Genet Program Evolvable Mach (2013) 14:395–427
DOI 10.1007/s10710-013-9185-x

A high performance genetic algorithm using bacterial conjugation operator (HPGA)

Amir Mehrafsa · Alireza Sokhandan · Ghader Karimian

Received: 5 December 2012 / Revised: 24 March 2013 / Published online: 18 April 2013

Abstract In this paper an efficient evolutionary algorithm is proposed which could be applied to real-time problems such as robotics applications. The only parameter of the proposed algorithm is the "Population Size" which makes the proposed algorithm similar to parameter-less algorithms, and the only operator applied during the algorithm execution is the bacterial conjugation operator, which makes using and implementation of the proposed algorithm much easier. The procedure of the bacterial conjugation operator used in this algorithm is different from operators of the same name previously used in other evolutionary algorithms such as the pseudo bacterial genetic algorithm or the microbial genetic algorithm. For a collection of 23 benchmark functions and some other well-known optimization problems, the experimental results show that the proposed algorithm has better performance when compared to particle swarm optimization and a simple genetic algorithm.

Keywords Evolutionary algorithm · Genetic algorithm · Bacterial conjugation · High performance · Real-time · Parameter less

1 Introduction

In recent decades, the genetic algorithm (GA) has been extensively applied to a large number of optimization problems with considerable success. Natural

A. Mehrafsa · A. Sokhandan
School of Engineering Emerging Technologies, University of Tabriz, Tabriz, Iran
e-mail: a.mehrafsa89@ms.tabrizu.ac.ir

A. Sokhandan
e-mail: ar.sokhandan89@ms.tabrizu.ac.ir

G. Karimian (✉)
Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran
e-mail: karimian@tabrizu.ac.ir

biological operators are the key inspiration for many operators of evolutionary algorithms (EAs) such as chromosome crossover and mutation. Most research related to this issue focus on developing adaptive behaviors for these operators [1, 2]. Others focus on obtaining an optimum answer by not letting the algorithms lead to premature convergence or stagnation [3]. However, the evolutionary nature of such algorithms has always posed problems. For example, most of the variations of EAs can not be applied to problems which need a real-time response. The other major problem is that most of them are introduced with many open parameters, which make their initialization and implementation a professional field, which necessitates a separate analysis for each type of problem. Even for a simple genetic algorithm (SGA) with three parameters—the population size, the rate of crossover, and mutation rate—you could find various implementations for a definite problem like the classic travelling salesman problem (TSP). These algorithms could be implemented in many different ways due to their open parameters. In addition, finding the optimum parameters for each problem is a time consuming and exhausting procedure. For example, in SGA keeping the population size constant—according to the sensitivity of the problem to the parameters change—different initializations could be considered for the other two parameters, and this will lead to a time consuming search in parameters space.

This paper proposes an efficient evolutionary algorithm called HPGA (a high performance genetic algorithm using bacterial conjugation operator). HPGA uses another genetic mechanism as an optimization operator called bacterial conjugation that is the direct transfer of genes between bacterial cells. Bacterial conjugation models a transfer procedure of the genetic material between bacterial cells that are in direct cell-to-cell connection. This operator is classified as one of the horizontal gene transfer mechanisms [4]. This operator is used in the proposed algorithm in a way that speeds up the evolutionary mechanism.

There are other algorithms [5–7] that introduce this mechanism as their major operator. The microbial genetic algorithm stripped down the conventional genetic algorithm to its basics, representing the horizontal gene transfer operator inspired by natural bacterial conjugation. The pseudo bacterial genetic algorithm (PBGA) incorporates a modified mutation operator called bacterial mutation that is based on the biological phenomenon of microbial evolution [8]. The bacterial evolutionary algorithm (BEA) has the same features of the PBGA, but introduces a new operation called gene transfer operation, which is equally inspired by a microbial evolution phenomenon [9]. In fact, BEA incorporates two operations to evolve its population, the bacterial mutation and the gene transfer operation. While the bacterial mutation behaves as a local improvement within the limits of a single chromosome, the gene transfer operation allows the chromosomes to transfer information directly to the other chromosomes in the population. Using this mechanism, one bacterium can rapidly spread its genetic information to other cells [8]. Although BEA is inspired by gene transfer operations, its implementation method suffers from an excessively large number of parameters and this increases its complexity. In addition, it appears to be a memory intensive algorithm, which has N_{inf} steps of sorting for each generation, where N_{inf} is the number of infection per generation [8], and this makes it useless for real-time optimization problems.

As mentioned earlier, the proposed algorithm is inspired by bacterial conjugation. In this mechanism, the genetic information of the donor cell proves to be beneficial to the recipient cell, which is inspired by the proposed method for transferring genetic information of donor cell to the recipient cell. Thus, an individual with the best fitness is considered as a donor cell for the whole population here.

This paper represents an evolutionary method which reduces the complexity of the implementation with less use of computational resources during execution. The most important advantage of the implemented method of the bacterial conjugation operator in the proposed algorithm versus other implemented methods is its performance without needing to configure any initial parameters, which makes the whole algorithm appropriate to use in real-time applications since there will be no wasted time finding the optimum and proper parameters for a problem. Moreover, this method of implantation prevents premature convergence, yielding more accurate results.

In the proposed method, the biologically inspired mechanisms of old-fashioned GAs are removed and the bacterial conjugation is the only operator. It should be emphasized that this operator does not need any other parameters when applied to various problems; therefore, the population size is the only parameter of the proposed algorithm. Unlike BEA or some derivations of GA, there is no sorting step; as a result, the algorithm is much faster and its implementations is much simpler. The proposed algorithm and the mentioned ones have the same inspiration, but they are implemented differently.

The procedures of the proposed algorithm are simple and low cost. The experimental results show that the proposed algorithm has better performance and accuracy than Particle Swarm Optimization (PSO) and SGA, when the parameters of the mentioned algorithms are initialized with their common values. Moreover, the proposed algorithm has similar performance to the mentioned algorithms when their parameters are initialized with near optimum values. Considering the fact that finding the optimum values for the competitor algorithms requires a lot of computational resources and time, the performance of the proposed algorithm is acceptable.

This paper is organized as follows: Sect. 2.1 describes the bacterial conjugation operator in the proposed algorithm. Section 2.2 expresses the procedure of the whole algorithm, and the Sect. 2.3 clarifies the advantages of the proposed method. Section 3 presents the experimental results of sample problems, and the final section is the conclusions.

2 The proposed algorithm

In this section, the main operator of the proposed algorithm is addressed and the procedure is explained. Next, the main features of the proposed algorithm are reviewed.

2.1 The bacterial conjugation operator

The bacterial conjugation (BC) operator is divided into two main steps.

- Horizontal gene transfer
- Competition

The horizontal gene transfer mechanism is applied to two parent chromosomes. The chromosome with a better fitness is the donor chromosome and the one with the worse fitness is the recipient. The values of best and worst fitness during the algorithm execution are known. After the gene transfer step, the recipient chromosome and the new generated chromosome from first step are entered into competition step. The resulting chromosome is the BC operator output.

In view of this, the input arguments of the BC operator are:

- Donor chromosome (CH_{Donor})
- Recipient chromosome ($CH_{Recipient}$)
- The best ever fitness obtained ($Fitness_{Best}$)
- The worst ever fitness obtained ($Fitness_{Worst}$)

Figure 1 shows the whole procedure of the BC operator, and Fig. 2 shows the application of this operator on two sample chromosomes.

2.1.1 Horizontal gene transfer

In the first step of the BC operator, a string of genes is selected from the donor chromosome and placed in the recipient chromosome in the same location. The start point of the gene string of the donor chromosome is selected randomly. Its length is proportional to the ratio of the fitness difference between parent chromosomes with that of the total population.

Horizontal gene transfer starts by selecting a continuous gene string of the donor chromosome. This procedure uses two parameters:

- The length of the string to be copied (L)
- The starting point of the string to be copied (P)

The length of the string will be calculated based on the fitness difference between two parent chromosomes and its ratio with the difference between the best and worst fitness obtained during the algorithm execution. The value of L is obtained through Eq. 1.

$$L = \frac{|Fitness(CH_{Donor}) - Fitness(CH_{Recipient})|}{|Fitness_{Best} - Fitness_{Worst}|} \times Length\ Of\ Chromosome \quad (1)$$

To obtain the start point of the gene string, a random value with a uniform distribution between zero and the length of the chromosome is used. The value of P is obtained through Eq. 2.

$$P = Uniform\ Random(0, Length\ Of\ Chromosome) \quad (2)$$

After these two parameters are determined, a gene string of the donor chromosome from the point P and with a length of L replaces the corresponding string in the recipient chromosome. If, during the gene transfer, the operator reaches the end of the chromosome before L number of donor chromosome genes are copied

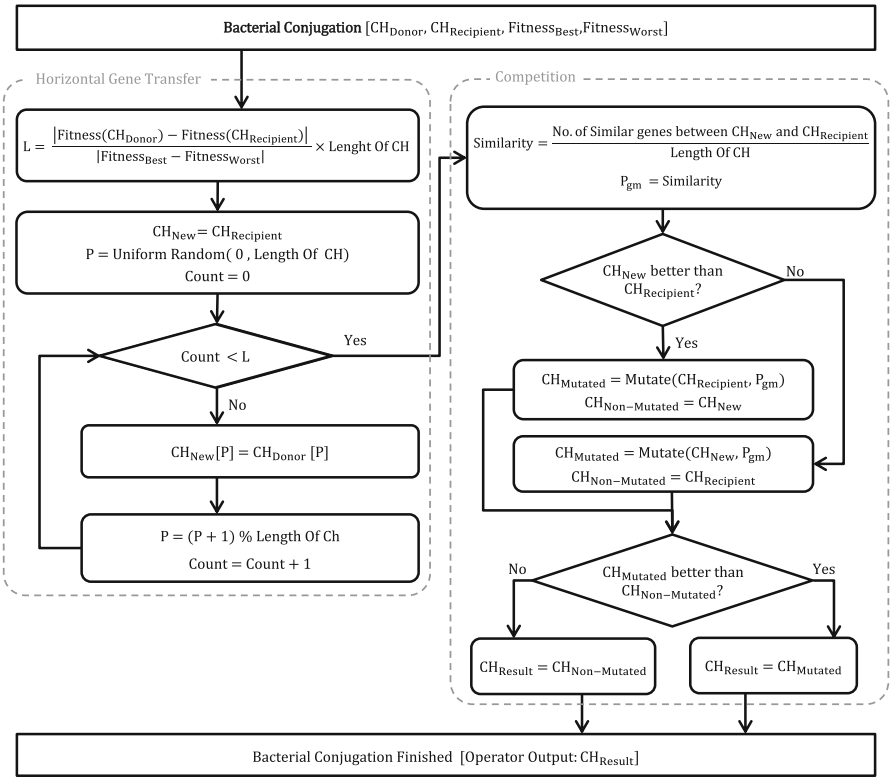

Fig. 1 Block diagram of bacterial conjugation operation. The procedure of bacterial conjugation operator is divided into two main steps: "Horizontal gene transfer" and "Competition", *bordered* with *dash-line*

onto the recipient chromosome; then the copy operation resumes from the beginning of the chromosomes.

It should be noted that as the start point of the copy operation is random, the resulting chromosome will be different each time the operator is applied to the same parents.

2.1.2 Competition

In the second step of BC operator, the chromosome from the gene transfer operation enters into a competition with the recipient. The mutation operator is applied to the chromosome with the worse fitness value in order to increase the probability of its winning. The fitness values of the mutated and non-mutated chromosomes are compared and the winning chromosome replaces the recipient in the population.

In the mutation operator, the number of genes which should be mutated is called the "Per Gene Mutation Rate" (P_{gm}) and it is calculated for each chromosome. The

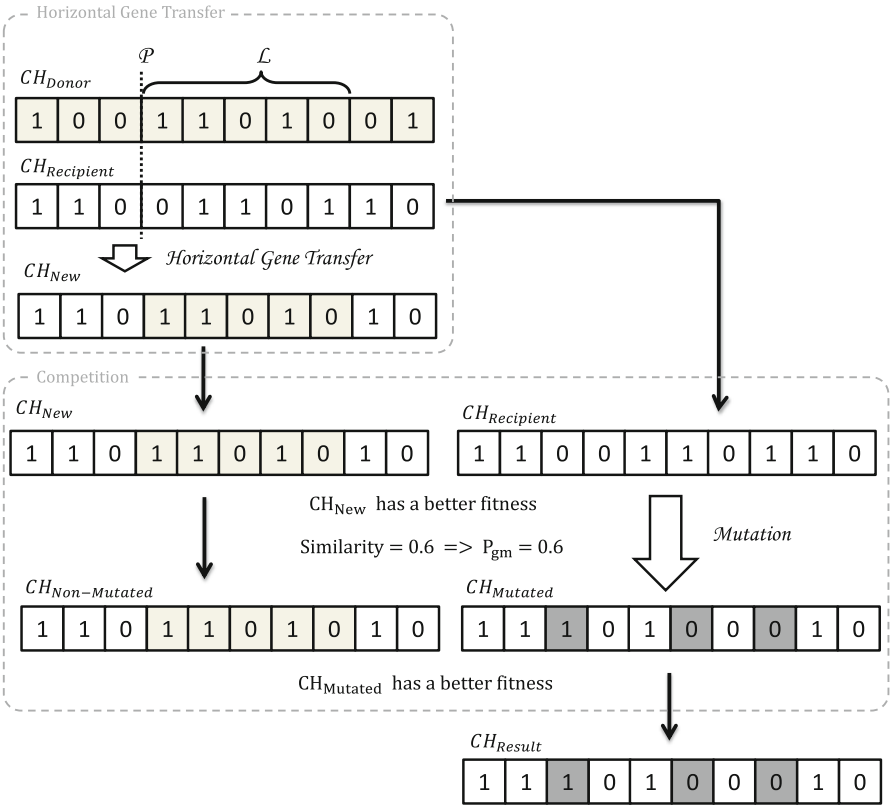

Fig. 2 Example of bacterial conjugation operation for two sample parent chromosomes with length of 10 genes

value of P_{gm} is related to the number of similar genes in the operand chromosomes, which makes it a self-adaptive rate during the algorithm execution.

For each gene a random number between zero and one is generated and when its value is less than P_{gm}, mutation is applied. The more similar the chromosomes are, the more mutation is required to the chromosome with worse fitness value to improve itself and vice versa. The criterion of similarity between chromosomes is defined as the ratio of similar genes in the equivalent positions in two chromosomes to the length of total chromosome, which will be a value between zero and one. Therefore the P_{gm} is calculated by Eq. 3.

$$\begin{aligned} Similarity &= \frac{Number\ of\ similar\ genes\ between\ two\ chromosomes}{Length\ of\ chromosome} \\ P_{gm} &= Similarity \end{aligned} \tag{3}$$

This operation causes the operator to be more effective by mutating the recipient chromosomes which are ineffective in gene transfer. This is most probable when the value of L becomes smaller than the chromosome length and the beneficial genes do

not have the chance of transferring to the recipient chromosome; therefore, the operator is designed so that the closer the recipient chromosome fitness value to the donor chromosome, the larger the mutation rate become.

2.2 The procedure of the HPGA algorithm

The steps of the proposed algorithm are shown in Fig. 3.

(A) The algorithm starts by generating an initial population of chromosomes. For this purpose, random chromosomes are generated up to the "Population Size"—the input argument of the algorithm amounts to the size of the population—which constitutes the initial population.

(B) The only operator in the proposed algorithm is bacterial conjugation and the donor chromosome will be the chromosome with the best fitness ever obtained in the population, which will be referred to as the *BestChromosome*. Therefore, the *BestChromosome* must be remembered. Also, as mentioned in Sect. 2.1, the calculation of L requires that the best and worst fitness values be known, which will be referred to as *BestFitness* and *WorstFitness* values. *BestFitness* can simply be obtained from the *BestChromosome*. The *WorstFitness* must be stored through out the algorithm execution. In this stage, the *BestChromosome* and the *WorstFitness* in the initial population are stored separately from the population.

(C) In the proposed algorithm, one generation is completed when BC operator is applied to all chromosomes of the population. As a result, the following steps are performed for each generation:

(C1) The first chromosome of the population is selected.

(C2) BC operator is applied to the *BestChromosome* as the donor chromosome and to the selected chromosome as the recipient one.

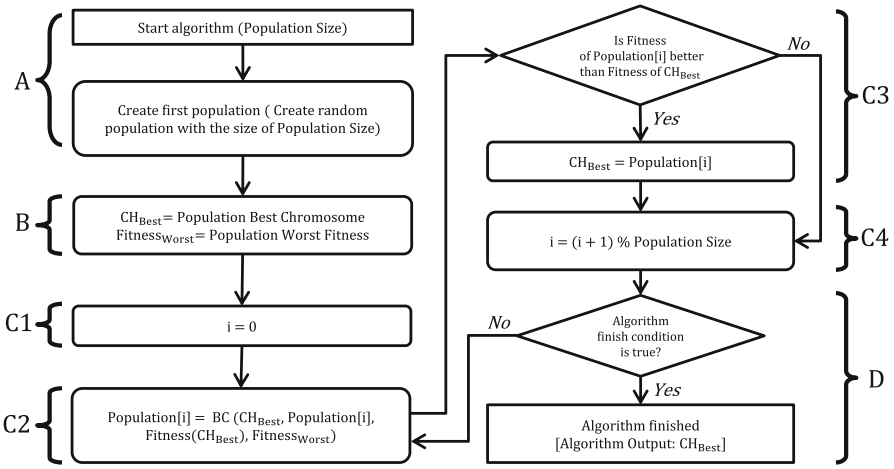

Fig. 3 Block diagram of the HPGA algorithm

(C3) The fitness of the resulting chromosome by the operator is compared with the *BestChromosome*, and if better, it will be chosen as the *BestChromosome*. It is essential to note that the resulting chromosomes from the BC operator always have better or equal fitness compared to the previous ones in the population, so it is not necessary to update the *WorstFitness* value. On the other hand the loser chromosome of the BC operator has not been evaluated to update the *WorstFitness* value either. This is because this chromosome will not ever placed in the population, and if the *WorstFitness* value has been updated according to this chromosome, the fitness value of the chromosomes in the population will become nearer to the *BestFitness* value in the *WorstFitness* and *BestFitness* range. And as a result the value of the L parameter in the gene transfer mechanism of BC operator will become almost zero, which decreases the algorithm performance.

(C4) The operation on the selected chromosome ends in this generation and the algorithm returns to C2 with the selection of the next chromosome in the population.

(D) As with the SGA, the condition for the termination of the proposed algorithm can be the number of generations, which is referred to as max-generation. In the problems solved in Sect. 3 of this paper, to achieve a fair comparison, the termination condition is defined as the number of fitness function calls. The algorithm terminates when the fitness function is called n times, or in other words, it terminates when n chromosomes—n state spaces—are evaluated.

2.3 The advantages of the proposed algorithm

2.3.1 The proposed algorithm as a combination of PSO and SGA algorithm

The PSO algorithm uses its "move" operator to improve its individuals. Similarly, the proposed algorithm uses its BC operator to improve the individuals based on the global best and applies this operator to all the individuals in the population. But the difference is that the BC operator uses evolutionary operators instead of state space vector field operations to improve the individuals. In addition, the gene transfer operator procedure in the BC is somewhat similar to the crossover operator of SGA. Therefore a two-point crossover operator is implied in the BC operator and the crossing length is calculated using a more informed method, which is based on the population state. This mechanism makes the implied operator rate a self-adaptive one.

2.3.2 Fewer input parameters

One of the advantages of the proposed algorithm is that it contains fewer input parameters; therefore, it does not need any special skill or expertise to use. The only input parameter of the algorithm is the population size. In a case in which the

termination condition of the algorithm is the number of the generations, the parameter of max-generation will also be added to its parameters list. Even the basic evolutionary algorithms like SGA and PSO need at least three parameters. Some common versions of PSO require four parameters.

One of today's problems in robotics is to solve unplanned problems, which a robot may encounter during its real world application without interference of a human. Considering every aspect of such problems is not possible in advance. Evolutionary algorithms are a good option for these problems since they are capable of solving almost any complex problem. However improper initialization of them may result in stagnation, so an evolutionary algorithm with a minimum number of parameters and an acceptable performance is a good candidate for solving these problems.

2.3.3 Better performance

- *Removal of the crossover operator computations:* because the crossover operator is not directly applied in the proposed algorithm, all of the calculations relating to this operator, such as selection, comparison and so on, are omitted.
- *Removal of the sorting and operand selecting steps:* In some GA algorithms, there is a sorting process whose best order can be $n \times \log(n)$, where n denotes the population size. For SGA with no sorting step, the selection operator—like tournament—order could be considered as a matter of performance, where there is no operand selecting step for BC operator of HPGA. In the proposed algorithm, the selection of the best chromosome and the worst fitness in the initial population is simultaneous with the generation of the initial population. During the next steps, updating of the best chromosome is carried out all through the algorithm execution. Furthermore, there is not any selection based on fitness throughout the execution. As a result, there is no sorting or process of any kind for finding the maximum and minimum values in the algorithm. Therefore, the proposed algorithm reaches the answer at a much faster rate than algorithms containing these processes like SGA with a sorting step. According to the simulation results in Sect. 3, the SGA with tournament selection and no sorting step and PSO with decimally encoded individuals, show better performance compared with the proposed algorithm in some cases, when they are initialized with their optimum parameter values. But since in the proposed algorithm, no time or energy is wasted on finding the optimum parameters, it could be claimed that the proposed algorithm outperforms the SGA and PSO algorithms. Also the performance of the proposed algorithm is not too sensitive to the individuals encoding, unlike PSO.

2.3.4 The ease of implementation compared with other evolutionary methods

The proposed algorithm has only one operator which has a simple procedure. In addition, as there is neither a selection nor sorting step, the implementation of the proposed algorithm is much simpler. For example, many derivations of the simple

genetic algorithm have been proposed by changing the mechanism of the selection operator, some of them have a high level of complexity in implementation.

Also it should be noticed that, the existence of the two steps in the BC operator which are similar to the classic evolutionary operators does not mean that the proposed algorithm contains two operators, because these two steps are not independent and their operation is completely dependent on each other; neither of them can be removed, nor their sequence can be changed. However, for example, in SGA both crossover and mutation can be removed from the algorithm procedure by setting crossover rate (P_C) or mutation rate (P_M) to zero or their sequence can be changed.

It should mentioned that, having fewer operators makes the implementation simpler because each operator needs its own selection method of operands where HPGA uses the whole population and the best chromosome as the operands for its only operator. When the number of operators in an algorithm grows, the sequence of using operators and the method of applying them to operands matters. E.g. even in text books there is no agreement on the method of applying mutation in SGA. However, the implemented PSO in the paper uses only one operator just like HPGA.

Considering the fact that the proposed algorithm is simpler and less complex even compared with SGA, it yields better results in terms of time and accuracy. Therefore, the proposed algorithm can be an appropriate alternative for SGA in most optimization problems.

2.3.5 Lack of stagnation and premature convergence

The second step of the proposed BC operator is designed to prevent the algorithm from premature convergence by not letting the BC operator generate similar chromosomes.

3 Experiments and results

In this section, the performance and results of the proposed algorithm are compared with SGA and PSO by solving 23 benchmark functions [10] and two extended knapsack and travelling salesman problems. Also, the performance of the proposed algorithm will be examined by solving a robotic problem which requires a real-time response, and the results will be compared with SGA and PSO. In Sects. 3.1 and 3.2 the SGA and PSO algorithms used for comparison are described. In Sects. 3.3 and 3.4 the experimental results are studied. In Sect. 3.5 the optimum input parameter values of the SGA and PSO algorithms for the mentioned problems and the procedure for finding these values are explained, and in final section the permutation version of the gene transfer mechanism, which is used in TSP optimization, is explained.

3.1 Implemented SGA

The implemented simple genetic algorithm is based on crossover, mutation and selection operators which are effected by "Crossover Rate" (P_C), "Mutation Rate"

(P_M) and "Population Size" input parameters. In this algorithm the first population was generated randomly and by applying these three operators on the current population, the next generation is created.

In the crossover operator, according to the value of P_C, a set of double chromosomes has been selected and a single point crossover has been applied to them, and the offspring chromosome is added to the population. After that, according to the value of P_M, a set of chromosomes is selected to mutate and one of their genes, which is selected randomly, is mutated. The selection operator chooses "Population Size" chromosomes from the current population to create the next generation. So, the size of population is constant. In this paper the tournament and sort methods are used as the selection operator.

The common values for P_C and P_M [11] are

$$\begin{cases} P_C = 0.8 \\ P_M = 0.2 \end{cases}$$

3.2 Implemented PSO

Particles moving in the search space using a vector operation is the main idea behind the PSO algorithm. In this algorithm each particle uses a velocity vector to move in the search space. Its value is computed using the best answer obtained between all particles (global optimum) and the best answer which the particle has found (local optimum).

According to "Particle Swarm Central" [12], three successive standard PSO versions have been introduced since 2006 named SPSO 2006, 2007 and 2011. In this paper the SPSO 2011 is used. Its implementation method can be found in [13]. Like SGA, SPSO contains three input parameters: "Population Size" used for generating the first population, "Inertia Weight" (W) and "global and local optimum answers impact factor" (C) which affect the computation of velocity vector.

The common values for W and C [14] are

$$\begin{cases} W = \frac{1}{2Ln(2)} = 0.721 \\ C = \frac{1}{2} + Ln(2) = 1.193 \end{cases}$$

3.3 Test problems

The problems chosen here involve:

- Finding the minimum value of 23 benchmark functions [10], which are noted in Sect. 3.3.1.
- Solving the travelling salesman problem
- The classic knapsack problem is also extended to become a multi-object optimization problem with a very large state space.

The 23 benchmark functions, knapsack problem and TSP with decimal, binary and integer phenotypes, respectively have been optimized using three SGA, SPSO

and HPGA algorithms with different genotypes. Excluding the population size, which is common parameter between these algorithms, HPGA has no other parameters. However both SPSO and SGA have two other parameters, so these algorithms are used in two modes; first, with common values for their input parameters and second, with optimum parameter values for each problem.

Each problem was tried 100 times by each algorithm. Tables 1 and 2 show the average running time of 100 executions, where SGA and PSO used by their optimum parameter values and their common parameter values, respectively. In all the result tables including Tables 1 and 2, the best result obtained is appeared in bold-face font. Figures 4 and 5 illustrate the average running time of the three algorithms for 25 problems, where SGA and PSO used by their optimum parameter values and their common parameter values. Also Table 3 and Fig. 6 show the results and runtimes of the HPGA algorithm in comparison to SGA with optimum parameter values when it used by tournament and sort selection. Figure 7 is diagram of Best Fitness and Average Fitness with respect to the generation of the "Quartic Function with Noise". The termination condition is 400,000 calling of the fitness function for these algorithms. Experimental results show that the HPGA has better performance in both aspect of results and runtime than SGA and SPSO, when they are used with their common parameter values. But in the case that these algorithm used their optimum parameter values, their results are similar to the HPGA results, and the SGA runtime is approximately equal to the HPGA runtime. But it should be mentioned that significant time and energy must be spent finding the SGA and PSO optimum parameter values for each problem. It's essential to note that, one can expect some classes of problems to suffer worse performance under the proposed algorithm, because of the "No Free Lunch Theorem" [15].

3.3.1 Benchmark functions

In this section the 23 benchmark functions used as the test problems are noted.

(A) Sphere Model

$$f_1(x) = \sum_{i=1}^{6} x_i^2$$
$$-100 \le x_i \le 100, min(f_1) = f_1(0, \ldots, 0) = 0$$

(B) Schwefel's Problem 2.22

$$f_2(x) = \sum_{i=1}^{6} |x_i| + \Pi_{i=1}^{6} |x_i|$$
$$-10 \le x_i \le 10, min(f_2) = f_2(0, \ldots, 0) = 0$$

Table 1 Average of best fitness obtained through 100 executions for the test problems when SGA and PSO used their optimum parameter values

Application	Encoding	Expected result	HPGA	SGA	PSO
f1	Decimal	0	0.000013	0.000033	**0**
f1	Binary	0	0.000005	**0.000001**	0.500603
f2	Decimal	0	0.008854	0.002203	**0**
f2	Binary	0	0.00042	0.00061	**0**
f3	Decimal	0	0.008745	12.03791	**0**
f3	Binary	0	0.000154	**0.000001**	0.3901
f4	Decimal	0	0.07426	0.036865	**0**
f4	Binary	0	**0.000054**	0.000763	11.418915
f5	Decimal	0	**3.638511**	13.745817	26.536643
f5	Binary	0	1.770393	**1.509439**	2.608419
f6	Decimal	0	0.00087	**0**	**0**
f6	Binary	0	**0**	**0**	633
f7	Decimal	0	**3.625306**	6.114657	6.009854
f7	Binary	0	1.094668	**0.563962**	1.964208
f8	Decimal	$30 \times (-418.983)$	−12,190.43188	**−12,569.486519**	−7,137.747677
f8	Binary	$10 \times (-418.983)$	−4,189.711612	**−4,189.828873**	−3,494.020397
f9	Decimal	0	0.007753	0.00003	**0**
f9	Binary	0	0.000697	**0**	54.098558
f10	Decimal	0	0.001838	0.001375	**0**
f10	Binary	0	0.001757	**0**	8.884939
f11	Decimal	0	0.001038	0.000068	**0**
f11	Binary	0	0.000038	**0**	6.671428
f12	Decimal	0	0.00053	**6.60322e−8**	0.602125
f12	Binary	0	0.000047	**9.73957e−8**	13.3368
f13	Decimal	0	0.10258	**1.51871e−7**	0.821629
f13	Binary	0	0.053274	**4.50992e−7**	763.096
f14	Decimal	0.998004	**0.998004**	**0.998004**	**0.998004**
f14	Binary	0.998004	**0.998004**	**0.998004**	**0.998004**
f15	Decimal	0.0003075	0.000330435	0.000370834	**0.000307668**
f15	Binary	0.0003075	0.00030957	0.000461189	**0.000307618**
f16	Decimal	−1.03163	**−1.03163**	**−1.03163**	**−1.03163**
f16	Binary	−1.03163	**−1.03163**	**−1.03163**	**−1.03163**
f17	Decimal	0.397	**0.397887**	**0.397887**	0.397888
f17	Binary	0.397	**0.397887**	0.397888	0.397888
f18	Decimal	3	**3**	**3**	3.00002
f18	Binary	3	**3**	**3**	**3**
f19	Decimal	−3.86	**−3.86278**	**−3.86278**	**−3.86278**
f19	Binary	−3.86	**−3.86278**	**−3.86278**	**−3.86278**
f20	Decimal	−3.32	−3.32198	**−3.322**	−3.31645
f20	Binary	−3.32	**−3.32199**	**−3.32199**	−3.32161
f21	Decimal	−10.1532	−10.1531	**−10.1532**	−9.84265

Table 1 continued

Application	Encoding	Expected result	HPGA	SGA	PSO
f21	Binary	−10.1532	**−10.1532**	**−10.1532**	−10.0955
f22	Decimal	−10.4029	**−10.4029**	**−10.4029**	−10.2002
f22	Binary	−10.4029	**−10.4029**	**−10.4029**	−10.3955
f23	Decimal	−10.5364	**−10.5364**	**−10.5364**	−10.1926
f23	Binary	−10.5364	**−10.5364**	**−10.5364**	−10.2612
EKP	Binary	897	**71**	**71**	**71**
TSP	Integer	19	**19**	**19**	**19**

The best result obtained is appeared in bold-face font

(C) Schwefel's Problem 1.2

$$f_3(x) = \sum_{i=1}^{6} \left(\sum_{j=1}^{i} x_j \right)^2$$
$$-100 \le x_i \le 100, min(f_3) = f_3(0, \ldots, 0) = 0$$

(D) Schwefel's Problem 2.21

$$f_4(x) = \max\{|x_i|, 1 \le i \le 6\}$$
$$-100 \le x_i \le 100, min(f_4) = f_4(0, \ldots, 0) = 0$$

(E) Generalized Rosenbrock's Function

$$f_5(x) = \sum_{i=1}^{5} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$
$$-30 \le x_i \le 30, min(f_5) = f_5(1, \ldots, 1) = 0$$

(F) Step Function

$$f_6(x) = \sum_{i=1}^{6} (\lfloor x_i + 0.5 \rfloor)^2$$
$$-100 \le x_i \le 100, min(f_6) = f_6(0, \ldots, 0) = 0$$

(G) Quartic Function with Noise

$$f_7(x) = \sum_{i=1}^{6} i x_i^4 + random[0, 1)$$
$$-1.28 \le x_i \le 1.28, min(f_7) = f_7(0, \ldots, 0) = 0$$

(H) Generalized Schwefel's Problem 2.26

$$f_8(x) = -\sum_{i=1}^{m} (x_i \sin(\sqrt{|x_i|}))$$
$$-500 \le x_i \le 500, min(f_8) = f_8(420.9687, \ldots, 420.9687) = m \times (-418.983)$$

Table 2 Average of best fitness obtained through 100 executions for the test problems when SGA and PSO used their common parameter values

Application	Encoding	Expected result	HPGA	SGA	PSO
f1	Decimal	0	**0.000013**	0.438687	0.128125
f1	Binary	0	**0.000005**	1.645251	6.619401
f2	Decimal	0	**0.008854**	0.365457	0.857875
f2	Binary	0	**0.00042**	1.099854	11.236007
f3	Decimal	0	**0.008745**	16.282234	7.764119
f3	Binary	0	**0.000154**	0.999153	1.052671
f4	Decimal	0	**0.07426**	4.125077	6.24558
f4	Binary	0	**0.000054**	8.008575	20.148468
f5	Decimal	0	**3.638511**	52.868608	69.092769
f5	Binary	0	**1.770393**	2.11677	6.688011
f6	Decimal	0	**0.00087**	0.01578	0.00874
f6	Binary	0	**0**	90	2,829
f7	Decimal	0	**3.625306**	7.28081	6.51352
f7	Binary	0	**1.094668**	1.121742	3.467391
f8	Decimal	$30 \times (-418.983)$	**−12,190.431879**	−8,268.428366	−6,014.414786
f8	Binary	$10 \times (-418.983)$	**−4,189.711612**	−4,172.302144	−2,565.533561
f9	Decimal	0	**0.007753**	145.928876	297.630614
f9	Binary	0	**0.000697**	12.723112	55.806845
f10	Decimal	0	**0.001838**	15.958961	19.697271
f10	Binary	0	**0.001757**	5.249146	13.740854
f11	Decimal	0	**0.001038**	97.124734	276.628994
f11	Binary	0	**0.000038**	1.318305	29.209391
f12	Decimal	0	**0.00053**	122.879	278.534
f12	Binary	0	**0.000047**	1.47835	18.1066
f13	Decimal	0	**0.10258**	18,487,200	34,759,7000
f13	Binary	0	**0.053274**	4.74326	1143170
f14	Decimal	0.998004	**0.998004**	**0.998004**	**0.998004**
f14	Binary	0.998004	**0.998004**	**0.998004**	**0.998004**
f15	Decimal	0.0003075	**0.000330435**	0.000437346	0.000638565
f15	Binary	0.0003075	**0.00030957**	0.000630207	0.000929222
f16	Decimal	−1.03163	**−1.03163**	**−1.03163**	**−1.03163**
f16	Binary	−1.03163	**−1.03163**	**−1.03163**	−1.0315
f17	Decimal	0.397	**0.397887**	0.397888	**0.397887**
f17	Binary	0.397	**0.397887**	0.397888	0.398012
f18	Decimal	3	**3**	**3**	**3**
f18	Binary	3	**3**	**3**	**3**
f19	Decimal	−3.86	**−3.86278**	**−3.86278**	−3.86273
f19	Binary	−3.86	**−3.86278**	**−3.86278**	−3.86204
f20	Decimal	−3.32	**−3.32198**	−3.30796	−3.3113
f20	Binary	−3.32	**−3.32199**	−3.31928	−3.16271
f21	Decimal	−10.1532	**−10.1531**	−9.48375	−7.26976

Table 2 continued

Application	Encoding	Expected result	HPGA	SGA	PSO
f21	Binary	−10.1532	**−10.1532**	−10.0299	−5.01281
f22	Decimal	−10.4029	**−10.4029**	−10.3088	−9.74389
f22	Binary	−10.4029	**−10.4029**	**−10.4029**	−10.0863
f23	Decimal	−10.5364	**−10.5364**	−10.3718	−7.4978
f23	Binary	−10.5364	**−10.5364**	**−10.5364**	−5.4992
EKP	Binary	897	**71**	**71**	69.5
TSP	Integer	19	**19**	**19**	21

The best result obtained is appeared in bold-face font

(I) Generalized Rastrigin's Function

$$f_9(x) = \sum_{i=1}^{6}[x_i^2 - 10\cos(2\pi x_i) + 10]$$
$$-5.12 \le x_i \le 5.12, min(f_9) = f_9(0, \ldots, 0) = 0$$

(J) Ackley's Function

$$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{6}x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{6}\cos(2\pi x_i)\right) + 20 + e$$
$$-32 \le x_i \le 32, min(f_{10}) = f_{10}(0, \ldots, 0) = 0$$

(K) Generalized Griewank Function

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{6}x_i^2 - \Pi_{i=1}^{6}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

(L) Generalized Penalized Functions

$$f_{12}(x) = \frac{\pi}{30}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{5}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_6 - 1)^2\right) + \sum_{i=1}^{6}u(x_i, 10, 100, 4)$$
$$-50 \le x_i \le 50, min(f_{12}) = f_{12}(1, \ldots, 1) = 0$$
$$f_{13}(x) = \frac{1}{10}(\sin^2(3\pi x_1) + \sum_{i=1}^{5}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_6 - 1)^2[1 + sin^2(2\pi x_6]) + \sum_{i=1}^{6}u(x_i, 5, 100, 4)$$
$$-50 \le x_i \le 50, min(f_{13}) = f_{13}(1, \ldots, 1) = 0$$

where

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

(M) Shekel's Foxholes Function

$$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - aij)^6}\right]^{-1}$$

$$-65.536 \le x_i \le 65.536, min(f_{14}) = f_{14}(-32, -32) = 0.998004$$

where

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

(N) Kowalik's Function

$$f_{15}(x) = \sum_{i=1}^{11}[a_i - (x_1(b_i^2 + b_i x_2))(b_i^2 + b_i x_3 + x_4)^{-1}]^2 \quad -5 \le x_i \le 5$$

$$min(f_{15}) = f_{15}(0.1928, 0.1908, 0.1231, 0.1358) = 0.0003075$$

where

$$(a_i) = \begin{pmatrix} 0.1957 \\ 0.1947 \\ 0.1735 \\ 0.16 \\ 0.0844 \\ 0.0627 \\ 0.0456 \\ 0.0342 \\ 0.0323 \\ 0.0235 \\ 0.0246 \end{pmatrix}, \quad (b_i^{-1}) = \begin{pmatrix} 0.25 \\ 0.5 \\ 1 \\ 2 \\ 4 \\ 6 \\ 8 \\ 10 \\ 12 \\ 14 \\ 16 \end{pmatrix}$$

(O) Six-Hump Camel-Back Function

$$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

$$-5 \le x_i \le 5$$

$$min(f_{16}) = f_{16}(0.08983, -0.7126) = f_{16}(-0.08983, 0.7126) = -1.0316285$$

(P) Branin Function

$$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$
$$-5 \le x_1 \le 10, 0 \le x_2 \le 15$$
$$min(f_{17}) = f_{17}(-3.14, 12.27) = f_{17}(3.14, 2.27) = f_{17}(9,42, 2.42) = 0.397888$$

(Q) Goldstein-Price Function

$$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$\times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$
$$-2 \le x_i \le 2, min(f_{18}) = f_{18}(0, -1) = 3$$

(R) Hartman's Family

$$f_{19}(x) = -\sum_{i=1}^{4} c_i exp\left[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right]$$
$$0 \le x_i \le 1, min(f_{19}) = f_{19}(0.114, 0.556, 0.852) = -3.86278$$

where

$$(a_{ij}) = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, (c_i) = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}, (p_{ij}) = \begin{pmatrix} 0.3689 & 0.117 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}$$

$$f_{20}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$$
$$0 \le x_i \le 1$$
$$min(f_{20}) = f_{20}(0.201, 0.150, 0.477) = f_{20}(0.275, 0.311, 0.657) = -3.32199$$

where

$$(a_{ij}) = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, (c_i) = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}$$

$$(p_{ij}) = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1415 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

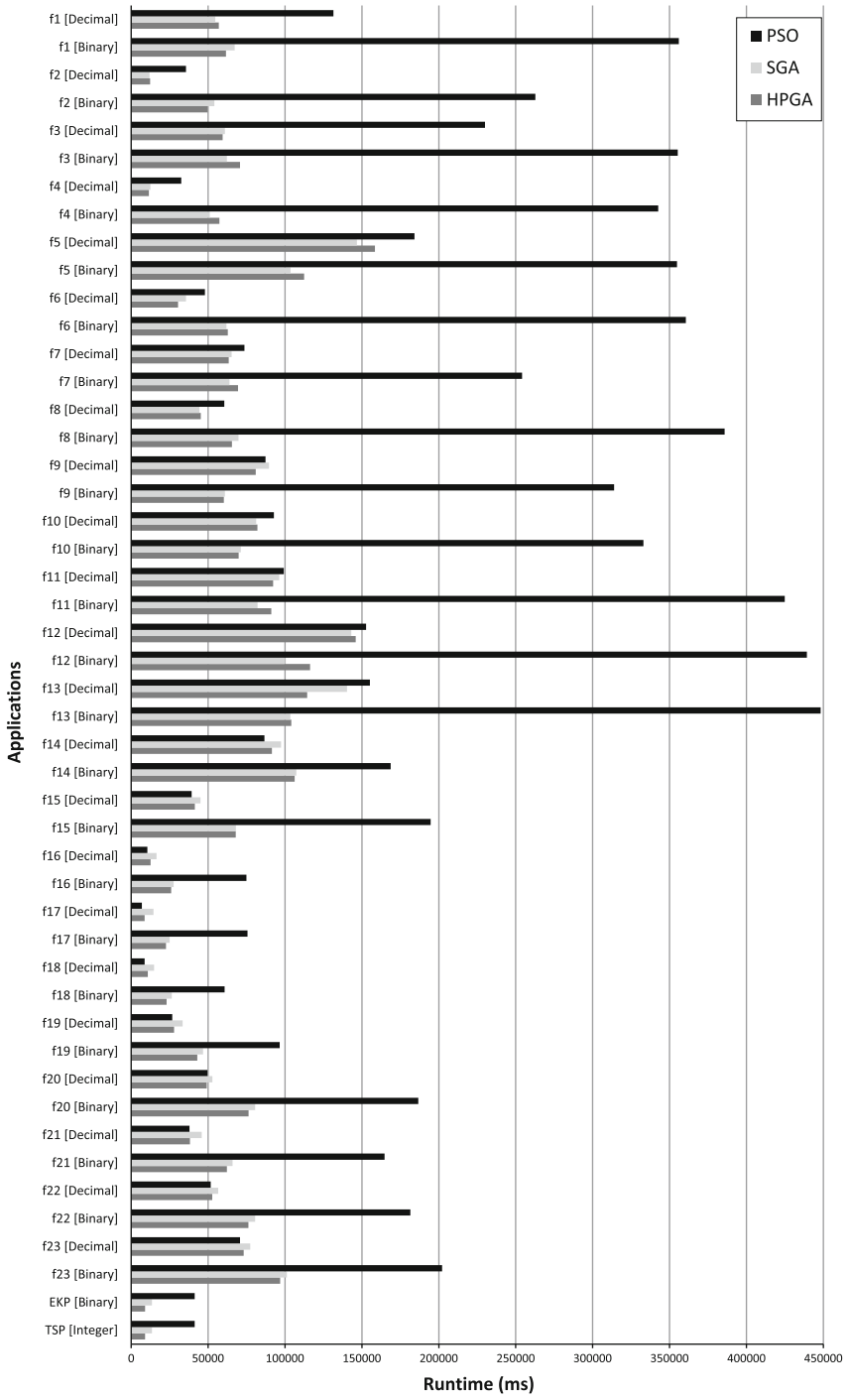

Fig. 4 Average run time for the three algorithms when SGA and PSO use their optimum parameter values

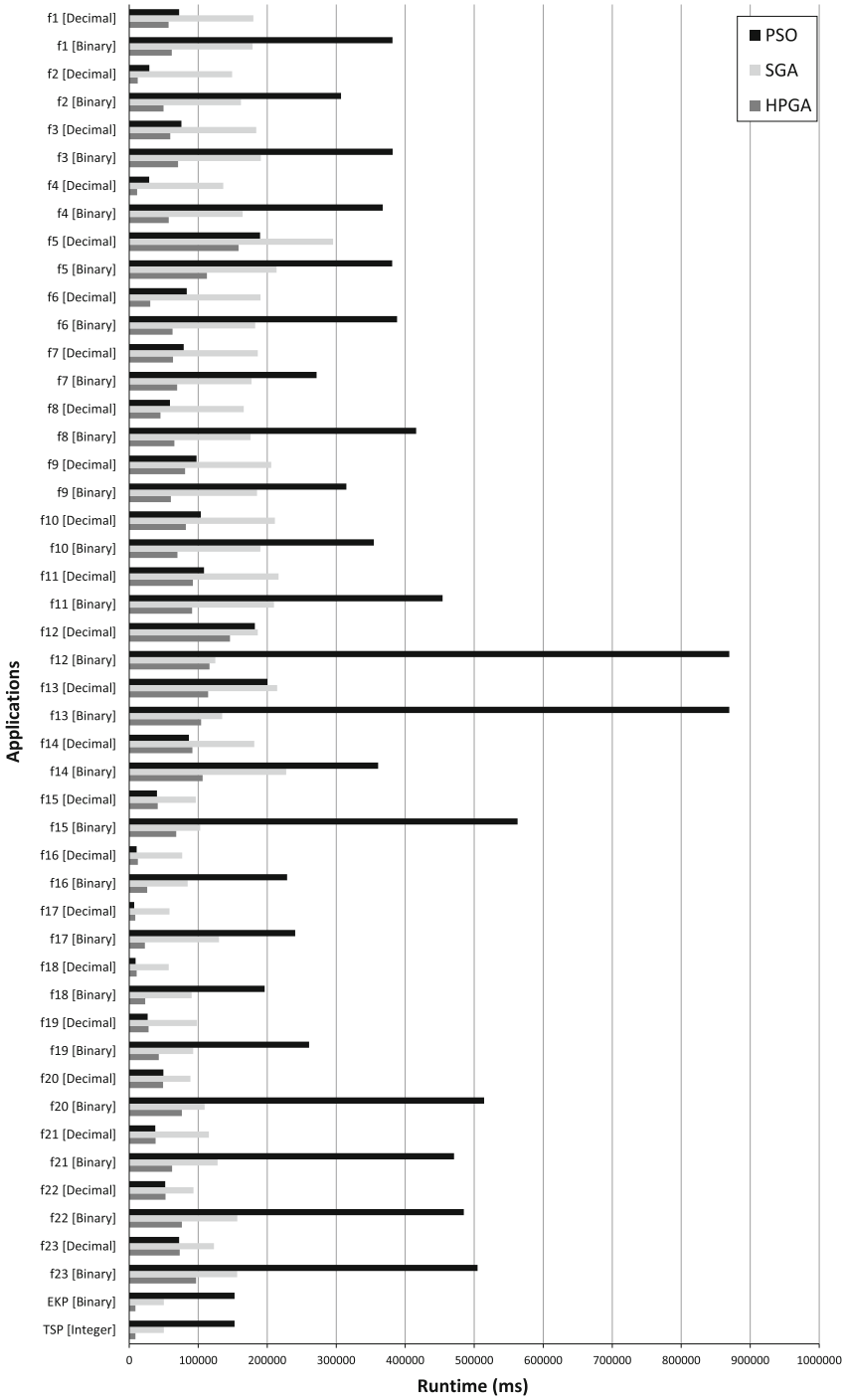

Fig. 5 Average run time for the three algorithms when SGA and PSO use their common parameter values

Table 3 Average of best fitness obtained through 100 executions for the test problems when SGA and PSO used tournament and sort selection

Application	Encoding	Expected result	HPGA	SGA (tournament)	SGA (sort)
f1	Decimal	0	**0.000013**	0.000033	0.000033
f1	Binary	0	0.000005	**0.000001**	0.00005
f2	Decimal	0	0.008854	**0.002203**	**0.002203**
f2	Binary	0	**0.00042**	0.00061	0.00108
f3	Decimal	0	**0.008745**	12.03791	13.03791
f3	Binary	0	0.000154	0.000001	**0**
f4	Decimal	0	0.07426	**0.036865**	**0.036865**
f4	Binary	0	0.000054	0.000763	**0**
f5	Decimal	0	**3.638511**	13.745817	3.748975
f5	Binary	0	1.770393	1.509439	**1.358078**
f6	Decimal	0	0.00087	**0**	**0**
f6	Binary	0	**0**	**0**	**0**
f7	Decimal	0	**3.625306**	6.114657	4.854199
f7	Binary	0	1.094668	0.563962	**0.239284**
f8	Decimal	$30 \times (-418.983)$	−12,190.431	**−12,569.486**	**−12,569.486**
f8	Binary	$10 \times (-418.983)$	−4,189.711612	**−4,189.828873**	−4,139.422779
f9	Decimal	0	0.007753	0.00003	**0**
f9	Binary	0	0.000697	**0**	**0**
f10	Decimal	0	0.001838	0.001375	**0**
f10	Binary	0	0.001757	**0**	**0**
f11	Decimal	0	0.001038	0.000068	**0**
f11	Binary	0	0.000038	**0**	**0**
f12	Decimal	0	0.00053	6.60322E−08	**0**
f12	Binary	0	0.000047	9.73957E−08	**0**
f13	Decimal	0	0.10258	1.51871E−06	**0**
f13	Binary	0	0.053274	4.50992E−07	**0**
f14	Decimal	0.998004	**0.998004**	**0.998004**	**0.998004**
f14	Binary	0.998004	**0.998004**	**0.998004**	**0.998004**
f15	Decimal	0.0003075	**0.000330435**	0.000370834	0.000451367
f15	Binary	0.0003075	**0.00030957**	0.000461189	0.00051446
f16	Decimal	−1.03163	**−1.03163**	**−1.03163**	**−1.03163**
f16	Binary	−1.03163	**−1.03163**	**−1.03163**	**−1.03163**
f17	Decimal	0.397	**0.397887**	**0.397887**	**0.397887**
f17	Binary	0.397	**0.397887**	0.397888	0.397888
f18	Decimal	3	**3**	**3**	3.00002
f18	Binary	3	**3**	**3**	**3**
f19	Decimal	−3.86	**−3.86278**	**−3.86278**	**−3.86278**
f19	Binary	−3.86	**−3.86278**	**−3.86278**	**−3.86278**
f20	Decimal	−3.32	−3.32198	**−3.322**	**−3.322**
f20	Binary	−3.32	**−3.32199**	**−3.32199**	**−3.32199**
f21	Decimal	−10.1532	−10.1531	**−10.1532**	**−10.1532**

Table 3 continued

Application	Encoding	Expected result	HPGA	SGA (tournament)	SGA (sort)
f21	Binary	−10.1532	**−10.1532**	**−10.1532**	**−10.1532**
f22	Decimal	−10.4029	**−10.4029**	**−10.4029**	**−10.4029**
f22	Binary	−10.4029	**−10.4029**	**−10.4029**	**−10.4029**
f23	Decimal	−10.5364	**−10.5364**	**−10.5364**	**−10.5364**
f23	Binary	−10.5364	**−10.5364**	**−10.5364**	**−10.5364**
EKP	Binary	897	**71**	**71**	**71**
TSP	Integer	19	**19**	**19**	**19**

The best result obtained is appeared in bold-face font

(S) Shekel's Family

$$f_{21}(x) = -\sum_{i=1}^{5}[(x-a_i)(x-a_i)^T + c_i]^{-1}$$
$$0 \le x_i \le 10, min(f_{21}) = f_{21}(4,4,4,4) = -10.1532$$
$$f_{22}(x) = -\sum_{i=1}^{7}[(x-a_i)(x-a_i)^T + c_i]^{-1}$$
$$0 \le x_i \le 10, min(f_{22}) = f_{22}(4,4,4,4) = -10.4029$$
$$f_{23}(x) = -\sum_{i=1}^{10}[(x-a_i)(x-a_i)^T + c_i]^{-1}$$
$$0 \le x_i \le 10, min(f_{23}) = f_{23}(4,4,4,4) = -10.5364$$

where

$$(a_{ij}) = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}, (c_i) = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}$$

3.4 Inverse kinematics problem

In robotics, one of the most difficult problems is the inverse kinematics problem.

The inverse kinematics of a robot manipulator is the operation of finding joint variables for a given position and orientation of end-effector. Real-time solving of

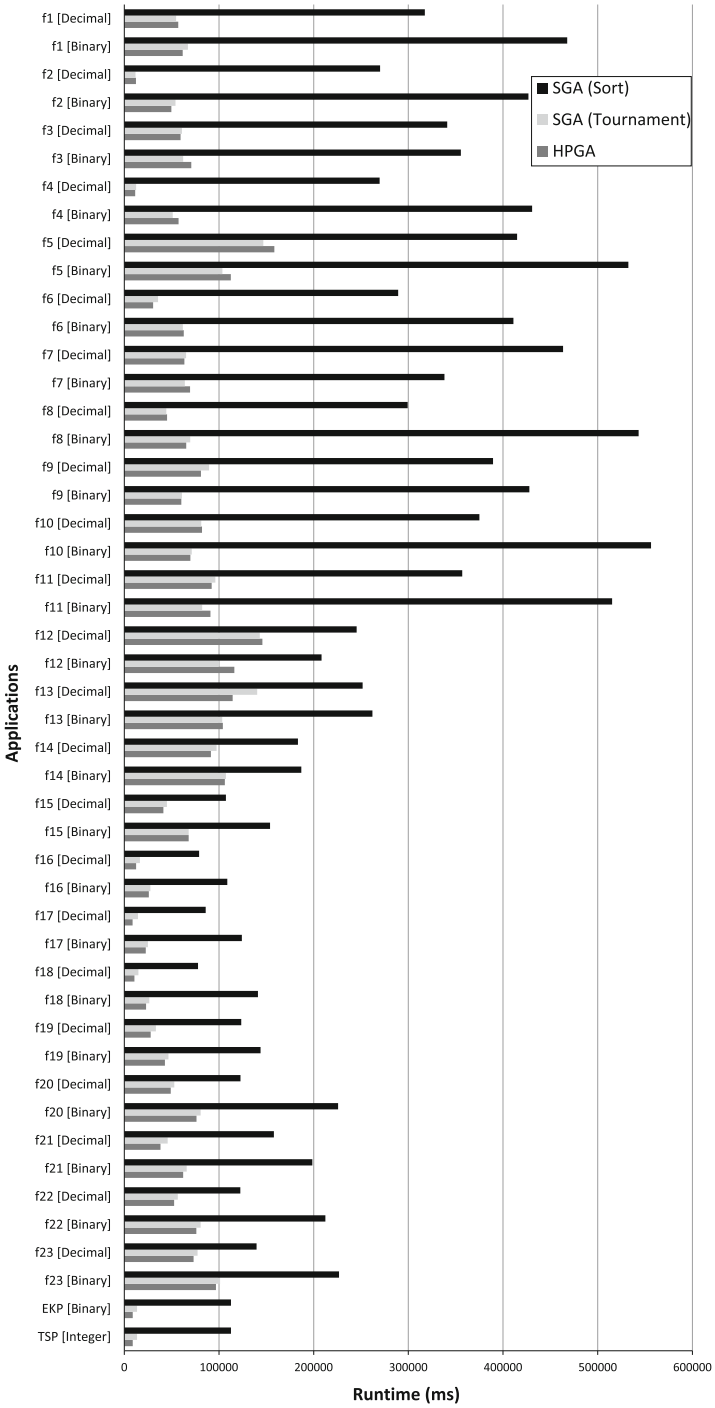

Fig. 6 Average run time for the three algorithms when SGA uses tournament and sort selection

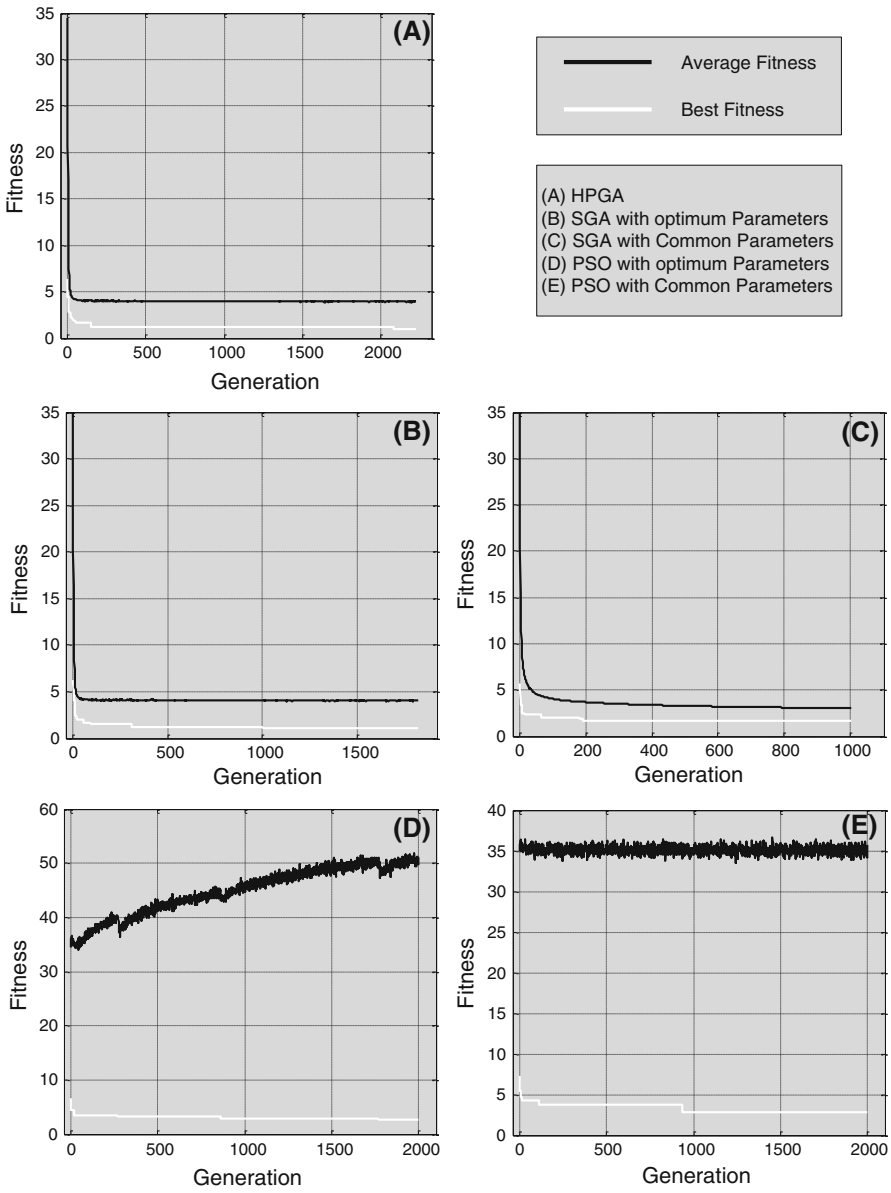

Fig. 7 Fitness diagram Of "Quartic Function with Noise" problem for three SGA, SPSO and HPGA algorithms. SGA and SPSO fitness diagrams are shown when they are initialized with both common and the optimum parameters

the inverse kinematics problem requires a time constraint for the algorithms. In this section, an inverse kinematics problem will be solved by the proposed algorithm, SGA and PSO, and the results will be compared.

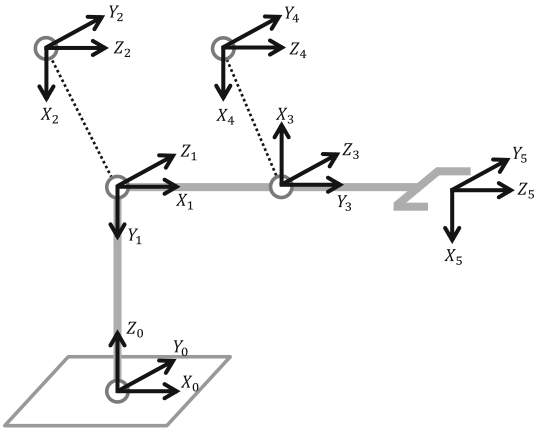

Fig. 8 Manipulator with five degree of freedom with coordinate frames assigned to each joints based on Denavit–Hartenberg convention

Table 4 Denavit–Hartenberg parameters for 5-DOF manipulator

Joint	D–H Parameters				Work range	
	a	α	d	θ	Start	End
1	0	−π/2	30	q_1	−180°	+180°
2	0	π/2	0	q_2	−150°	+150°
3	0	π/2	q_3	π	30 cm	60 cm
4	0	π/2	0	q_4	−150°	+150°
5	0	0	50	q_5	−180°	+180°

One way to express the structure of the manipulator is to use the Denavit–Hartenberg (D–H) model [16]. In this model, each joint is represented by Cartesian coordinates of $X_iY_iZ_i$, and for each joint, four parameters are obtained from these coordinates.

For this problem, a manipulator with five degrees of freedom is used (Fig. 8). D–H parameters of the manipulator are obtained, which are shown in Table 4. In Table 4, θ signifies rotation about Z axis; α, rotation about X axis; d, distance between Z axes of two consecutive joints; and a, distance between X axes of two consecutive joints [17]. So far, there is a homogeneous transformation matrix for each joint obtained from the D–H parameters, which is noted as Q_i and is calculated through Eq. 4, whose consecutive multiplying gives the position and orientation of the end-effector to the base coordinates called **Q**, which depends on the joint variables. In these matrices, C_x and S_x are denoted by $\cos(x)$ and $\sin(x)$, respectively.

$$\mathbf{Q}_i = \begin{bmatrix} C_{\theta_i} & -C_{\alpha_i}S_{\theta_i} & S_{\alpha_i}S_{\theta_i} & a_iC_{\theta_i} \\ S_{\theta_i} & C_{\alpha_i}C_{\theta_i} & -S_{\alpha_i}C_{\theta_i} & a_iS_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Table 5 Algorithms results for the inverse kinematics problem

	Common params		Optimal params		HPGA
	SGA	PSO	SGA	PSO	
Average position error (cm)	0.4310	1.7624	0.1135	0.7317	**0.1134**
Average orientation error (°)	1.0363	1.5586	0.9153	1.3785	**0.8960**
Max position error (cm)	1.3707	3.0349	0.5056	1.6510	**0.4594**
Max orientation error (°)	2.0410	2.4759	1.9178	2.2877	**1.6803**
Average evaluated chromosomes	66,444.8	33,025	67,058.8	34,577	**68,174**

The best result obtained is appeared in bold-face font

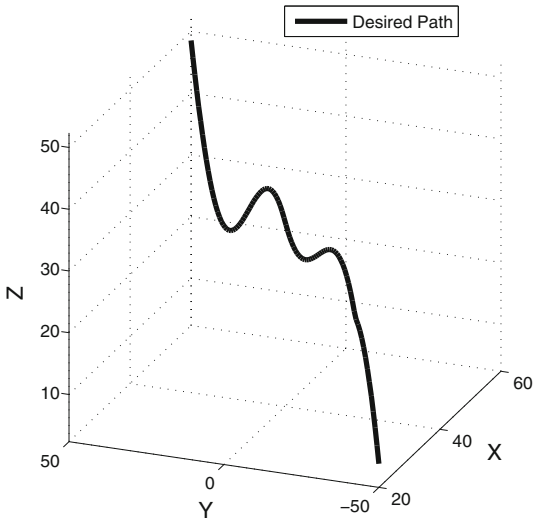

Fig. 9 The manipulator's desired path achieved from the Eq. (7). The manipulator's end-effector should track this path

In the second stage, the desired position and orientation of the end-effector is defined through the calculation of the homogeneous transformation matrix. The homogeneous transformation matrix of the end-effector is called **E**. It is calculated through Eq. 5, where X, Y and Z are coordinates of the end-effector position and α, β and γ are the angles of its orientation in 3D Euclidean space.

$$\mathbf{E} = \begin{bmatrix} C_\beta C_\gamma & -C_\beta S_\gamma & S_\beta & X \\ S_\alpha S_\beta C_\gamma + C_\alpha S_\gamma & -S_\alpha S_\beta S_\gamma + C_\alpha C_\gamma & -S_\alpha C_\beta & Y \\ -C_\alpha S_\beta C_\gamma + S_\alpha S_\gamma & C_\alpha S_\beta S_\gamma + S_\alpha C_\gamma & C_\alpha C_\beta & Z \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Now the problem can be defined as: acquire the joint variables in a way that matrix **E** is obtained by assigning them in matrix **Q**.

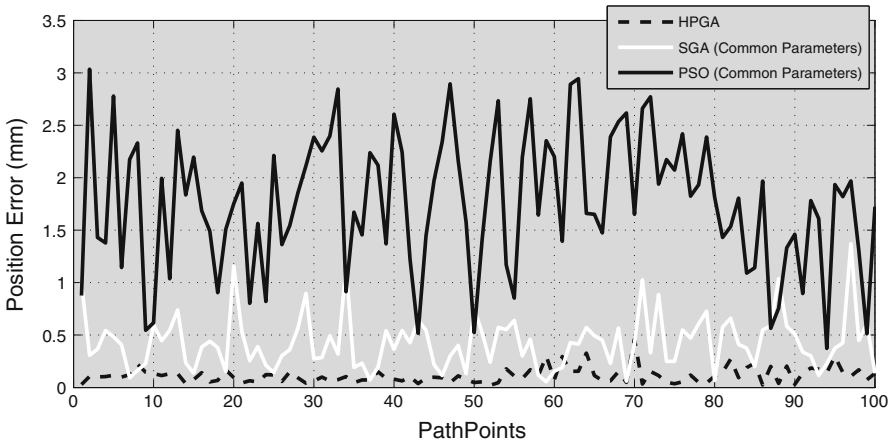

Fig. 10 Position error in the path tracked by the manipulator when SGA and PSO use their common parameter values

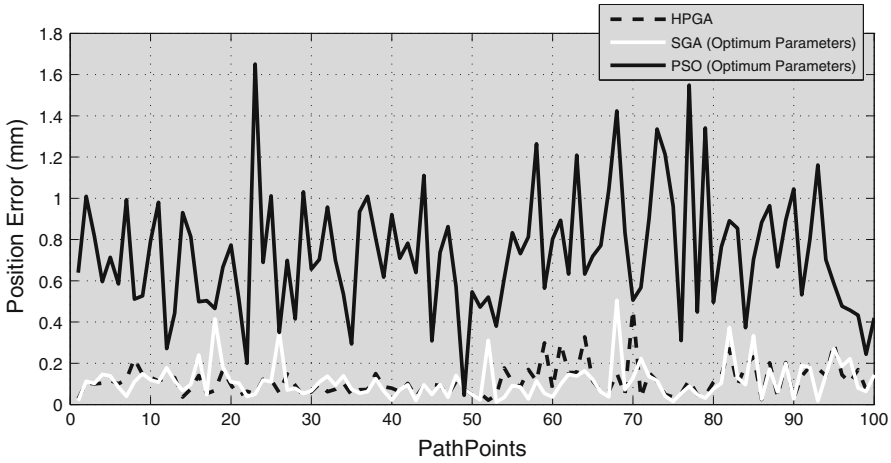

Fig. 11 Position error in the path tracked by the manipulator when SGA and PSO use their optimum parameter values

The fitness function is defined as the sum of the difference of the elements of the two matrices, which are shown in Eq. 6. Then Eq. 6 should be minimized. Since Eq. 6 represents the min norm of two matrices **E** and **Q**, its minimum value is zero.

$$Fitness = \sum_{i=1}^{3} \sum_{j=1}^{4} |E_{ij} - Q_{ij}| \tag{6}$$

As mentioned before, a manipulator with five degree of freedom, shown in Fig. 8, is used for this problem. The joint specifications of the manipulator, including its D–H parameters and work range, is shown in Table 4.

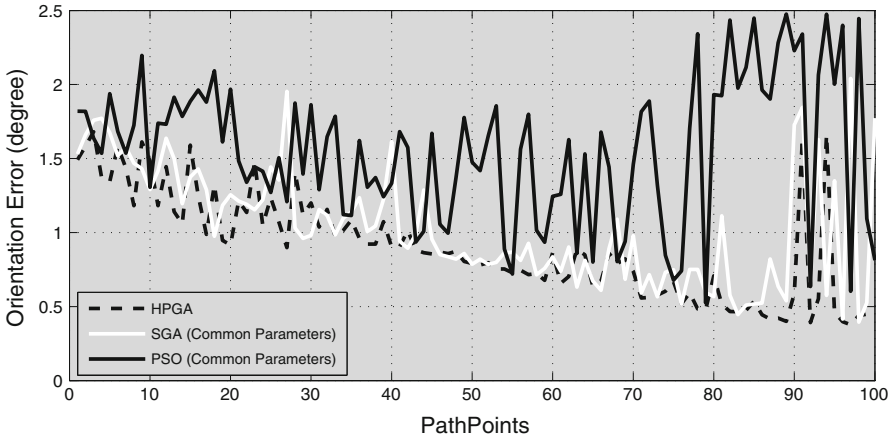

Fig. 12 Orientation error in the path tracked by the manipulator when SGA and PSO use their common parameter values

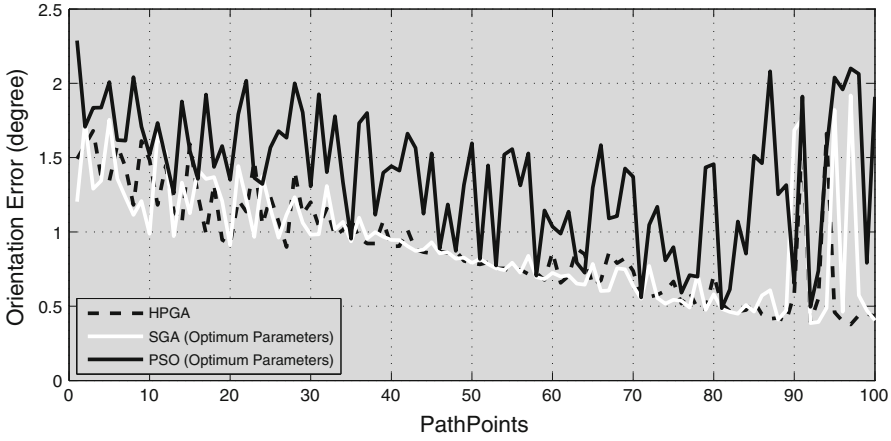

Fig. 13 Orientation error in the path tracked by the manipulator when SGA and PSO use their optimum parameter values

The goal of the problem is to move the end-effector between two points based on Eq. 7.

100 points are selected uniformly between the start and end points. The joint variables for each of these points are calculated, and then the end-effector can track the path point by point.

$$\begin{aligned} Z &= Y \times |\sin(Y)| - X \times |\cos(X)| \\ \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} &= \begin{bmatrix} -50 \\ 20 \end{bmatrix}, \begin{bmatrix} X_{100} \\ Y_{100} \end{bmatrix} = \begin{bmatrix} 50 \\ 60 \end{bmatrix} \end{aligned} \tag{7}$$

Table 6 Optimum values of the SGA and PSO input parameters for each test problem

Problem	Encoding	P_C	P_M	W	C
f_1 (Sphere Model Function)	Decimal	0.7	0.4	0.2	1.4
f_1 (Sphere Model Function)	Binary	0.5	0.1	0.4	1.4
f_2 (Schwefel's Problem 2.22)	Decimal	0.9	0.4	0.2	1.2
f_2 (Schwefel's Problem 2.22)	Binary	0.6	0.1	1.0	1.4
f_3 (Schwefel's Problem 1.2)	Decimal	0.9	0.3	0.2	0.4
f_3 (Schwefel's Problem 1.2)	Binary	0.6	0.3	0.6	1.8
f_4 (Schwefel's Problem 2.1)	Decimal	1.0	0.4	0.2	0.6
f_4 (Schwefel's Problem 2.1)	Binary	0.3	0.2	0.4	1.6
f_5 (Generalized Rosenbrok's Function)	Decimal	0.7	0.4	0.2	1.6
f_5 (Generalized Rosenbrok's Function)	Binary	0.4	0.1	0.6	1.4
f_6 (Step Function)	Decimal	0.9	0.6	0.4	0.2
f_6 (Step Function)	Binary	0.5	0.2	0.4	0.8
f_7 (Quartic Function with Noise)	Decimal	0.5	0.1	0.2	1.2
f_7 (Quartic Function with Noise)	Binary	1.0	0.3	0.6	1.8
f_8 (Generalized Schwefel's Function 2.26)	Decimal	1.0	0.4	2.0	1.4
f_8 (Generalized Schwefel's Function 2.26)	Binary	0.3	0.1	0.8	2.0
f_9 (Generalized Rastrigin's Function)	Decimal	1.0	0.2	0.2	1.0
f_9 (Generalized Rastrigin's Function)	Binary	0.4	0.2	2.0	1.0
f_{10} (Akley's Function)	Decimal	0.6	0.3	0.2	1.4
f_{10} (Akley's Function)	Binary	0.3	0.1	0.4	1.8
f_{11} (Generalized Griewank Function)	Decimal	0.8	0.4	0.2	1.4
f_{11} (Generalized Griewank Function)	Binary	0.4	0.1	0.6	1.6
f_{12} (Generalized Penalised Function 1)	Decimal	0.5	0.3	0.6	0.2
f_{12} (Generalized Penalised Function 1)	Binary	0.5	0.2	0.6	0.8
f_{13} (Generalized Penalised Function 2)	Decimal	0.9	0.6	0.2	1.6
f_{13} (Generalized Penalised Function 2)	Binary	0.3	0.1	0.4	1.6
f_{14} (Shekel's Foxholes Function)	Decimal	0.8	0.3	1.0	0.2
f_{14} (Shekel's Foxholes Function)	Binary	0.7	0.4	1.0	1.2
f_{15} (Kowalik's Function)	Decimal	0.7	0.7	0.2	1.2
f_{15} (Kowalik's Function)	Binary	0.8	0.3	1.0	1.8
f_{16} (Six-Hump Camel-back Function)	Decimal	1.0	0.8	0.8	0.4
f_{16} (Six-Hump Camel-back Function)	Binary	0.6	0.4	1.0	1.6
f_{17} (Branin Function)	Decimal	0.4	0.5	0.6	1.4
f_{17} (Branin Function)	Binary	0.4	0.3	0.2	2.0
f_{18} (Goldstein Price Function)	Decimal	1.0	0.8	0.8	0.2
f_{18} (Goldstein Price Function)	Binary	0.4	0.2	1.0	1.0
f_{19} (Hartman's Family Function 1)	Decimal	0.9	0.3	0.2	1.8
f_{19} (Hartman's Family Function 1)	Binary	0.9	0.2	1.0	1.6
f_{20} (Hartman's Family Function 2)	Decimal	0.9	0.6	0.2	1.8
f_{20} (Hartman's Family Function 2)	Binary	0.3	0.1	1.0	1.6
f_{21} (Shekel's Family Function 1)	Decimal	0.7	0.1	0.2	2.0
f_{21} (Shekel's Family Function 1)	Binary	0.4	0.1	1.0	1.4

Table 6 continued

Problem	Encoding	P_C	P_M	W	C
f_{22} (Shekel's Family Function 2)	Decimal	0.9	0.7	0.6	1.6
f_{22} (Shekel's Family Function 2)	Binary	0.4	0.1	1.0	1.6
f_{23} (Shekel's Family Function 3)	Decimal	1.0	0.8	2.0	1.2
f_{23} (Shekel's Family Function 3)	Binary	0.5	0.1	0.2	0.4
EKP (Extended Knapsack Problem)	Binary	0.7	0.2	1.0	1.4
TSP (Travelling Salesman Problem)	Integer	1.0	0.5	1.4	0.8
IKP (Inverse Kinematics Problem)	Binary	0.9	0.2	1.2	0.4

The SGA, PSO, and HPGA algorithms are used to solve this problem with the following conditions:

1. For the manipulator motion to be smooth, it is crucial that the end-effector pose does not violate the defined path; therefore, the fitness should not be larger than a certain value, which depends on its application.
2. To achieve real-time behavior for the algorithms, the termination condition is considered 300 ms of execution time for each point.
3. To ensure the validity of the results, the path is tracked 100 times by all three algorithms.

For each algorithm, five values: the maximum position error, maximum orientation error, average position error, average orientation error, and average evaluated chromosome count are saved for the 100 executions. Table 5 shows these values.

As the results show, HPGA is able to achieve shorter runtimes and better results than PSO or SGA. Figure 9 shows the manipulator's desired path and Figs. 10, 11, 12 and 13 show the position and orientation error of the path tracked by the manipulator for each point.

3.5 Optimum parameters of SGA and SPSO

Table 6 shows the list of optimum values for the SGA and SPSO input parameters for the problems used in this paper.

To find the optimum parameter values of SGA, 11 different values (0, 0.1, 0.2, … , 1) for P_C and P_M were combined, producing 121 different sets of SGA parameters to test. Each problem was solved using SGA with these 121 different parameters and according to the results the optimum values of SGA parameters for each problem were chosen. For the SPSO parameters, the same approach was used, 31 different values (0, 0.1, 0.2, … , 3) were tested for C and W.

3.6 Permutation bacterial conjugation operator

Equalizing the specified gene sequence of the recipient chromosome with the donor one is the main purpose of the gene transfer mechanism. To achieve this purpose in

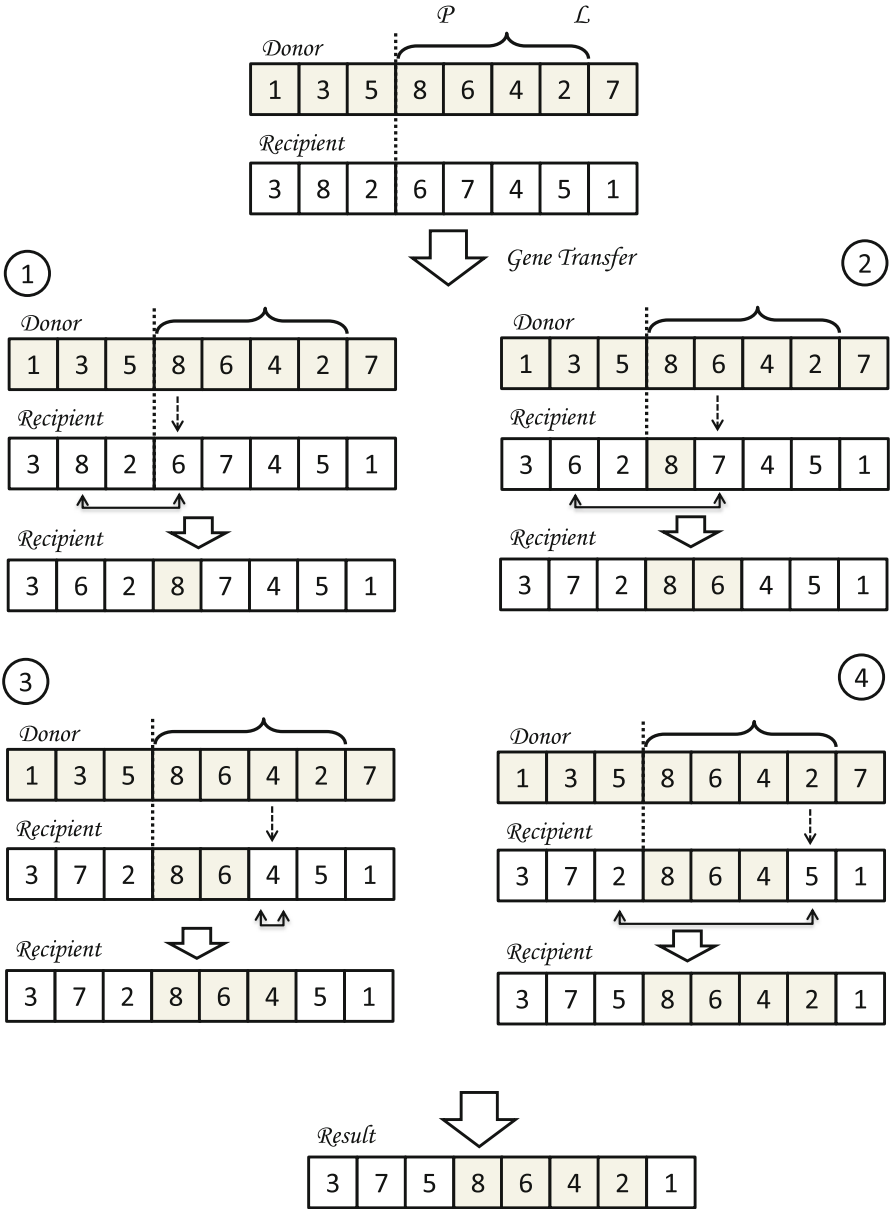

Fig. 14 An example of the gene transfer mechanism for permutation problems

permutation problems, the following procedure is used. Assuming the ith gene of the donor chromosome should be copied to the recipient—the value of ith genes in donor and recipient chromosomes are G_D^i and G_R^i receptively—and, considering the value of the jth gene in the recipient chromosome is G_D^i, the value of the ith and jth

genes will be swapped. By applying this procedure to the whole sequence of L genes, the genes' values in the recipient and the donor chromosomes, from position P with length L, will be equal, while keeping the permutation condition of the recipient chromosome untouched. Figure 14 shows the application of the permutation gene transfer mechanism on two sample chromosomes.

4 Conclusions

Considering the results obtained from the proposed algorithm, compared to SGA and PSO, and considering the fact that the implementation of the proposed algorithm is simpler and its runtime is faster than SGA and PSO, it is an appropriate approach for real-time applications, and a suitable alternative to SGA. In addition, the higher performance and speed of the proposed algorithm makes it a good candidate for solving pseudo real-time problems.

Having the population size as the only parameter of the proposed algorithm, and its linear dependence with respect to the total order of the algorithm execution, approximates the overall costs of the proposed algorithm to those of parameter-less EAs. Consequently, the proposed algorithm can act as a complete self-adaptive algorithm.

Acknowledgments The authors would like to thank the anonymous referees for their helpful comments and suggestions to improve the paper. Also the authors would like to thank their partners in the University of Tabriz for their cooperation in preparing this paper.

References

1. L. Huang, L.-x. Ding, W.-w. Du, Improved self-adaptive genetic algorithm with varying population size, in *Proceedings of the 2009 Fifth International Conference on MEMS NANO, and Smart Systems (ICMENS '09)* (2009), pp. 77–79
2. R. Breukelaar, T. Baeck, Self-adaptive mutation rates in genetic algorithm for inverse design of cellular automata, in *Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO '08)* (2008), pp. 1101–1102
3. A. Mehrafsa, A. Sokhandan, G. Karimian, A Timed-based approach for genetic algorithm: theory and applications. IEICE Trans. Inf. Syst. **E94-D**(6), 1306–1320 (2011)
4. A.J.F. Griffiths, J.H. Miller, D.T. Suzuki, R.C. Lewontin, W.M. Gelbart, An Introduction to Genetic Analysis, 8th edn. (W. H. Freeman, New York, 2004)
5. I. Harvey, The Microbial Genetic Algorithm, Springer, Lecture Notes in Computer Science, vol. 5778 (2011), pp. 126–133
6. P. Smith, Conjugation—a bacterially inspired form of genetic recombination, in *Proceedings of Late Breaking Papers at the Genetic Programming Conference* (1996), pp. 167–176
7. C. Perales-Gravan, R. Lahoz-Beltra, An AM radio receiver designed with a genetic algorithm based on a bacterial conjugation genetic operator. IEEE Trans. Evol. Comput. **12**, 129–142 (2008)
8. N.E. Nawa, T. Furuhashi, Fuzzy system parameters discovery by bacterial evolutionary algorithm. IEEE Trans. Fuzzy Syst. **7**, 608–616 (1999)
9. S. Das, A. Chowdhury, A. Abraham, A bacterial evolutionary algorithm for automatic data clustering, in *Proceedings of the Eleventh conference on Congress on Evolutionary Computation* (2009), pp. 2403–2410
10. X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster. IEEE Trans. Evol. Comput. **3**(2), 82–102 (1999)

11. M. Angelova, T. Pencheva, Tuning genetic algorithm parameters to improve convergence time. Int. J. Chem. Eng. **2011**, 1–7 (2011)
12. PSC. Particle Swarm Central, http://www.particleswarm.info
13. M. Clerc, Standard Particle Swarm Optimisation: From 2006 to 2011, Particle Swarm Central (2011)
14. M. Clerc, Beyond standard particle swarm optimisation. Int. J. Swarm Intell. Res. (IJSIR) **1**(4), 46–61 (2010)
15. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
16. H. Asada, J.-J. E. Slotine, *Robot Analysis and Control*, 1st edn. (John Wiley & Sons, New York, 1986)
17. Y. Yang, G. Peng, Y. Wang, H. Zhang, A new solution for inverse kinematics of 7-DOF manipulator based on genetic algorithm, in *Proceedings of the IEEE International Conference on Automation and Logistics* (2007), pp. 1947–1951