guarantees not to result in premature convergence or stagnation providing necessary convergence to achieve optimal answer. Indeed, the positive features of genetic algorithm in converging population to achieve optimal answer are combined with one of the features of random search algorithms which is the hope of finding better answers if the algorithm execution is continued. Therefore in finding the optimum answer using the proposed algorithm, unlike other non-systematic algorithms derived from genetic algorithm, it is not necessary to execute it for several times. In addition, the use of any relatively improved individuals in the population will not have any undesirable effect from the point of premature convergence on the algorithm process.

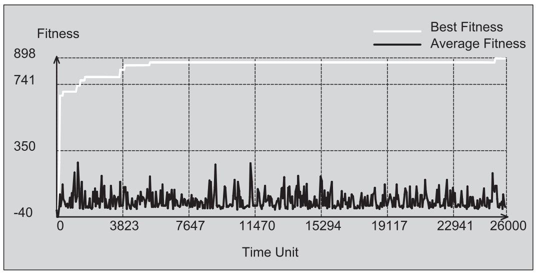
2.2 Time Concept in the Proposed Algorithm

The process of the proposed algorithm is based on time. Each individual has longevity and its life will be reduced in time. The remaining life time of each chromosome is calculated using its birth time and the value of its fitness. This parameter is called "TTL" which stand for TimeToLive.

The proposed algorithm represents a time axis called "TimeLine" which is used to illustrate the remaining longevity of each chromosome. The minimum value of TimeLine is always zero and we will use a parameter called "MaxLife" as its maximum value. It should be noted that inappropriate value assignment to MaxLife or other parameters of the algorithm will not lead to violations of mentioned features in the proposed algorithm, but it will only reduce its speed. For example, we run the problem 3.2.5 twice with values of 50 and 200 for MaxLife while keeping other parameters of algorithm unchanged. The result is



(a)



(b)

Fig. 1 Average fitness and Best fitness of purposed algorithm with respect to the time unit for the problem 3.2.5; (a) MaxLife is 200 and the time to reach the optimum answer is 1615 time unit (b) MaxLife is 50 and the time to reach the optimum answer is 25341 time unit.

shown in Fig. 1. The effects of the changes in the values of the algorithm parameters are studied in Sect. 3.3.

Since mutation and crossover operators apply to individuals through their longevity, therefore the generation concept, which GA uses is not needed by the proposed algorithm anymore. The methods of applying crossover and mutation operators will be explained later in this article.

In each time unit, the remaining MaxLife of each chromosome is decreased by one until the TTL parameter becomes zero, and then the chromosome will be removed from the process of algorithm execution. Therefore, none of the chromosomes will get immortal life during the algorithm process. It can be concluded from what we have mentioned so far, chromosomes with higher fitness will achieve higher TTL and this will make the chromosome participate in the process of the algorithm for longer periods of time.

The value of TTL for each chromosome has a direct relationship with its fitness and will be recalculated for each chromosome *i* in each time unit as mentioned in Eq (1).

$$TTL[i] = \frac{MaxLife}{|BestFitness - WorstFitness|} \times |Fitness[i] - WorstFitness| \quad (1)$$

Moreover, as Eq. (1) represents, for calculating TTL[i] a mapping from range of [Worst Fitness, Best Fitness] to TimeLine is applied first. Then the Fitness value of each chromosome is mapped onto the TimeLine.

The problem with the proposed method is that if a chromosome gets the worst fitness at BirthTime, according to the Eq (1), its TTL will be zero and it will be removed from the algorithm process at the time of its creation.

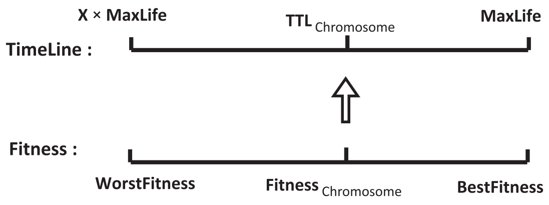


Fig. 2 Mapping fitness value to TTL.

To solve this problem, the fitness values is mapped onto the range of [X × MaxLife, MaxLife] instead of [0, MaxLife], where X is a number between zero and one. This mapping is shown in Fig. 2. So, the TTL calculation method is revised from Eq (1) to Eq. (2).

$$TTL[i] = \frac{(1 - X) \times MaxLife}{|BestFitness - WorstFitness|} \times |Fitness[i] - WorstFitness| + X \times MaxLife \quad (2)$$

The variable X in the Eq (2) is the maximum chance of worst chromosomes or chromosomes that are in the infeasible space of problem to participate in the algorithm process.

It should be noted that the proposed algorithm does not put any chromosome out of algorithm process, even if the chromosome is in the infeasible space of problem. This feature is more useful in applications in which their infeasible state spaces are greater than their feasible state spaces. The problem 3.2.5 demonstrates this feature clearly.

Where the values of the BestFitness and WorstFitness are equal at the time of first chromosome generation, the value of WorstFitness is considered one unit less than the BestFitness; therefore, according to the Eq (2), the TTL value of first chromosome equals to the value of MaxLife.

Another point which should be considered is that whenever the BestFitness or WorstFitness values of the algorithm change, the TTL value of the whole chromosomes changes, too.

Pseudo Code 1 Generator work conditions

```
if (ChromosomeCount < MaxChromosome × a)
  start Generator
elseif (ChromosomeCount > MaxChromosome × b)
  stop Generator
MaxChromosome × a ≥ 2
  $0 < a, b < 1$
  $a < b$
```

2.3 Generator

One of the features of the proposed algorithm is a unit called generator, which is responsible to produce the new chromosomes of the population. The generator will start to work in two conditions: First, when the algorithm begins and there are not enough chromosomes in the population. Second,

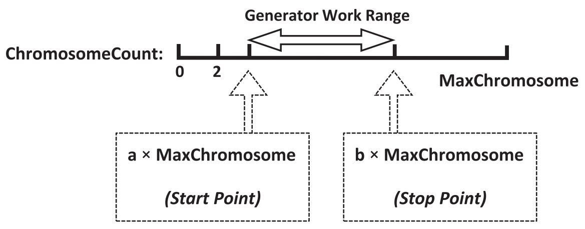


Fig. 3 Generator work range as a function of chromosomecount.

when most of chromosomes die out. This will happen because their TTL gets a small value due to a sudden difference between the best and average fitness. In these two cases, the generator produces new random chromosomes for resolving population lack problem.

Unlike the GA and its derived algorithms there is no first population concept in the proposed algorithm, and chromosomes will be produced during the time intervals if necessary.

When a chromosome with more optimal fitness than average fitness has been generated, the new TTL assignment to individuals cause the inferior individuals to achieve smaller TTL value, so they die out much earlier; then the new randomly generated chromosomes with higher finesses will correct the wrong way of population convergence. The new random chromosomes which are placed in older section of population, die sooner. Therefore, the new chromosomes generation is continued by the generator until the algorithm reach its goal.

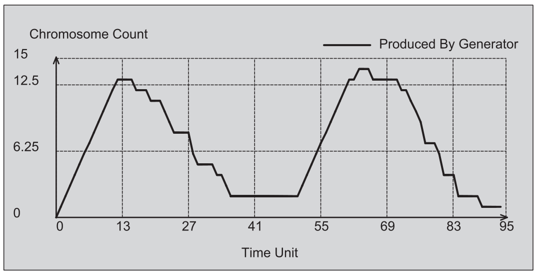
In the proposed algorithm a new parameter called "MaxChromosome" is used to represent the maximum number of individuals allowed. Since all chromosomes have permission to participate in the algorithm process proportional to their TTL values, it is possible that the number of chromosomes may grow uncontrolled, so the population size is controlled by MaxChromosome.

If the population size is less than the percentage of the MaxChromosome, the generator starts to work and produces new random chromosome on each time unit. The generator functionality is shown in Pseudo code 1.

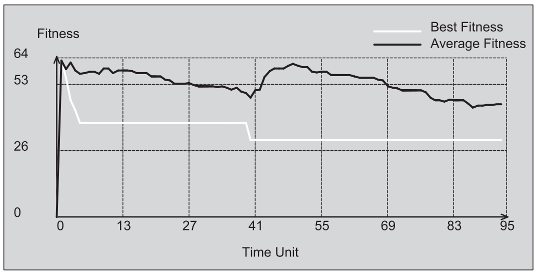
According to the Pseudo code 1, when the size of population shown by "ChromosomeCount" is less than the percentage of the MaxChromosome, e.g. MaxChromosome/4, then the generator starts and produces new chromosome on each time unit, and when the population size is somewhat more, e.g. MaxChromosome/2, then the generator stops.

In Pseudo code 1, 'a' and 'b' are user-defined parameters. If the distance between a and b is large, the generator operates for longer time, and scattering of chromosomes will be more on the TimeLine because of more random chromosome production.

However, as mentioned above, the value of (MaxChromosome $\times$ a) should not be less than two, because in this case, only one chromosome is produced by generator, therefore no crossover operation will occur and this causes the algorithm to fail. The efficiency of the generator based on the *a* and *b* parameters is shown in Fig. 3.



(a)



(b)

Fig. 4 Generator operation over time. (a) Population produced by generator w.r.t time, (b) Fitness values w.r.t time. Generator starts to generate new chromosomes at time T=40 after it was already stopped.

The Fig. 4 presents a sample run of generator for problem 3.2.7. According to Fig. 4, when the population begins to converge, the generator tends to stop generation of chromosomes. In the time of 40, a sudden improvement in the value of best fitness causes the denominator of fraction in Eq. (2) to increase which results in TTL reduction of chromosomes. Therefore, the old chromosomes dies out in smaller time and new random chromosomes are produced by generator. This procedure enables directed random generation of the chromosomes such that the population convergence is in the way of optimum answer and the probability of the generator to start again would be reduced. Where the maximum number of population for a run of the algorithm is so low that it delays the first convergence, then the generator is more likely to behave in the way mentioned above.

2.4 Chromosome Structure

According what was mentioned, chromosome structure in this algorithm is as follows:

2.4.1 Alphabet String

Decimal or binary array, which forms the main body of the chromosome. The fitness value of each chromosome is calculated from its alphabet string.

2.4.2 BirthTime

BirthTime is a time in which a chromosome is generated by the generator or crossover operator.

2.4.3 LCT (Last Crossover Time)

LCT of a chromosome is the last time when it participates in the crossover operation. This parameter is used to avoid selecting same chromosomes for crossover. The initial value of this parameter for each chromosome is −1, which means that the chromosome has not participated in the crossover yet.

This parameter is also included in the parent chromosome selection process for the crossover operation.

2.4.4 Fitness

The fitness value of each chromosome which is calculated from the fitness function is similar to GA.

2.4.5 TTL

Equation (2) shows the TTL of *ith* chromosome in the moment of birth. Therefore, to calculate MaxLife of *ith* chromosome in every moment, we should subtract its past life (CurrentTime - BirthTime[i]) from its initial TTL value. Therefore, Eq. (2) can be rewritten in the form of Eq. (3).

$$\begin{aligned} TTL[i] = {} & \frac{(1-X)\times MaxLife}{|BestFitness - WorstFitness|} \\ & \times |Fitness[i] - WorstFitness| \\ & + X \times MaxLife - (CurrentTime - BirthTime[i]) \end{aligned} \quad (3)$$

Hence, the value of TTL for each chromosome depends on the obtained fitness range. Then the value of TTL is constantly updated during the algorithm execution.

Our proposed algorithm is capable of acting like GA or random search algorithm by adding a parameter called LifeStep to Eq. (3). Equation (4) shows how to apply this parameter.

$$\begin{aligned} TTL[i] = {} & \frac{(1-X)\times MaxLife}{|BestFitness - WorstFitness|} \\ & \times |Fitness[i] - WorstFitness| \\ & + X \times MaxLife \\ & - LifeStep \times (CurrentTime - BirthTime[i]) \end{aligned} \quad (4)$$

According to the Eq. (4), when this parameter is zero, TTL value of chromosomes will be constant during the execution time, so the size of population achieves its maximum value. In this case in order to reach GA functionality, a strategy should be taken to remove a number of chromosomes with bad fitness. If this parameter is considered to be ∞, the algorithm will act like random search algorithm. In this case, the TTL value of all chromosomes becomes negative. Therefore, the fitness of chromosomes are evaluated exactly at the moment of their birth, then they die. The value of LifeStep in the proposed algorithm is considered one as shown in Fig. 5.

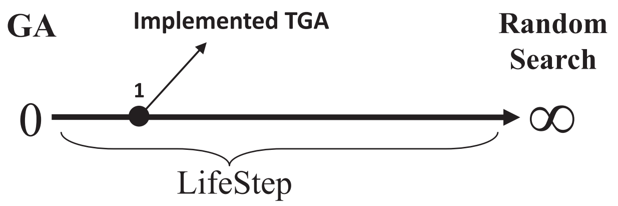


Fig. 5 The TGA algorithm acts like GA or random search algorithm by adjusting LifeStep parameter.

2.5 Crossover Operator

In the proposed algorithm, crossover operation is similar to GA, but the difference is that the crossover is done in each time unit with the probability of Pc. Thus, for each time unit a uniformly distributed random number between zero and one is generated. If the generated number is less than Pc, the crossover operation will occur.

Two concepts of “Percentage of young population” and “Population ratio” are declared to be used in the calculation of Pc parameter.

Pseudo Code 2 Calculation of crossover probability

```
if (ChromosomeCount < 2)
  Pc=0
else
  Pc=N ×A
```

2.5.1 Percentage of Young Population

Percentage of young population is denoted by A and calculated by the Eq. (5).

$$A = \frac{\sum TTL_{chromosomes}}{ChromosomeCount} \times \frac{1}{MaxLife} \quad (5)$$

In the first part of Eq. (5), the average age of chromosomes is calculated, which is certainly a number between zero and MaxLife. Then by division of this value to MaxLife, a number between zero and one is obtained which is Percentage of young population.

The closer the value of A is to zero, the older is the population. The closer this value is to one, the younger is the population, i.e. the average TTL of population become closer to zero or MaxLife.

2.5.2 Population Ratio

Population ratio is denoted by N and calculated by the Eq. (6).

$$N = \sqrt{1 - \left(\frac{ChromosomeCount}{MaxChromosome}\right)^2} \quad (6)$$

As the Eq. (6) shows, the value of N is between zero

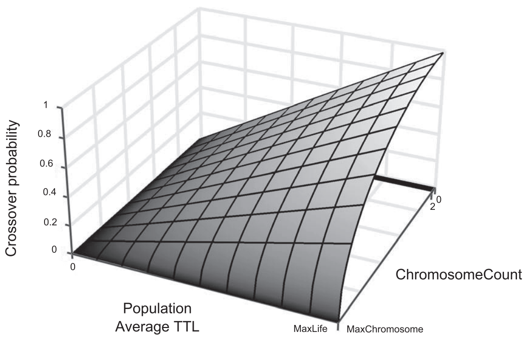


Fig. 6 Crossover probability as a function of population size and average of chromosome's TTL.

and one. Initially, when the ChromosomeCount is zero, the value of N is one. As the algorithm execution advances and the population size gets closer to the MaxChromosome, then the value of N tends to zero.

Of course, any other equation which controls the population size can be used instead of Eq. (6).

2.5.3 Crossover Probability

Crossover probability (Pc) depends on Percentage of young population (A) and Population ratio (N). According to these two parameters, the probability of crossover in each time unit will be as shown in Pseudo code 2:

Since the crossover operation requires at least two chromosomes, if the number of chromosomes is less than two, then the crossover probability becomes zero (the first section of Pseudo code 2). Otherwise, crossover probability is calculated based on the values of A and N.

Figure 6 illustrates the crossover probability as a function of population size and average of chromosome TTL which is based on Eqs. (5) and (6).

2.6 Mutation Operator

Genetic similarity of parents increase the probability of mutation. The mutation operator is applied to a chromosome exactly the same as GA. Only the offspring chromosomes may be mutated and mutation probability has a direct relationship with parent similarity. The parent similarity is defined as the ratio of the number of similar genes to the total number of genes in their alphabet string. Thus, the parent similarity is calculated after the crossover operation, and the result is used as the probability of mutation on the offspring chromosomes. Therefore, the more similar the chromosomes participating in crossover operation are, the higher is the probability of mutation on offspring chromosomes.

This type of mutation decreases the percentage of barren mutations which do not have any effect on the improvement of the answer.

2.7 Selection Method

Any selection algorithm can be used to select chromosomes as parents for crossover operation. However, unlike the GA, in the proposed algorithm the fitness value is not the parameter which determines the selection probability of the chromosomes. In the proposed algorithm, the chance of chromosomes to be selected as a parent in the crossover operation has a relationship with the TTL and LCT parameters. The selection probability is calculated in Pseudo code 3.

Pseudo Code 3 Calculation of the chance of chromosomes to be selected as a parent for crossover operation

```
if LCT(Chromosome[i]) = CurrentTime
  Selection Probability(Chromosome[i]) = 0
if LCT(Chromosome[i]) = -1
  Selection Probability(Chromosome[i]) = TTL(Chromosome[i])+ (CurrentTime -BrithTime(Chromosome[i]))
else
  Selection Probability(Chromosome[i]) = TTL(Chromosome[i]) + (CurrentTime - LCT(Chromosome[i]))
```

When a chromosome is selected for the crossover operation, its LCT value changes to current time value. Hence, the first section of Pseudo code 3 prevents the parent chromosomes to be the same. According to the second section of Pseudo code 3, the greater is the chromosomes TTL and the more time lapses from the last crossover time, the higher is chance of chromosome for selection as a parent in the crossover operation.

Now by using a regular selection algorithms such as Roulette Wheel Selection, Rank Selection and etc. and using the above parameters, the parents for crossover operation can be selected.

2.8 The Procedure of Algorithm in a Time Unit

The generator unit starts up the time of algorithm by generating the first chromosome.

In each time interval, algorithm procedure begins by checking current population size. If conditions of the generator are satisfied, a new chromosome would be generated (Sect. 1 of Fig. 7).

The next step of each time interval is the crossover operation. If the population size is less than two (Sect. 2 of Fig. 7), the algorithm enters the interval finishing step. Otherwise, if the population size is greater than two, the crossover operation will be performed with the probability of Pc. But, if the crossover operation does not occur, the algorithm goes to the interval finishing step. The selection method used for choosing the parents of the crossover operation has been explained in Sect. 2.7.

The time interval is continued by mutation operation. For muting an offspring, a uniformly distributed random number between zero and one is generated. If the generated number is greater than parent similarity percentage of that chromosome, the mutation will occur and the mutated chromosome will be added to the population; otherwise, the offspring will be added directly to the population.

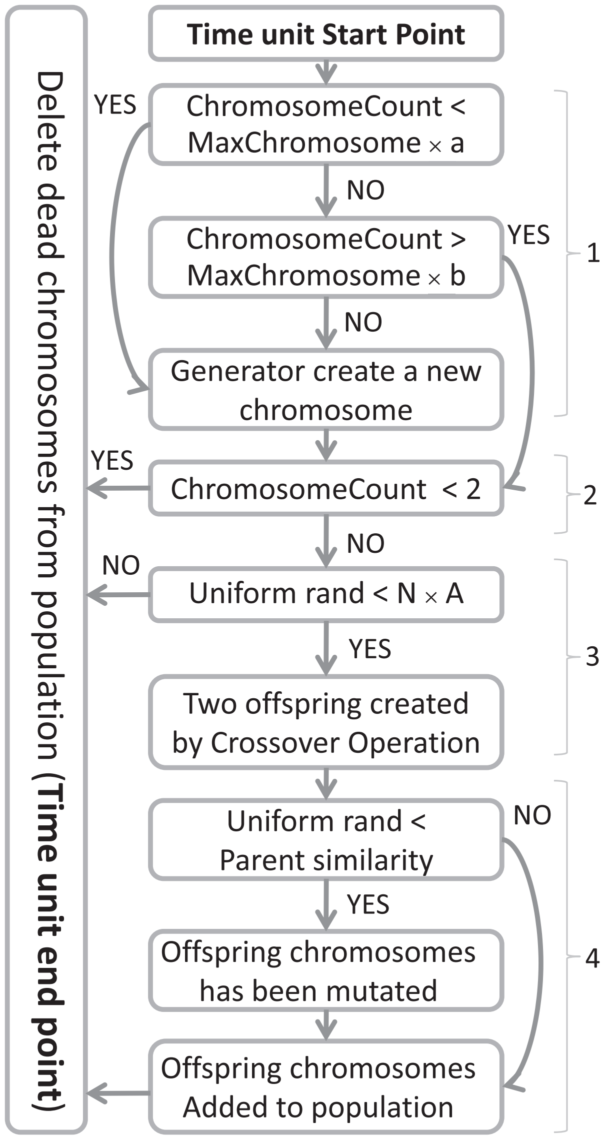

Time unit Start Point
ChromosomeCount < MaxChromosome × a
YES
NO
ChromosomeCount > MaxChromosome × b
YES
NO
Generator create a new chromosome
ChromosomeCount < 2
YES
NO
Uniform rand < N × A
NO
YES
Two offspring created by Crossover Operation
Uniform rand < Parent similarity
NO
YES
Offspring chromosomes has been mutated
Offspring chromosomes Added to population
Delete dead chromosomes from population (Time unit end point)
1
2
3
4

Fig. 7 Algorithm flowchart in a time unit.

Each time interval is finished with the interval finishing step. At the interval finishing step, the algorithm removes dead chromosomes (chromosomes with zero or negative TTL). Then the algorithm procedure in this time unit ends, and the algorithm enters the next time unit.

3. Experiments and Results

3.1 TGA vs. GAVaPS

In the GAVaPS algorithm, a parameter called lifetime is assigned to each chromosome which expresses its life time. At the moment of generating a chromosome, a fixed value is assigned to this parameter which is based on the current state of the algorithm and average fitness of population. The goal of this parameter is to vary the size of population, so if the difference of best and average fitness of population gets higher, the population size increases which enable the algorithm to search larger state spaces. When the algorithm best fitness gets closer to the optimal answer, the population size decreases to save the computing resources.

This method of lifetime allocation could cause some problems. Previously allocated lifetime might wrongly affect the forthcoming state of algorithm, which has different conditions and average fitness. In the problems in which almost all the chromosomes have the same value of fitness with different alphabet string, the allocation of lifetime on basis of the average fitness of total population will make the algorithm such a non-efficient and computation-wasting algorithm that it may face premature convergence. The algorithm will reduce the size of population because of determination of the sufficient convergence in the population, for the average fitness and best fitness values are close to each other. Therefore, the answer becomes difficult to achieve. In other words, the potential of GAVaPS algorithm for stagnation will be higher.

In the proposed TGA (Timed-GA) algorithm, a parameter called TimeToLive is considered for each chromosome. The TimeToLive parameter for a chromosome is a time duration in which it could participate in the process of the algorithm. Its value has a direct relationship with the fitness of the chromosome and inverse relationship with the difference of the worst fitness and best fitness of the whole population. The value of this parameter will change for all chromosomes whenever a new optimal answer is found in the algorithm. The change of worst fitness will also change the parameter value.

The goal of this parameter is to make an ideal convergence while keeping the progress of the algorithm away from stagnation. Thus, after sufficient convergence in the population, the population will have a dynamic average fitness with relatively constant variance in the process of algorithm. This will keep the dynamism of algorithm while maintaining its convergence sufficiently and stably.

Moreover, the algorithm search for the optimum answer without facing any premature convergence and stagnation by using fixed computational resources and memory, which is one of the features of non-systematic search algorithms. Hence, a reasonable algorithm is achieved which uses GA operators to speed up reaching the optimal answer while using random search algorithm feature which maintains the possibility of reaching better answers by continuation of algorithm execution.

The amount of population in the proposed algorithm can not be more than a certain value, and whenever the difference between best fitness and the worst fitness becomes higher, then the destruction of the defected chromosomes will speed up automatically, so the algorithm survives any chromosome production which converges the algorithm in the wrong way, i.e. the algorithm converges to local optimum instead of global optimum. The sudden destruction of defected chromosomes may result in population decline, which will be compensated by restarting the generator unit.

3.2 Problems

Test problems used for comparing proposed TGA algorithm with other algorithms contain different types of examples which behave differently in optimization problems. Each problem has been executed for twenty times and they terminate if there is no progress in terms of the best value found for 10,000 consecutive time units.

There are eight test problems. The first five problems are maximization problems and the rest are minimization problems. In addition to SGA (Simple Genetic Algorithm), The results of the first four problems are compared with GAVaPS [8]. These three algorithms (TGA, SGA, GAVaPS) use the same encoding method. Fixed-point method is used for storing a decimal number as a binary-encoded and the length of chromosomes are twenty.

For the SGA results, the same method is applied. The different is that when the algorithm is fully converged, its execution stops. In the result tables, **SGA (1)** shows the **average** results of SGA for 20 runs. For the first four problems, the results are obtained from [8], and for the remaining problems, the results are obtained from the self-implemented SGA using the same approach. **SGA (2)** indicates the **best** result of SGA run through 20 runs for that problem obtained from self-implemented SGA. For legibility of Time-Fitness diagrams of TGA, results have been truncated until the algorithm reaches the best fitness. It should be mentioned that for the problems in which the SGA results do not exist in [8], the results are obtained from a similar implementation of those problems.

It should be mentioned that only the optimum answers of all algorithms have been compared. However, the time need to achieve the optimal answer in the TGA cannot be compared with the generation numbers in the SGA, because the number of chromosomes which have been processed in each time unit of TGA is less than that of each generation in other genetic algorithms.

3.2.1 Finding the Maximum Value of G1 Function

In this problem, a one-dimensional function with a large number of local maximum is studied, which is shown in Fig. 8.

During maximizing such a function, two directions of growth can easily be recognized, but the boundaries are chosen in such a way that a global maximum is obtained for only one of them.

According to the length of binary chromosomes (twenty bit), the value of possible optimal answer is 2.95032.

$$G_1(x) = -x\sin(10\pi x) + 1 \qquad -2.0 \leq x \leq 1.0 \tag{7}$$

Table 1 and Fig. 9 show the results of maximizing Eq. (7).

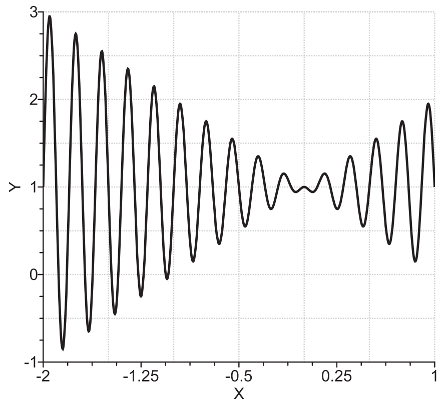
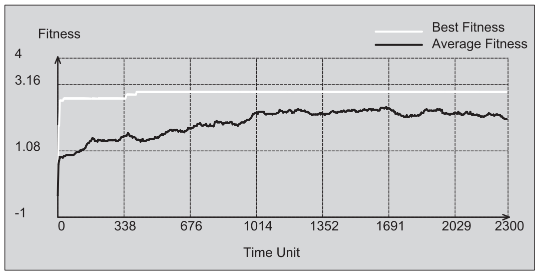


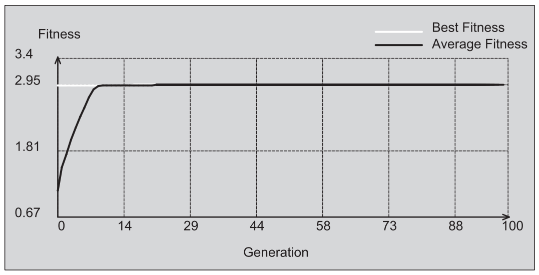
Fig. 8 Diagram of G1 function.

Table 1 Maximization results for G1 function.

Approach	Best Fitness
TGA	2.95032
SGA(1)	2.814
SGA(2)	2.95030
GAVaPS	2.841



(a)



(b)

Fig. 9 Algorithm run for problem 3.2.1 (a) TGA (b) SGA.

3.2.2 Finding the Maximum Value of G2 Function

Function G2 cannot be optimized by means of any gradient technique, since there is no gradient information available. In this problem, population has been converged immediately, so GAVaPS algorithm reduces the population size quickly instead of trying to search another part of state space, and the continuation of search become ineffective.

The SGA converges more quickly for this problem too and continuation of the algorithm execution cannot improve the answer.

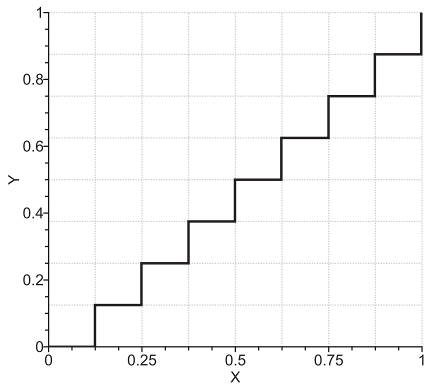


Fig. 10 Diagram of G2 function.

Table 2 Maximization results for G2 function.

Approach	Best Fitness
TGA	1
SGA(1)	0.875
SGA(2)	0.875
GAVaPS	0.875

The TGA has shown its efficiency to solve this problem, since it is far from stagnation. It can find the optimum answer in all its twenty executions.

Another feature of this problem is that, the G2 function has only one maximum point in the given boundaries. The value of this maximum point has a big difference from the value of other points. Therefore, there is only one chromosome with the best fitness, which in terms of alphabet string has a big difference from the other chromosomes.

Considering Fig. 10, the value of function G2 in the interval [0.875, 1) is 0.875, so all chromosomes have the same fitness values in this interval; then the algorithm should search all the state space to reach the optimal answer. This feature is satisfied by TGA as well.

$$G_2(x) = \frac{[8x]}{8} \qquad 0.0 \le x \le 1.0 \tag{8}$$

Table 2 and Fig. 11 show the results of maximizing Eq. (8).

3.2.3 Finding the Maximum Value of G3 Function

The G3 function as illustrated in Fig. 12, is another deceptive problem like G1 function but a nonlinear one.

$$G_3(x) = |x| \qquad -1.0 \le x \le 2.0 \tag{9}$$

Table 3 and Fig. 13 show the results of maximizing Eq. (9).

3.2.4 Finding the Maximum Value of G4 Function

This function as shown in Fig. 14, has many local maximum and minimum points close to each other like G1 function and in the neighborhood of global minimum and maximum points, the distance between local maximum and minimum points becomes smaller.

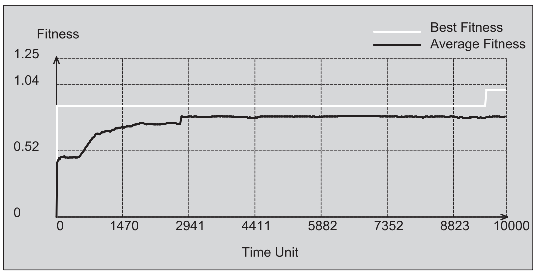
$$G_4(x) = 0.5 + \frac{\sin(\sqrt{x^2+y^2})^2}{(1+\frac{x^2+y^2}{1000})^2} \quad -100.0 \le x \le 100.0 \tag{10}$$

Table 4 and Fig. 15 show the results of maximizing Eq. (10).

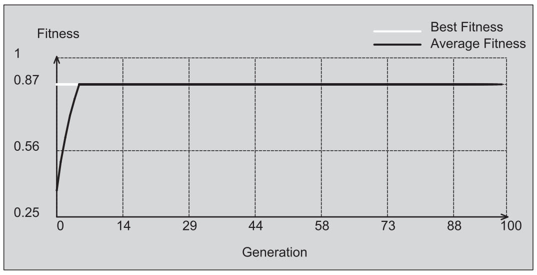
3.2.5 The Extended Knapsack Problem

The classic knapsack problem is extended to become a multi-object optimization problem with two limitations. In order to reach this approach, there is a set of *n* items with definite weight and value.

The goal is to choose a subset of these items in which



(a)



(b)

Fig. 11 Algorithm run for problem 3.2.2 (a) TGA (b) SGA.

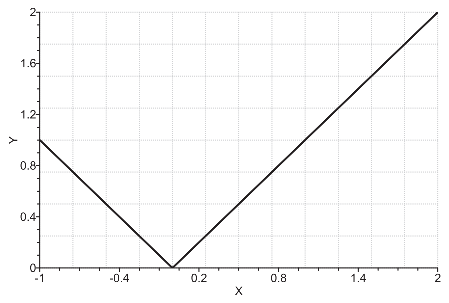


Fig. 12 Diagram of G3 function.

Table 3 Maximization results for G3 function.

Approach	Best Fitness
TGA	2
SGA(1)	1.992
SGA(2)	2
GAVaPS	1.999

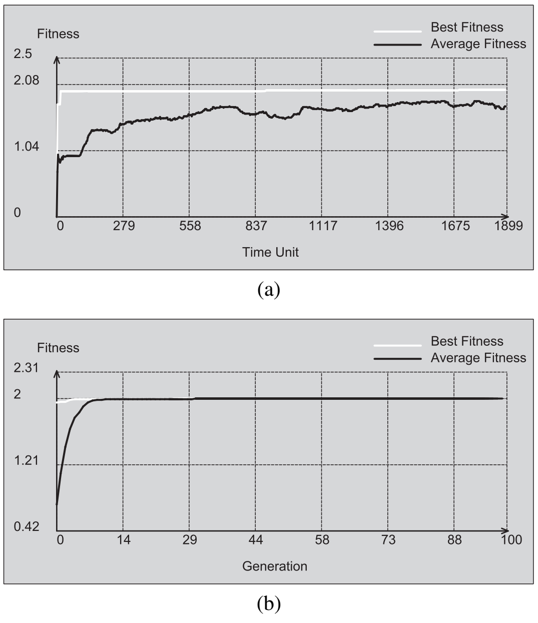


Fig. 13 Algorithm run for problem 3.2.3 (a) TGA (b) SGA.

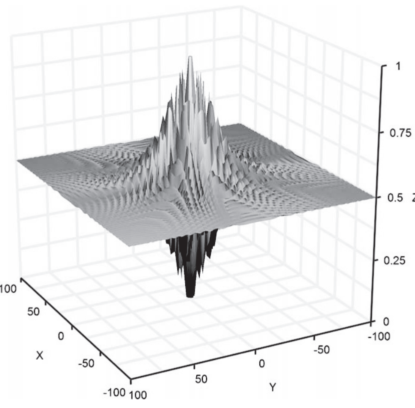


Fig. 14 Diagram of G4 function.

Table 4 Maximization results for G4 function.

Approach	Best Fitness
TGA	0.996987
SGA(1)	0.959
SGA(2)	0.93751
GAVaPS	0.972

their total weight is minimized and their total value is maximized. Moreover, their total weight should not violate a certain W_{max} value, while their total value should be more than a certain V_{min} value. Thus, Eq. (11) is the accumulative fitness function for this problem, which should be maximized. In this equation *list* is selected items set, α is coefficient of value parameter, β is coefficient of weight parameter and m is size of selected items set.

In the following a knapsack problem for 20 proposed items is simulated. This example presents a combinational problem, which have 2^{20} states. Table 5 shows the set of items with their value and weight. According to the aver-

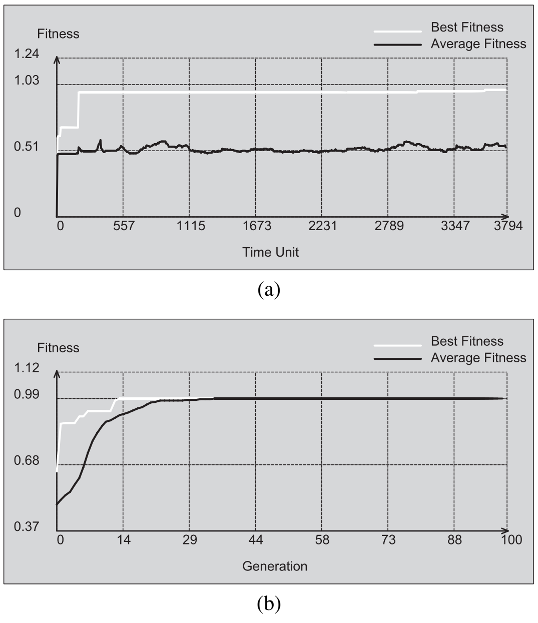


Fig. 15 Algorithm run for problem 3.2.4 (a) TGA (b) SGA.

Table 5 Set of items with their value and weight.

Item	value	weight	Item	value	weight
N1	1	5	N11	7	51
N2	4	9	N12	16	29
N3	3	7	N13	31	1
N4	2	7	N14	24	63
N5	9	15	N15	22	19
N6	12	21	N16	9	1
N7	7	11	N17	7	74
N8	8	4	N18	13	65
N9	41	2	N19	41	23
N10	16	42	N20	63	23

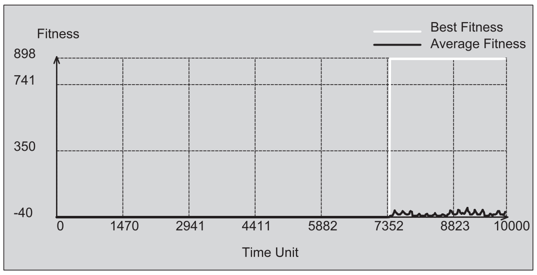
age weight of items (23.6), and the weight limitation of 70 (W_{max} = 70), approximately 3 items could be chosen on average. On the other hand, according to the average value of items (16.8), and avoiding the violation of value limitation of 200 (V_{min} = 200), approximately 12 items should be chosen, but the optimal answer which could satisfy these two limitations contains 6 items, while most of the other states fail to satisfy the limitation, so they are given penalty. The failure of SGA in finding the optimal answer proves that this problem could be considered as a problem with hard limitations. Table 6 and Fig. 16 shows the results of maximizing Eq. (11). In this simulation α and β are chosen 5 and 2 respectively.

$$F(list) = \alpha \sum_{i=1}^{m} value(i) - \beta \sum_{i=1}^{m} weight(i) - P_V - P_W \quad (11)$$

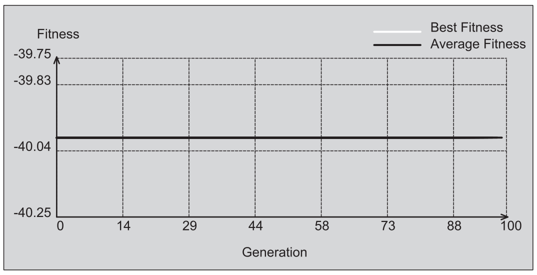
$$P_V = \begin{cases} 0 & \sum_{i=1}^{m} value(i) > V_{min} \\ V_{min} - \sum_{i=1}^{m} value(i) & otherwise \end{cases}$$

Table 6 Maximization results for the extended knapsack problem.

method	best fitness	best chromosome	value	weight
TGA	897	00000000100010110011	207	69
SGA(1)	-40	10011001110011111110	224	321
SGA(2)	-40	10011001110011111110	224	321



(a)



(b)

Fig. 16 Algorithm run for problem 3.2.5 (a) TGA (b) SGA.

$$P_W = \begin{cases} 0 & \sum_{i=1}^{m} weight(i) < W_{max} \\ W_{max} - \sum_{i=1}^{m} weight(i) & otherwise \end{cases}$$

3.2.6 Word Guess [4]

It is a simple-word-guess problem which the algorithm is given the number of letters in a word, and it guesses the letters that composes the word until it finds the right answer. In this case, we will use a GA where each letter is given the integer corresponding to its location in the alphabet (a = 1, b = 2, etc.). Suppose that the GA must find unknown word "Colorado". The fitness function is given by Eq. (12) that calculate the sum of absolute differences between the codes of *str* and *pass*, in which the *str* is guessed word by GA and *pass* is the word "Colorado".

$$F(str) = \sum_{i=1}^{8} |str[i] - pass[i]| \qquad (12)$$

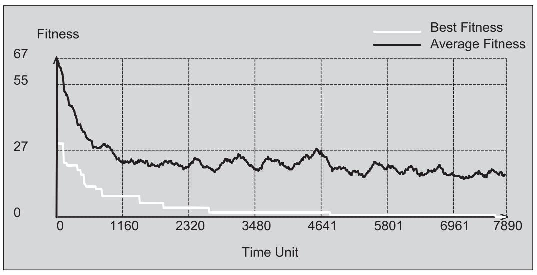
Table 7 and Fig. 17 provide the results of minimizing Eq. (12).

3.2.7 Travelling Salesman Problem [4]

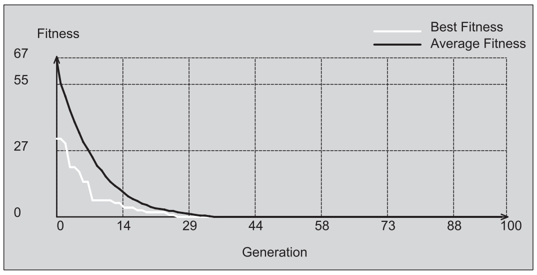
The travelling salesman problem is a permutation problem in which the goal is to find the shortest path between N different cities that the salesman takes which is called a tour. In other words, the problem deals with finding a route covering all the cities so that the total distance traveled is minimal.

Table 7 Minimization results for word guess problem.

method	best fitness	best chromosome
TGA	0	2.14.11.14.17.0.3.14
SGA(1)	0	2.14.11.14.17.0.3.14
SGA(2)	0	2.14.11.14.17.0.3.14



(a)



(b)

Fig. 17 Algorithm run for problem 3.2.6 (a) TGA (b) SGA.

Table 8 Cities distance from each other.

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	–	10	3	2	5	6	7	2	5	4
c2	20	–	3	5	10	2	8	1	15	6
c3	10	5	–	7	8	3	11	12	3	2
c4	1	2	3	–	5	6	7	8	9	10
c5	1	2	3	4	–	5	10	20	11	2
c6	8	5	3	10	2	–	6	9	20	1
c7	3	8	5	2	20	21	–	3	5	6
c8	5	2	1	25	15	10	6	–	8	1
c9	10	11	6	8	3	4	2	15	–	1
c10	5	10	6	4	15	1	3	5	2	–

For this problem, ten cities have been assumed, which all are connected to each other. Distances between the cities have been defined in Table 8.

The purpose of this algorithm is to find the minimum distance which the salesman should go to cover the whole cities and each city must be visited once. Thus, the purpose is to minimize the Eq. (13), in which the *path* is the order cites covered by the salesman.

Table 9 Minimization results for travelling salesman problem.

method	best fitness	best chromosome
TGA	19	4-9-7-8-2-3-10-6-5-1
SGA(1)	21	5-2-8-3-10-7-1-4-9-6
SGA(2)	21	8-3-10 -9-7-1-4-6-5-2

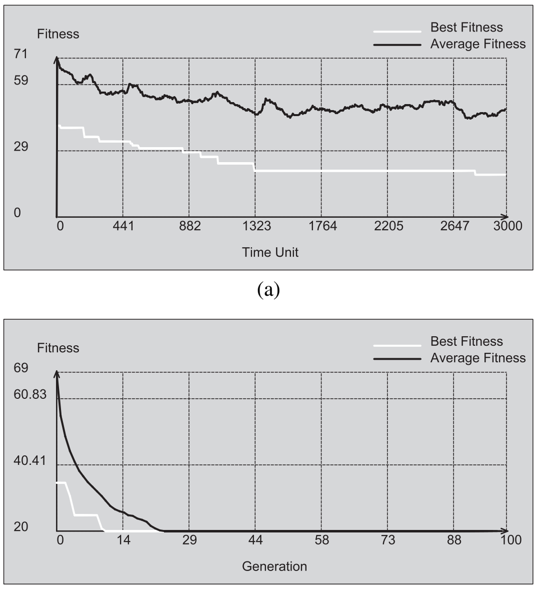


Fig. 18 Algorithm run for problem 3.2.7 (a) TGA (b) SGA.

Table 10 Minimization results for travelling salesman problem using 100 cities.

method	best fitness
TGA	370
SGA(1)	404
SGA(2)	472

$$F(path) = \sum_{i=1}^{9}(distance(path[i], path[i+1])) + distance(path[10], path[1]) \quad (13)$$

Table 9 and Fig. 18 show the results of minimizing Eq. (13).

To study the operation of the algorithm in problems with larger state space, the problem of the travelling salesman using 100 cities and an average distance of 20 between them was examined. The results are shown in Table 10 and Fig. 19. As the results show, TGA has a better performance and yield better results than SGA.

3.2.8 Wireless ATM Network [17]

For this Problem, consider a group of cells and a group of switches in an ATM network (whose locations are fixed and known). The problem is to assign cells to switches in the ATM network in an optimum manner. Topological design of a two-level hierarchical network has been considered. The upper-level network is a connected ATM network, and the lower-level network is a PCS network which is configured as an H-mesh (Fig. 20). The assumptions of the problem are stated as follows:

1) The structures and positions of the ATM network and cell network are known.

2) Each cell in the cell network will be directly assigned and connected to only one switch in the ATM network.

3) The number of calls that can be handled by each cell per unit time is equal to 1.

4) The capacity of a switch, the number of cells that it can be assigned, is limited to a constant called Cap.

5) The cost has two components. One is the cost of handoffs that involve two switches, and the other is the cost of cabling (or trucking).

6) Minimal switches assumption: the number of switches assigned is assumed to be minimized.

7) Load balance assumption: The load of assigned

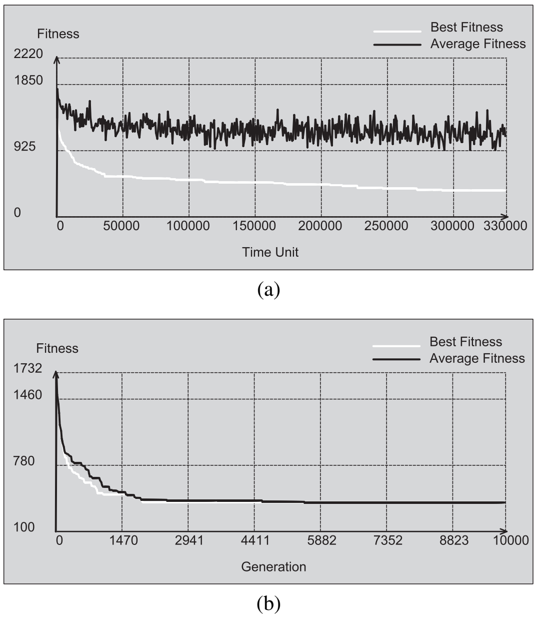


Fig. 19 Algorithm run for problem 3.2.7 using 100 cities (a) TGA (b) SGA.

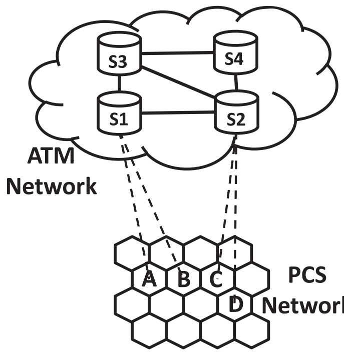


Fig. 20 Two-level hierarchical network.

switches is assumed to be balanced. If this load balance assumption is satisfied, $\acute{m}$ = [n/Cap] switches need to be assigned, and the number of cells assigned to switches is [n/$\acute{m}$].

The various notations used here are:
n: total number of cells in the cell network
m: total number of switches in the ATM network
L_{ik}: cost of cabling per unit time and between cell c_i and switch s_k
F_{ij}: cost per unit time of the handoffs that occur between cell c_i and c_j
W_{ij}: weight of edge (c_i, c_j), where
$W_{ij} = F_{ij} + F_{ji}$, $W_{ij} = W_{ji}$, and $W_{ii} = 0$
D_{ki}: minimal cost between switches s_k and s_i
C_x:number of cells that are connected to the switch x
Cap: cell handling capacity of the switch
$\acute{m} = \lceil n/Cap \rceil$: number of switches that need to be assigned
α: ratio of the cost of cabling to that of handoff
Thus the objective is to minimize the Eq. (14).

$$\begin{aligned} T(v) &= \sum_{i=1}^{n} L_{i,v(i)} + \alpha \sum_{i=1}^{n}\sum_{j=1}^{n} W_{i,j} \times D_{v(i),v(j)} \\ P &= \beta(\sum_{k=1}^{\acute{m}} |C_k - Cap| + \sum_{k=\acute{m}+1}^{m} |C_k|) \\ F(v) &= T(v) + P \end{aligned} \quad (14)$$

Since our problem is the way of connecting cells and switches, cells are labeled from one to n (the total number of cells); and switches are labeled from one to m (the total number of switches). The cell-oriented representation of the chromosome structure express that the i_{th} cell belongs to the *v(i)* switch. P is the penalty measure associated with a chromosome, and β is the penalty weight. Proposed values for parameters are as follows:

The area is divided into fifty cells (n = 50), including eight switches (m = 8) with the capacity of ten for each switch (Cap = 10).

Table 11 and Fig. 21 show the results of minimizing Eq. (14).

Table 11 Minimization results for wireless ATM problem.

method	best fitness
TGA	91
SGA(1)	91
SGA(2)	91

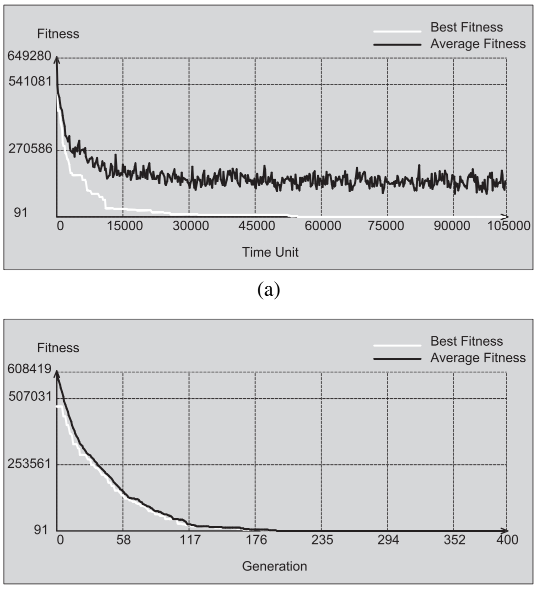


Fig. 21 Algorithm run for problem 3.2.8 (a) TGA (b) SGA.

3.3 Parameters Effect

In this section, the effect of parameters on the algorithm functioning is studied. The problem of finding the maximum value of G4 function as stated in Sect. 3.2.4, with varying values of its input parameters is solved by the proposed algorithm (TGA). The results are shown in Table 12. The first row in Table 12 shows proper initial values for the parameters. The rest of rows in Table 12 show the behavior of the algorithm with changing a parameter while keeping the others unchanged.

- As shown in Sect. 2.2 and Fig. 2, MaxLife is the highest

Table 12 Effect of algorithm parameters changes.

Row	Description	MaxLife	LifeStep	a	b	X	Best fitness	Best fitness achievement time
1	Proper Initialization	100	1	0.25	0.5	0.25	0.996987	3490
2	Smaller MaxLife	50	1	0.25	0.5	0.25	0.996987	19413
3	Bigger MaxLife	200	1	0.25	0.5	0.25	0.996987	8859
4	Smallest X	100	1	0.25	0.5	0	0.996987	8700
5	Bigger X	100	1	0.25	0.5	0.8	0.996987	13775
6	Lower Range of a, b	100	1	0.1	0.3	0.25	0.996987	19915
7	Higher Range of a, b	100	1	0.7	0.9	0.25	0.996987	26850
8	Wider Range of a, b	100	1	0.15	0.7	0.25	0.996987	6483
9	Narrower Range of a, b	100	1	0.45	0.5	0.25	0.996987	25252
10	Smaller LifeStep	100	0.1	0.25	0.5	0.25	0.996987	6224
11	Bigger LifeStep	100	10	0.25	0.5	0.25	0.996987	30806

TTL value that a chromosome can possibly obtain. As the value of this parameter increases, so does the life of the chromosomes, and this enables the chromosomes to participate in crossover and mutation operations for longer periods of time. When MaxLife has a low value, the chromosomes will have a shorter life, acting only fewer times in the crossover and mutation operations. The effect of MaxLife parameter in solving the problem 3.2.4 is explained in rows 2 and 3 of Table 12. It should be noted that inappropriate value assignment to MaxLife will only reduce the speed of algorithm.

- As shown in Sect. 2.2 and Fig. 2, the value of parameter X represents the maximum chance of the worst chromosome participating in the algorithm process. The closer its value to zero, the lower TTL allocated to bad chromosomes (chromosomes with fitness values close to the worst fitness); consequently, they will have less participation in the algorithm process. On the other hand, as this value approximates 1, TTL of chromosomes with bad fitness values will be closer to the TTL of chromosomes with better fitness values; and their participation will be more likewise. Rows 4 and 5 in Table 12 show the effect of this parameter on the solution of the problem 3.2.4 by TGA.
- According to the Pseudo code 1 and Fig. 3, a change in the distance between a and b affects the generator operation period. The effect is such that as the distance between a and b increases, the generator operates for longer periods, producing more and more random chromosomes. On the other hand as this distance decreases, the efficiency of the generator will decrease and fewer random chromosomes are produced. As a result, the effect of crossover and mutation operators will be more prominent in the execution of the algorithm. the effect of a and b parameter values is shown in rows 6 through 9 of Table 12.
- The effect of LifeStep parameter is addressed in Sect. 2.4.5. As explained in this section and from the Eq. (4) and Fig. (5), it can be said that a change in the value of this parameter, ranging from zero to infinity, causes a change in algorithm behavior from a genetic algorithm to a random search one. Row 10 and 11 of Table 12 explain the effect of this parameter on the solution of problem 3.2.4 by TGA.

3.4 Discussion

In the all applications presented in this paper, the TGA reaches the optimal answer with only one run, unlike other genetic algorithms which should be run several times in order to ensure the optimal answer. Unlike the SGA results, the average Fitness curve in Time-Fitness diagram of the all problems for TGA, changes dynamically over the time unit and this guarantees that TGA never stagnates by the continuation of execution.

TGA appears to solve the problems with hard limitations better than other optimization algorithms. This is because TGA behaves with chromosomes located in the infeasible state space of problem much more efficiently. The problem 3.2.5 proves this as well. For the same problem shown in Fig. 16, the SGA algorithm stagnates just as the algorithm starts up.

Unlike other genetic algorithms, TGA guarantees not to stagnate by the continuation of execution, so we can be sure that it would search non-searched spaces of problem to find a better answer if there is any. The problem 3.2.2 proves this as well.

While searching state space of NP problems by non-systematic algorithms, where there is no estimation of the optimal answer, no one can claim finding the final optimum answer. When solving NP problems with SGA and similar algorithms, because the execution mostly ends with stagnation, we have to accept the results which are achieved by several runs. But in TGA, as we never face stagnation, we should decide on stop time of the algorithm, considering the computational resources, time constraint and estimation of problem answer.

4. Conclusion

In this paper, a new algorithm called TGA is introduced which for the first time defines the concept of time more naturally. Moreover, the mutation operator is used more meaningfully. According to the results of simulated problems, unlike other genetic algorithms, only one run of proposed algorithm is enough to reach the optimum answer. The results of TGA algorithm for multi-object optimization are much better than the results of SGA. For an unskilled user it is much easier to work with TGA because inaccurate initialization of parameters in TGA only affects the achievement time of optimal answer, but not the optimal answer itself.

It can be concluded that the proposed algorithm provides an intermediate solution between genetic and random search algorithm and uses main features of both such that the speed of reaching the optimal answer is comparable with genetic algorithm while avoiding premature convergence or stagnation like random search algorithm.

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.
[2] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Reading, Addison-Wesley, MA, 1989.
[3] R.S. Zebulum, M.A.C. Pacheco, and M.M.B.R. Vellasco, Evolutionary Electronics, Automatic Design of Electronic Circuits and Systems by Genetic Algorithms, CRC Press, 2002.
[4] R.L. Haupt and S.E. Haupt, Practical Genetic Algorithms, John Wiley & Sons, 2004.
[5] M. Last and S. Eyal, "A fuzzy-based lifetime extension of genetic algorithms," Fuzzy Sets and Systems, vol.149, pp.131–147, 2005.
[6] T. Back, A.E. Eiben, and N.A.L. van der Vaart, "An empirical study on GAs "without parameters"," Parallel Problem Solving from Nature, PPSN VI, pp.315–324, 2000.
[7] K. Deb and S. Agrawal, "Understanding interactions among genetic algorithm parameters," in Foundations of Genetic Algorithm

5, ed. W. Banzhaf and C. Reeves, pp.265–286, Morgan Kaufmann, San Francisco, CA, 1998.

[8] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS-a Genetic Algorithm with varying population size," Proc. 1st IEEE Conf. on Evolutionary Computation, pp.73–78, 1994.

[9] G.F. Minetti and H.A. Alfonso, "Variable size population in parallel evolutionary algorithms," Proc. 2005 5th International Conference on Intelligent Systems Design and Applications (ISDA'05), 2005.

[10] T. Hu and W. Banzhaf, "The role of population size in rate of evolution in genetic programming," EuroGP 2009, LNCS 5481, pp.85–96, 2009.

[11] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, and L.M. Wang, "An improved GA and a novel PSO-GA-based hybrid algorithm," Inf. Process. Lett., vol.93, pp.255–261, 2005.

[12] L. Lanzarini, C. Sanz, M. Naiouf, and F. Romero, "Mixed alternative in the assignment by classes vs. conventional methods for calculation of individuals lifetime in GAVaPS," Int. Conf. Information Technology Interfaces ITI 2000.

[13] A.E. Eiben, E. Marchiori, and V.A. Valko, "Evolutionary algorithms with on-the-fly population size adjustment," Parallel Problem Solving from Nature PPSN VIII, LNCS 3242, pp.41–50, Springer, 2004.

[14] F.G. Lobo and C.F. Lima, "Revisiting evolutionary algorithms with on-the-fly population size adjustment," GECCO'06, July 2006.

[15] G.R. Harik and F.G. Lobo, "A parameter-less genetic algorithm," Proc. Genetic and Evolutionary Computation Conference (GECCO 1999), pp.258–267, 1999.

[16] A. Roy, S. Pal, and M.K. Maiti, "A production inventory model with stock dependent demand incorporating learning and inflationary effect in a random planning horizon: A fuzzy genetic algorithm with varying population size approach," Comput. Ind. Eng., vol.57, pp.1324–1335, 2009.

[17] S.N. Sivanandam and S.N. Deepa, Introduction to Genetic Algorithms, Springer Berlin Heidelberg, 2009.

[18] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., Prentice Hall, 2002.

Amir Mehrafsa was born in 1984 in Tabriz, Iran. He received his B.Sc., degree in Computer Engineering from the University of Tabriz in 2010. His interests include algorithm design, artificial intelligence, computer vision and image processing. E-mail: a.mehrafsa89@ms.tabrizu.ac.ir

Alireza Sokhandan was born in 1988 in Tabriz, Iran. He received his B.Sc., degree in Computer Engineering from the University of Tabriz in 2010. His interests include algorithm design, artificial intelligence, computer vision and image processing. E-mail: ar.sokhandan89@ms.tabrizu.ac.ir

Ghader Karimian was born in 1975 in Khosrowshar, Iran. He received his B.Sc., M.Sc. and Ph.D. degrees in Electrical Engineering from the University of Tabriz, Amirkabir University of Technology and Amirkabir University of Technology in 1998, 2000, and 2005, respectively. He is a faculty member at Electrical Department of University of Tabriz since 2005. His interests include robot vision, target tracking, image processing, digital signal processing and Hardware Design.