SVPCGA: Selection on Virtual Population based Compact Genetic Algorithm

Yi Hong, Sam Kwong, Hanli Wang, Zhihui Xie, and Qingsheng Ren

Abstract— **This paper describes a novel virtual population based truncation selection operator that extends our previously proposed virtual population based tournament selection operator [1]. Moreover, two extensions of compact genetic algorithm (CGA) that make use of virtual population based selection operators are presented in this paper: one is the tournament selection on virtual population based compact genetic algorithm (SVPCGA-TO); the other is the truncation selection on virtual population based compact genetic algorithm (SVPCGA-TR). Both SVPCGA-TO and SVPCGA-TR are tested on several benchmark problems and their results are compared with those obtained by CGA [2] and ne-CGA [3]. Some superiorities of SVPCGA in search reliability can be achieved.**

I. Introduction

Genetic algorithms (GAs) are a class of population based search methods that loosely mimic the behavior of Darwinian evolution. Unlike hill-climbing search methods where only one candidate solution is used, GAs maintain a population of candidate partial solutions during their search. Therefore, GAs have the ability of jumping over local optimal solutions and may converge to the global optimal one. GAs have gained a wide range of real-world applications. However, it has also been known that the performances of GAs are quite sensitive to their parameter settings. In order to obtain a high-quality solution, several parameters of GAs should be chosen with care. Among them is the population size. If the population size is too small, GAs may not be able to explore enough of the search space for consistently identifying good candidate solutions. Whereas a large population size usually leads to a slow convergence and a high memory cost. In addition, it is a trivial task to set an appropriate population size that is able to draw a good balance between convergent reliability and convergent velocity.

In recent years, there is another tendency in the area of evolutionary computation that directly removes the population from GAs, but maintaining their population based search abilities of GAs. This kind of GAs is commonly known as the compact genetic algorithm (CGA), firstly proposed by Harik et al. in [2]. CGA tries to mimic the behavior of GAs without storing the whole population in the memory explicitly and its operation is considered equivalent to the operation of GAs that does not assume any linkages among variables.

Yi Hong, Sam Kwong and Hanli Wang are with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong (email: {yihong, CSSAMK}@cityu.edu.hk, wanghl@cs.cityu.edu.hk). Zhihui Xie is with the Department of Mathematics, Qingsheng Ren is the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, P.R. China. (email: ren-qs@cs.sjtu.edu.cn).

This work was supported by City University of Hong Kong, Research grant 114707.

CGA has the advantage of solving problems under the condition that lacks of memory. However, the convergent reliability of CGA sometimes is not satisfactory enough due to its low selection pressure and its assumption that variables are independent. In order to increase the selection pressure and to improve the performance of CGA, Ahn and Ramakrishna proposed the Elitism-Based Compact Genetic Algorithm (ne-CGA)[3]. In this paper, we extend our previous work on the virtual population based tournament selection and thus introduce a virtual population based truncation selection operator [1]. Moreover, we propose two types of CGA that make use of virtual population based selection operators: one is the tournament selection on virtual population based compact genetic algorithm (SVPCGA-TO); the other is the truncation selection on virtual population based compact genetic algorithm (SVPCGA-TR). Both SVPCGA-TO and SVPCGA-TR are tested on several benchmark problems and their results are compared with those obtained by CGA and ne-CGA.

The rest of this paper is arranged as follows. Section II introduces the virtual population based selection operators. Section III goes into details of describing the selection on virtual population based compact genetic algorithm. Experimental results and their analysis are given in section IV. Section V concludes this paper.

II. Virtual population based selection operators

In GAs, new candidate solutions are reproduced by using crossover and mutation operators. Among them, the crossover operator combines building blocks contained in two different solutions to form new better ones, while the mutation operator reproduces new candidate solutions by perturbing old ones. It is noted that both crossover and mutation operators are executed on the whole population. Therefore, the population can not be removed from traditional GAs. However in Estimation of Distribution Algorithms (EDAs), new candidate solutions are generated through sampling from the density of promising solutions. Generic EDAs finish the search task with the following steps employed [4] [5]:

Algorithm 1 Estimation of distribution algorithms.

(1) $P_1 \leftarrow$set the initial density, $t = 1$;
(2) $G \leftarrow$generate M solutions through sampling from P_t;
(3) $f \leftarrow$calculate fitness values of solutions;
(4) $G^{Se} \leftarrow$select $N (N < M)$ promising solutions from G;
(5) $P_{t+1} \leftarrow$estimate the density of solutions in G^{Se};
(6) $t = t + 1$, return to (2).

978-1-4244-1823-7/08/$25.00©2008 IEEE

It can be observed from Algorithm 1 that EDAs completely abandon crossover and mutation operators from GAs. Therefore, only the selection operator in EDAs is directly executed on the whole population G. This interesting characteristic of EDAs has motivated us to design the virtual population based selection operators, where promising individuals are selected from a virtual population represented as a density P_t. EDAs that use virtual population based selection operators instead of real population based selection operators have the advantage of being executed under a small memory consumption. Therefore, EDAs that use virtual population based selection operators are more suitable for solving many memory limited optimization problems when compared with EDAs that use real population based selection operators.

A. Virtual population based tournament selection operator

Let k denote the tournament size and N be the size of the selected subpopulation G^{Se}. The real population based tournament selection operator randomly chooses k solutions from the population G with replacement and lets them compete, then the winner is stored into the selected subpopulation G^{Se}. The above steps iterate for N rounds and the selected subpopulation G^{Se} that contains N selected solutions can be obtained.

Unlike the real population based tournament selection operator, the virtual population based tournament selection operator does not store the whole population G in the memory. It constructs the selected subpopulation G^{Se} with the following steps employed: to generate k solutions by sampling from the density P_t and let them compete; then the winner is stored into the selected subpopulation G^{Se}, while the losers are directly removed from the memory. The above steps iterate for N rounds and the selected subpopulation G^{Se} that contains N solutions can be obtained. Full steps of the virtual population based tournament selection are described in Algorithm 2. More details about the virtual population based tournament selection can be consulted from [1].

Algorithm 2 Virtual population based tournament selection (the tournament size is equal to 2).

(1) $s = 1$;
(2) $a, b \leftarrow$generate two solutions through sampling from the density P_t;
(3) $(w, l) \leftarrow$let a and b compete, then the winner is denoted as w and the loser as l;
(4) $G^{Se} \leftarrow w$ is stored into G^{Se} and l is directly removed from the memory, $s = s + 1$;
(5) go to (2), if $s \leq N$.

B. Virtual population based truncation selection operator

Herein, a novel virtual population based truncation selection operator is proposed. The real population based truncation selection operator firstly ranks all solutions in the population G according to their fitness values, then selects $N(N < M)$ solutions with the highest fitness values to construct the selected subpopulation G^{Se}. Provided that only one solution is replaced at each iteration, then the truncation selection process can be described as follows: to generate one solution a through sampling from the density P_t and calculate its fitness value $f(a)$. If $f(a)$ is larger than that of the N^{th} best solution in the population G, then replace the N^{th} best solution in the population G with the new individual a. In this case, the density P_{t+1} should be recalculated, because the selected subpopulation G^{Se} has changed. Otherwise, no change occurs on the selected subpopulation G^{Se}, therefore the density P_{t+1} is the same as the density P_t.

Algorithm 3 Virtual population based truncation selection.

(1) $\beta = 0, s = 0, v_m = \{v_1, v_2, ..., v_{N'}\}$;
(2) $a \leftarrow$generate one solution through sampling from the density P_t;
(3) if $v_m \leq f(a)$
  a) $G^{Se} \leftarrow$the solution a is stored, $s = s + 1$;
  b) $v_m \leftarrow f(a)$;
  c) $v_m = \{v_1, v_2, ..., v_{N'}\}$;
  else $\beta = \beta + 1$;
(4) if $\beta \geq \eta$, then to update the float vector V by generating N' solutions through sampling from P_t and their fitness values are stored into V and $\beta = 0$;
(5) go to (2), if $s \leq N$.

Our proposed virtual population based truncation selection operator utilizes the above processes without storing the whole population G in the memory. In particular, a float vector V is used to store fitness values of the N' best solutions in the previous generation, that is:

$$V = \{v_1, v_2, ..., v_{N'}\}$$

Let v_m denote the minimal value of elements in V, that is:

$$v_m = \min\{v_1, v_2, ..., v_{N'}\}$$

Then one solution a is generated through sampling from the density P_t. If its fitness value is higher than v_m, then the solution a is stored into the selected subpopulation G^{Se}. At the same time, the element v_m in the vector V is replaced by $f(a)$. The above steps iterate until N solutions have been selected into the subpopulation G^{Se}. One problem associated with the above approach is when v_m in the float vector V is much larger than the fitness values of most solutions sampled from the density P_t, the number of solutions that will be able to update the density P_t becomes less. In this case, the evolutionary process of EDAs tends to be stagnant and the search of EDAs becomes like the random walk.

To mitigate the above problem, we set an integer β to count the number of the solutions whose fitness values are lower than v_m in each selection round. If β is larger than a predefined value η, then the float vector V is updated as follows: to generate N' solutions through sampling from the density P_t and their fitness values are stored into the float vector V one by one. Full steps of the virtual population

TABLE I
MEMORY CONSUMPTIONS OF CGA, NE-CGA AND SVPCGA.

	CGA	NE-CGA	SVPC.-TR	SVPC.-TO
SOLUTIONS	2	1	2	1
FLOAT NUM.	2	2	2	$N'+1$

based truncation selection are described in Algorithm 3. In Algorithm 3, the float vector V is initialized as the fitness values of N' random solutions:

$$v_i = f(b_i) \qquad i = 1, 2, ..., N'$$

where $\{b_1, b_2, ..., b_{N'}\}$ are random solutions that are successively generated[1].

It is worthwhile mentioning that our proposed virtual population based truncation selection operator extends the elitism based selection operator in the following two facets [3]: (1) The elitism in the virtual population based truncation selection operator is the minimal value of a float vector that contains $N(N \gg 1)$ highest fitness values, while the elitism in the elitism-based selection operator is a float number that equals to the fitness value of the best individual. (2) If a stagnancy of evolution is met, the elitism-based selection operator replaces the elitism by the fitness value of a solution that is generated at random, while the updating strategy in the virtual population based truncation selection operator is guided by the density P_t. The above two differences guarantee that the virtual population based truncation selection operator is more flexible and efficient than the elitism-based selection operator.

Algorithm 4 Tournament selection on virtual population based compact genetic algorithm (SVPCGA-TO).

(1) to initialize the density P_1 and $t = 1$;
(2) $e = \overrightarrow{0}$;
(3) to learn the activation e as follows:
  a) $s = 1$;
  b) $a, b \leftarrow$ to generate two solutions through sampling from density P_t;
  c) $(w, l) \leftarrow$ to let a and b compete, the winner is denoted as w and the loser as l;
  d) to remove l from the memory;
  e) $e = \frac{s-1}{s} \cdot e + \frac{1}{s} \cdot w$;
  f) to remove w from the memory;
  g) $s = s + 1$, then go to b) until $s > N$;
(4) to update the density P_t by the activation e as:
  $P_{t+1} = (1 - \lambda) \cdot P_t + \lambda \cdot e$;
(5) if the stop condition is not met, $t = t + 1$, go to (2).

[1]In this paper, "successively generated" means the solution will be removed from the memory after their fitness value is calculated and stored.

Algorithm 5 Truncation selection on virtual population based compact genetic algorithm (SVPCGA-TR).

(1) to initialize the density P_1 and $t = 1$;
(2) $e = \overrightarrow{0}$;
(3) to learn the activation e as follows:
  1) $s = 1$ and $\beta = 0$;
  2) $w \leftarrow$ to generate one solution w through sampling from density P_t;
  3) if $v_m \leq f(w)$
    (a) $e = \frac{s-1}{s} \cdot e + \frac{1}{s} \cdot w$;
    (b) $v_m \leftarrow f(w)$;
    (c) $v_m = \min\{v_1, v_2, ..., v_{N'}\}$;
  else $\beta = \beta + 1$;
  4) to remove l from the memory;
  5) if $\beta \geq \eta$, to successively generate N solutions through sampling from P_t and the float vector V is updated by their fitness values and $\beta = 0$;
  6) to remove w from the memory;
  7) $s = s + 1$, then go to (2) until $s > N$;
(4) to update the density P_t by the activation e:
  $P_{t+1} = (1 - \lambda) \cdot P_t + \lambda \cdot e$;
(5) if the stop condition is not met, $t = t + 1$, go to 3.

III. SELECTION ON VIRTUAL POPULATION BASED COMPACT GENETIC ALGORITHM

In this section, we describe two extensions of compact genetic algorithm that make use of virtual population based selection operators. The main idea of these two extensions is to combine virtual population based selection operators with an incremental learning process. The incremental learning process updates the density P_t by the following rule:

$$P_{t+1} = P_t + \lambda \times (e - P_t) \tag{1}$$

where $\lambda (0 < \lambda \leq 0)$ is the learning rate and e is an activation vector that is learned from the selected solutions in the t^{th} generation. Provided that $w_1, w_2, ..., w_N$ are the N solutions that are successively selected in the t^{th} generation. In order to learn the activation e without storing the solutions $w_1, w_2, ..., w_N$ in the memory, another incremental learning process is used as follows:

$$e = \frac{s-1}{s} \cdot e + \frac{1}{s} \cdot w_s \tag{2}$$

where $s = 1, 2, ..., N$ and the initial value of e is set as $\overrightarrow{0}$. Frameworks of tournament selection on virtual population based compact genetic algorithm (SVPCGA-TO) and truncation selection on virtual population based compact genetic algorithm (SVPCGA-TR) are given in Algorithm 4 and Algorithm 5. The stop condition is set as the maximal number of fitness evaluations. Memory consumptions of CGA, ne-CGA and SVPCGA are given in Table 1, where "solutions" represents the number of solutions that are stored in the memory and "float num." denotes the number of float numbers that are used to store the fitness values.

IV. Experimental results and analysis

The proposed SVPCGA-TO and SVPCGA-TR are tested on six benchmark problems (see Appendix) and their results are compared with those obtained by CGA and ne-CGA. Parameter settings in our experiments are given as follows: the learning rate λ is set as 0.01 for ne-CGA and SVPCGA, the tournament size for CGA and SVPCGA-TO equals to 2, the threshold value η to update the elitism or the float vector is set as 30 for SVPCGA-TR and ne-CGA, the number of solutions N selected at each generation is fixed to 30 for SVPCGA and the length N' of the vector V is fixed to 100 for SVPCGA-TR.

First, we test and compare the convergent reliability of CGA, ne-CGA and SVPCGA under different numbers of variables through calculating the difference between the converged solutions and the optimal solutions. The results are shown in Figure 1. From Figure 1, we can observe that the performance of CGA is not satisfactory enough for all tested six benchmark problems. For example, the differences between the converged value of the Weighed One-Max Problem and the global optimal value of the Weighed One-Max Problem is as large as 5900, when the number of variables increases up to 1000. One method to improve the reliability of CGA is to decrease the learning rate λ from 0.01 to 0.001. However, experimental results let us know that this improvement is sometimes limited. To take the Max-Min problem as an example, the difference between the converged result and the optimal result still comes up to 3.0, even if our learning rate is set as small as 0.001. ne-CGA outperforms CGA on all tested six benchmark problems. However, the solutions captured by ne-CGA still significantly deviate from the global optimal solutions. The reliability of SVPCGA are much better than those of CGA and ne-CGA, particular when the number of the variables are larger than 500. For all tested problems, both SVPCGA-TO and SVPCGA-TR are able to identify high-quality solutions, even if the number of variables is 1000.

Second, we test and compare the convergent velocity of CGA, ne-CGA and SVPCGA. The results are shown in Figure 2. From Figure 2, we can see that SVPCGA requires much more fitness evaluations to converge when compared with CGA and ne-CGA. Another phenomenon observed from Figure 2 is that SVPCGA-TR converges faster than SVPCGA-TO.

In addition, the effects of the learning rate on performances of SVPCGA are also tested. Figure 3 and Figure 4 give the results. Figure 4 lets us know that a large value of the learning rate usually leads to a fast convergence, but a poor convergent result.

V. Conclusions

This paper has proposed the virtual population based truncation selection operator that extended our previous work on the virtual population based tournament selection. In addition, we have introduced two novel compact genetic algorithms that make use of virtual population based selection operators: one is the tournament selection on virtual population based compact genetic algorithm (SVPCGA-TO) and the other is the truncation selection on virtual population based compact genetic algorithm (SVPCGA-TR). The superiority of SVPCGA-TR and SVPCGA-TO in convergent reliability over CGA and ne-CGA has been demonstrated by experimental results on several benchmark problems under different numbers of variables.

VI. Appendix

The first benchmark problem is the Weighed One-Max:

$$\max\left(\sum_{i=1}^{n} i \cdot x_i\right) \qquad x_i \in \{0,1\}$$

The second benchmark problem is the Quadratic Problem:

$$\max\left(\sum_{i=1}^{n} f(x_{2i-1}, x_{2i})\right) \qquad x_{2i-1}, x_{2i} \in \{0,1\}$$

The third benchmark problem is the Bit Layout Problem:

$$\max\left(\sum_{i=2}^{n-1} f(x_{i-1}, x_i, x_{i+1})\right) \qquad x_{i-1}, x_i, x_{i+1} \in \{0,1\}$$

where $f(\cdot)$ equals 1 if and only if x_i is different from both x_{i-1} and x_{i+1}, otherwise it equals 0.

The fourth benchmark problem is the Max-Min Problem:

$$\max\min\{u_1, u_2, ..., u_n\}$$

where $u_i = \sum_{j=1}^{20} x_{ij}, x_{ij} \in \{0,1\}$

The fifth benchmark problem is the Noisy Optimization:

$$\max\left(\sum_{i=1}^{n} x_i + N(0,1)\right) \qquad x_i \in \{0,1\}$$

where $N(0,1)$ is the noise that satisfies normal distribution.

The sixth benchmark problem is the Satisfaction Problem:

$$\max\left(\sum_{i=1}^{n} f(x_{5i-4}, x_{5i-3}, x_{5i-2}, x_{5i-1}, x_{5i})\right) \quad x_i \in \{0,1\}$$

where $f(x_{5i-4}, x_{5i-3}, x_{5i-2}, x_{5i-1}, x_{5i})$ equals 5 if and only if all variables are 1, otherwise it equals 0.

References

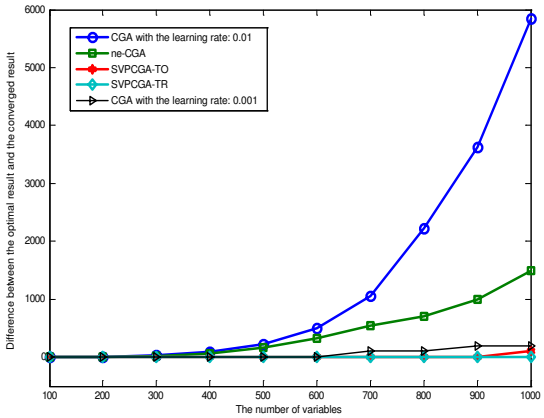
[1] Y. Hong, S. Kwong, Q. Ren and X. Wang, "A comprehensive comparison between real population based tournament selection and virtual population based tournament selection," *IEEE Congress on Evolutionary Computation*, pp. 445-452, 2007.

[2] G.R. Harik, F.G. Lobo and D.E. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 287-297, 1999.

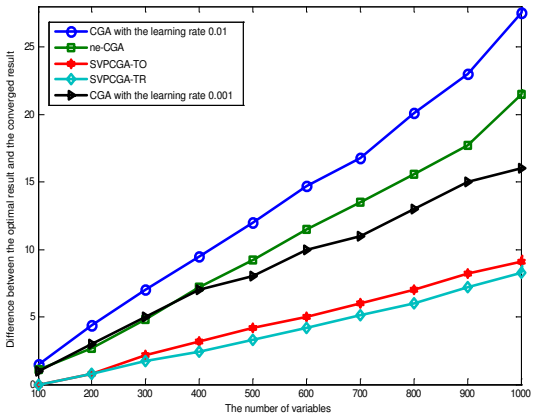
[3] C.W. Ahn and R.S. Ramahrishna, "Elistism-based compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 367-387, 2003.

[4] P. Larrañage and J.A. Lozano, *Estimation of Distribution Algorithms.* Kluwer Academic Publisher. 2002.

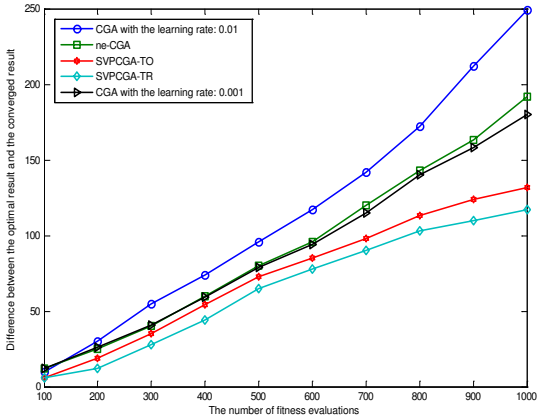
[5] Q. Zhang and H. Mühleubein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 127-136, 2004.



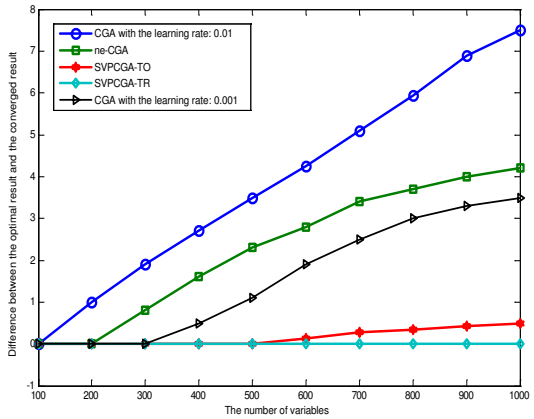
(a) Weighed One-Max Problem



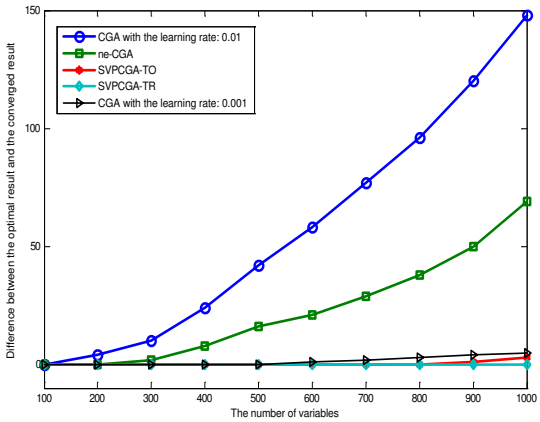
(b) Quadratic Problem



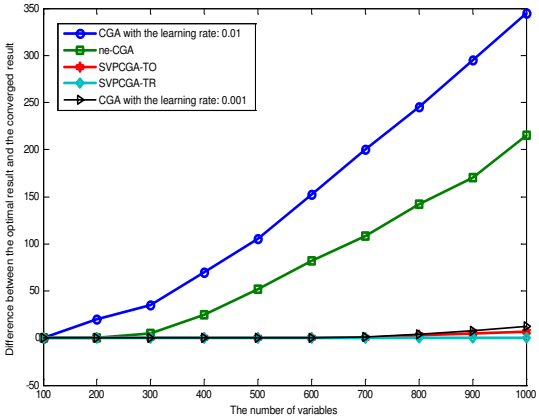
(c) Bit Layout Problem



(d) Max-Min Problem

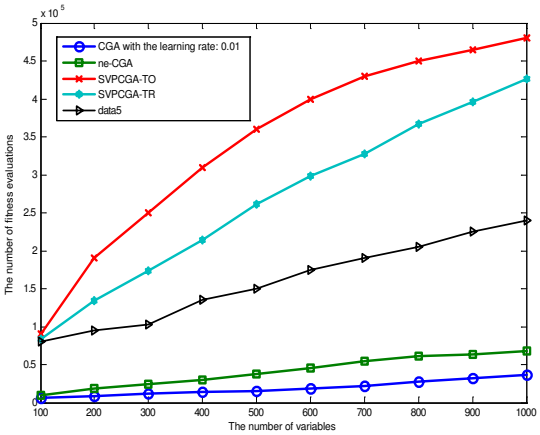


(e) Noisy Optimization Problem

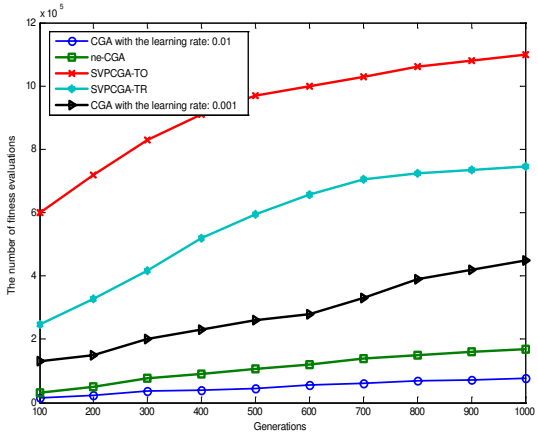


(f) Satisfaction Problem

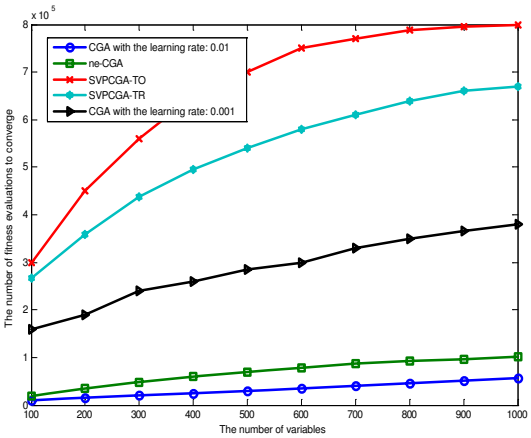
Fig. 1. Convergent reliability. All algorithms are executed for 20 independent runs. Their results are averaged and reported.



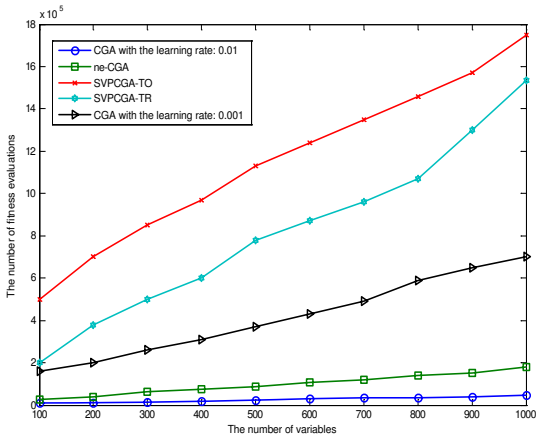
(a) Weighed One-Max Problem



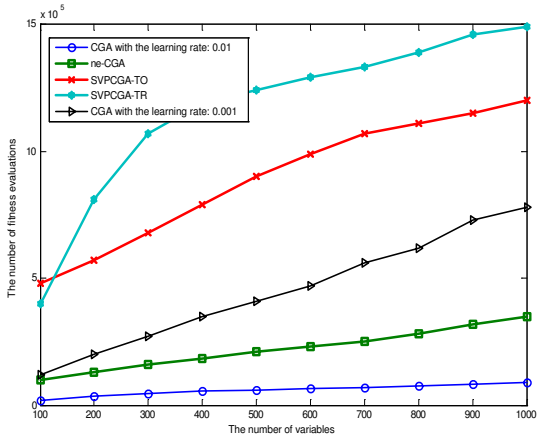
(b) Quadratic Problem



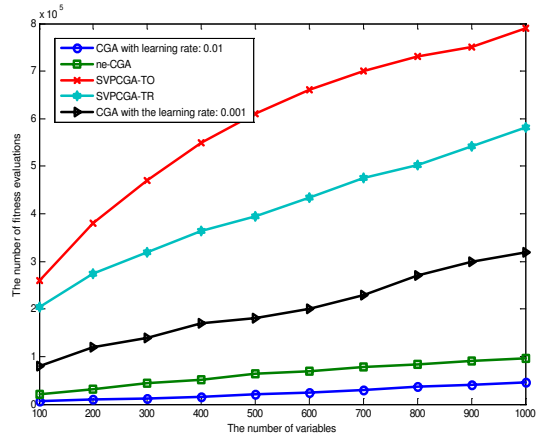
(c) Bit Layout Problem



(d) Max-Min Problem

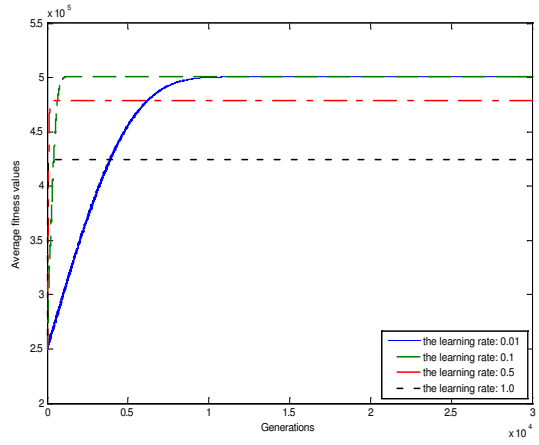


(e) Noisy Optimization Problem

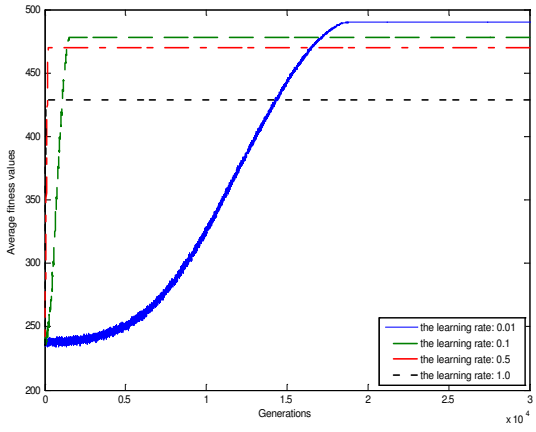


(f) Satisfaction Problem

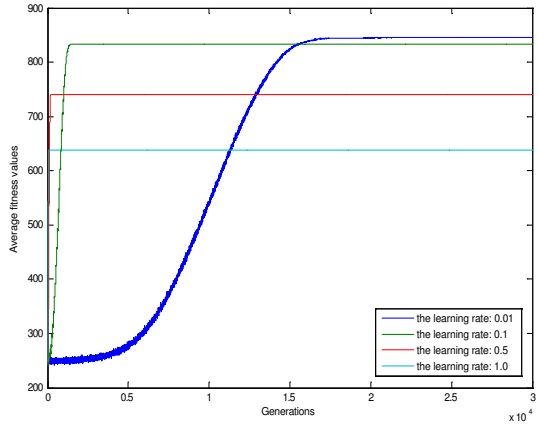
Fig. 2. Convergent velocity. All algorithms are executed for 20 independent runs. Their results are averaged and reported.



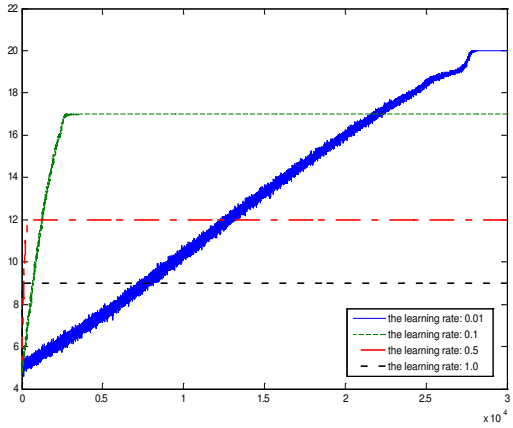
(a) Weighed One-Max Problem



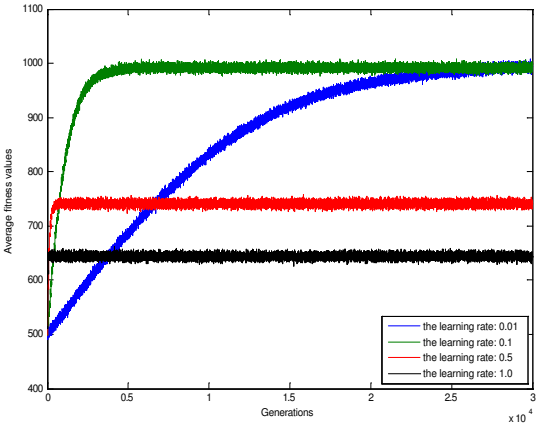
(b) Quadratic Problem



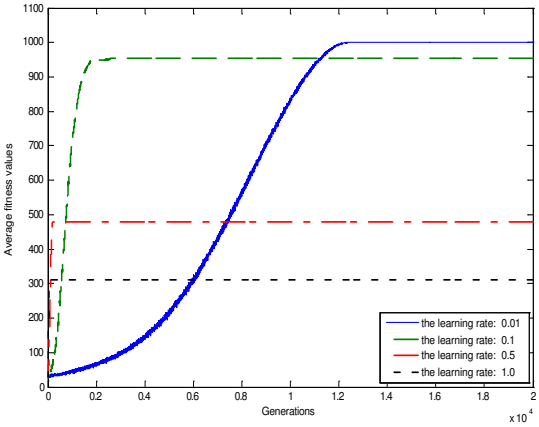
(c) Bit Layout Problem



(d) Max-Min Problem

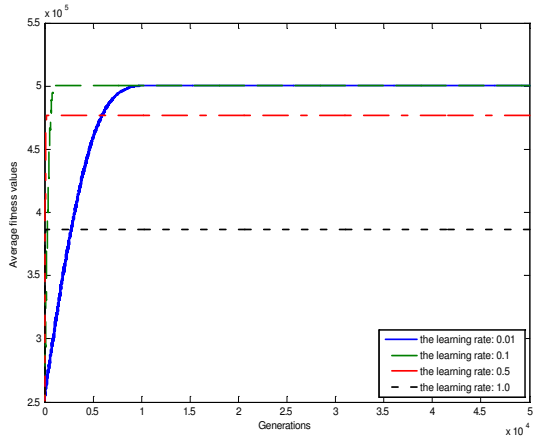


(e) Noisy Optimization Problem

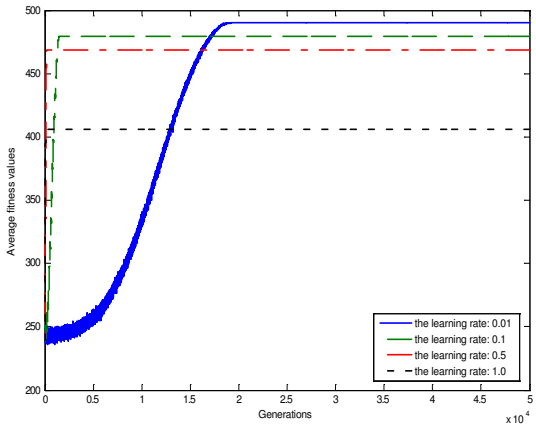


(f) Satisfaction Problem

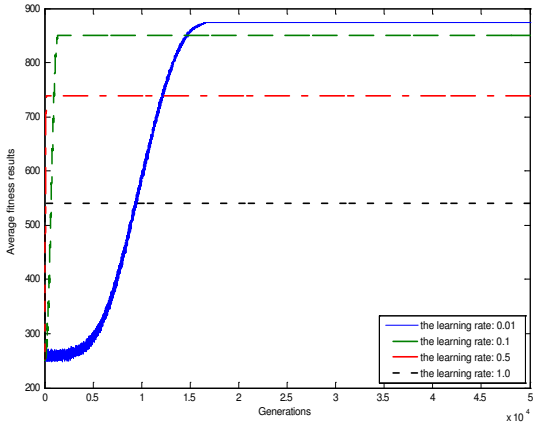
Fig. 3. Effect of the learning rate on SVPCGA-TO. All algorithms are executed for one time and their results are reported.



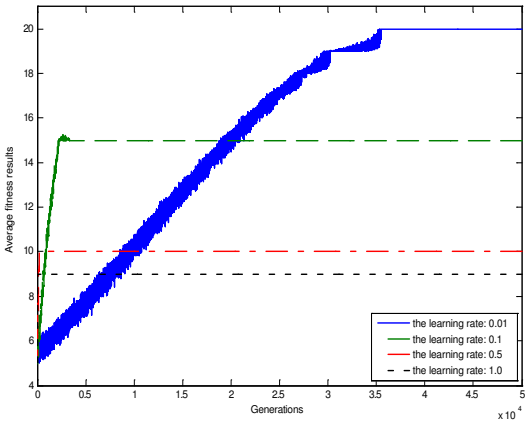
(a) Weighed One-Max Problem



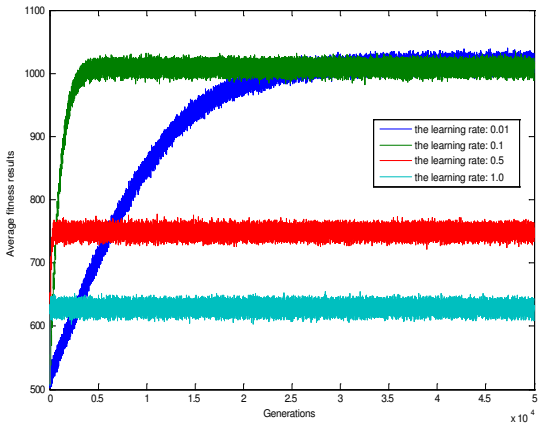
(b) Quadratic Problem



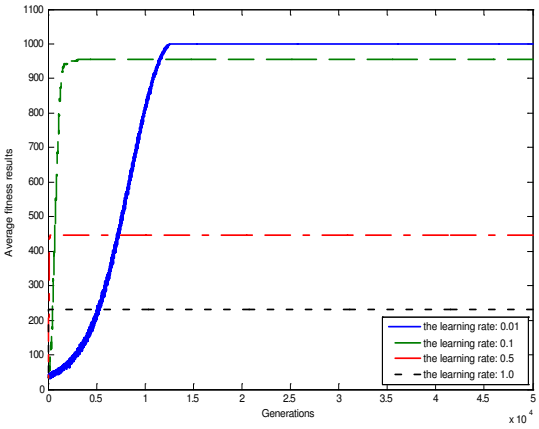
(c) Bit Layout Problem



(d) Max-Min Problem



(e) Noisy Optimization Problem



(f) Satisfaction Problem

Fig. 4. Effect of the learning rate on SVPCGA-TR. All algorithms are executed for one time and their results are reported.