



IMPROVED SELECTION OPERATOR FOR GA

¹Omar Al Jadaan, ²Lakishmi Rajamani, ³C. R. Rao

¹ Dept. CSE, EC. Osmania University, Hyderabad 500-007, INDIA.

² Prof, Dept. CSE, EC. Osmania University, Hyderabad 500-007, INDIA.

³ Prof., Department of CIS, University of Hyderabad, Hyderabad500-046, INDIA.

E-mail: o_jadaan@yahoo.com, Lakshmiraja@yahoo.com, crrcs@uohyd.ernet.in

ABSTRACT

Selection operator is one of the important aspects in the GA process. There are several ways for selection. Some of them are Tournament selection, Ranking selection, and Proportional selection. There are many ways for proportional selection. The most popular are Roulette Wheel Selection (RWS), Stochastic Reminder Roulette Wheel Selection (SRRWS), and Stochastic Universal Sampling (SUS). In this paper a modified RWS method is proposed to increase the gain of resources, reliability and diversity; and decrease the uncertainty in selection process.

Keywords: Elitism, Genetic Algorithms, Selection, Optimization, Robust.

1. INTRODUCTION

John Holland proposed the first Genetic Algorithm (GA) in 1975 [1], and GAs became popularized by the publication of David Goldberg's book in 1989 [2]. Since that time, GAs have been used in a wide range of applications where optimization is needed. GAs are evolved from evolutionary process and based on evolutionary operators like selection, mutation and crossover with the help of survival characteristics through fitness for arriving at the best solutions for specified problems. A modified algorithm of RWS (named ranked based roulette wheel selection RRWS) is presented in this paper. It is faster and robust than RWS.

This paper is organized as follows: Section 2 is an Introduction to GA, Section 3 presents Design of GA with Ranked Based Roulette, Section 4 illustrates the experimental work, followed by Section 5 which discusses the Results, and ends with Section 6 the Conclusions.

2. AN INTRODUCTION TO GA

Genetic Algorithms are among several types of optimization methods that use a stochastic approach to randomly search for good solutions to a specified problem, including Simulated Annealing [3], Tabu search [4], and numerous variations. These stochastic approaches use various analogies to natural systems to build from

promising solutions, ensuring greater efficiency than completely random search. The advantage of using these types of approaches over traditional optimization methods, such as nonlinear programming, is that they can solve any type of problem without explicit specifications of problem characteristics (e.g. derivatives of the objective function). This property is particularly important for complex applications, where the optimization problem often involves integer decision variables or potential solutions that must be evaluated with complex existing simulation models, where derivative calculations would be difficult or impossible. Stochastic approaches also perform broad global search, unlike traditional nonlinear optimization approaches such as nonlinear programming that can converge to local minima in multimodal optimization problems. These benefits have resulted in widespread use and acceptance of these approaches among the researchers. The disadvantages of these approaches are that they are not guaranteed to find the globally optimal solution and they can be substantially slower than traditional optimization methods for problems that can be solved using traditional approaches. Therefore, they recommend the use of these methods only for problems that cannot be effectively solved using traditional optimization approaches, such as those with numerous integer decision variables, non-convexities, or other irregularities. When applied to such problems, these algorithms have



demonstrated to find substantially better solutions than could be found using traditional search algorithms. Numerous types and variations of genetic algorithms exist, but this paper will provide details of the most widely used approach called the binary simple genetic algorithm (SGA). The basic operations of an SGA, First, the decision variables are encoded into binary form, called a “chromosome” (or sometimes “string”) because it gives the genetic encoding (“genes” or “bits”) describing each potential solution. Next, an initial “population” of potential solutions is created, usually by filling a set of chromosomes (population “members”/“individuals”) with random initial values. Each member of the population is then evaluated to see how well it performs (i.e., its “fitness”) with respect to the user-specified objective function and constraints (“fitness function”). Then the population is transformed into a new population (the next “generation”) using three primary operations: selection, crossover, and mutation. A fourth operator, elitism, is also usually included to ensure that good solutions are not lost from one generation to the next. This transformation process from one generation to the next continues until the population converges to the optimal solution, which usually occurs when a certain percentage of the population (e.g., 90%) has the same optimal chromosome. For more details on the GA process can be found in Goldberg [2].

A. Selection (or Reproduction)

The selection operator involves randomly choosing members of the population to enter a mating pool. The operator is carefully formulated to ensure that better members of the population (with higher fitness) have a greater probability of being selected for mating, but that worse members of the population still have a small probability of being selected. Having some probability of choosing worse members is important to ensure that the search process is global and does not simply converge to the nearest local optimum. Selection is one of the important aspects of the GA process, and there are several ways for the selection: some of these are Tournament selection, Ranking selection, and Proportional selection. In the proportional selection a string is selected for the mating with a probability proportional to its fitness. There are many ways of proportional selection: the most popular are Roulette Wheel Selection (RWS), Stochastic Remainder Roulette Wheel Selection (SRRWS), and Stochastic Universal Sampling (SUS).

B. Crossover

Crossover creates a new individual's representation from parts of its parent's representations. During crossover, pairs of chromosomes (parents) are randomly selected from the mating population. With a user-specified crossover probability, P_c , genes from one parent chromosome are swapped with corresponding genes on the other parent chromosome to create two children. When the swap does not occur (probability $1 - P_c$), the two parents are transferred to the child population unchanged. In multi-point crossover, multiple locations on the chromosome are selected for gene exchange, each with probability P_c . The highest amount of exchange occurs during uniform crossover, where every gene has a probability P_c of being exchanged with its corresponding gene on the other parent chromosome.

C. Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation. Once the children are created during crossover, the mutation operator is applied to each child. Each gene has a user-specified mutation probability, P_m , of being mutated. In binary mutation, a value of 0 converted to a value of 1, or vice versa.

D. Elitism

Elitism involves replacing worst chromosomes in the children population with the best members of the parent population. This operator has proved to increase the speed of convergence of the GA because it ensures that the best solution found in each generation is retained. While this operator could be applied more broadly (e.g. retaining the 2 or 3 best solutions or 10% - 20% of the population size) overuse of it can lead to premature convergence to the incorrect solution. Next section will be design of the GA with Ranked based Roulette Wheel.

3. DESIGN OF THE GA WITH RANKED BASED ROULETTE WHEEL

Let us first explain how Roulette wheel selection method.



1) Roulette Wheel Selection

Roulette wheel probabilistically selects individuals based on their fitness values F_i . A real-valued interval, S , is determined as either the sum of the individuals expected selection probabilities $S = \sum P_i$, where $P_i = \frac{F_i}{\sum F_i}$ or the

sum of the raw fitness values $S = \sum F_i$ over all the individuals in the current population. Individuals are then mapped one-to-one into contiguous intervals in the range $[0, S]$. The size of each individual interval corresponds to the fitness value of the associated individual. The circumference of the roulette wheel is the sum of all fitness values of the individuals. The fittest individual occupies the largest interval, whereas the least fit have correspondingly smaller intervals within the roulette wheel. To select an individual, a random number is generated in the interval $[0, S]$ and the individual whose segment spans the random number is selected. This process is repeated until the desired number of individuals has been selected.

2) Ranked Based Roulette Wheel Selection

This paper uses modified roulette wheel selection algorithm where each individual is assigned a fitness value equal to its rank in the population: the highest rank has the highest probability to be selected. The probability is calculated as illustrated in the following equation:

$$P_i = \frac{2 * \text{Rank}}{N_Pop * (N_Pop + 1)}$$

4. EXPERIMENTAL WORK

In this paper, Matlab code has been developed to assess the performance of the ranked roulette wheel selection GA in contrast with the classical roulette wheel selection GA. It considers standard test functions for illustrative proposes.

The study of ranked roulette wheel selection GA is confined to a single point crossover with probability $P_c = 1.00$, binary mutation with probability $Max\left\{\frac{1}{L}, \frac{1}{Pop_size}\right\}$, population

size $Pop_size = 1.4L$, and number of generations $N_gen = 2L$, where L is the length of the chromosome. The same specifications are used

for roulette wheel selection GA with only one different parameter: the roulette wheel selection operator is used. Next section discusses the results.

5. RESULTS

This section presents the results of the comparison between applying the roulette wheel selection GA and ranked based roulette wheel selection GA, by applying them on eight test functions from the GA literature, see table 1.

In this paper, all the results have been gathered by applying the RWS GA and RRWS GA on the test functions, each function with 10 variables, each of them encoded with 10 bits. All the graphs and tables are in appendix A. The graphs X-1 and X-2 for each test function are for a single run, and the graphs from X-3 to X-12 for each test function are for 100 runs. During those runs, the best values and mean ones as well as average values are collected and the performance graphs are plotted using $mean \pm 1.96 \text{ STD}$ and $mean \pm 3 \text{ STD}$, Where STD is the standard deviation.

We will discuss two test functions out of eight because of the space limitation, for more information contacts the authors. Figure 1.1 shows the best, average and poorest results of single run for RWS GA; while figure 1.2 shows the best, average and poorest results of single run for RRWS GA. From the figures, the RRWS steadily converging close to the optimum solution in both best and average graphs whereas the fluctuation is bigger in both the best and average RWS graphs. RRWS hits the optimum value at generation 30 whereas RWS does not hit it. This means the number of trials required to converge to the optimum solution in RRWS is smaller than RWS number. This means by using RRWS GA the average resource saving is $170/200 \approx 85\%$.

Table 2 explains the first hit of the optimum solution for both (RWS and RRWS). From Table 2, RRWS GA solutions dominate RWS GA solutions. From figure 1.3, mean of RRWS GA (ranking) hits the optimum value at generation 40 whereas the RWS GA hits it at generation 180. This means that the number of trials required to converge to the optimum solution in RRWS is smaller than RWS number. This means by using RRWS GA the average resource saving is $140/180 \approx 78\%$. Table 3 explains the first hit of the optimum solution for both (RWS and RRWS) and the average resource saving for the remaining test functions. From Table 3, RRWS GA succeeds 88% (seven out of eight succeeds) in reaching the



optimum solution. From figure 1.4 RRWS GA hits the zero value deviation at generation 50 whereas RWS GA hits it at generation 200. This means that the RRWS GA captures the optimal solution earlier than RWS GA, which, using RRWS GA, indicates that the average resource saving is $150/200 \approx 75\%$. Table 4 explains the first hit of the zero deviation value for both (RWS and RRWS) and the average resource saving for the remaining test functions. From Table 4, RRWS GA succeeds 62.5% (five out of eight succeeds) in reaching the zero deviation value. Figures 1.5 and 1.6 show the performance graphs of RWS and RRWS (by drawing mean ± 1.96 STD) where the RRWS graphs are fit to one graph at generation 40 whereas in RWS graphs do not fit up to 200 generation. This means that 95% of the values (solutions) are close to the optimum solution. And this indicates that RRWS is faster in getting closer to the optimum value than RWS in $160/200 \approx 80\%$.

Table 5 depicts this for the remaining test functions. From Table 5, RRWS GA succeeds 62.5% (five out of eight succeeds) outperforming RWS. Figures 1.7 and 1.8 show the 95% performance graph of average values of RWS and RRWS (by drawing mean ± 1.96 STD). The band created by RRWS is narrower than the band created by RWS, which implies that the fluctuation of RRWS is less than that of RWS. RWS is more uncertain than RRWS; and this is applicable to all the performance graphs of the remaining test functions. Figures 1.9 and 1.10 show the performance graph of RWS and RRWS (by drawing mean ± 3 STD) where the RRWS are fit to one graph at generation 60 whereas in RWS graphs do not fit up to 200 generation. This means that 99% of the values (solutions) close to the optimum solution, which indicate that RRWS is faster in getting closer to the optimum value than RWS in $140/200 \approx 70\%$. Table 6 displays this for the remaining test functions. From Table 6, RRWS GA succeeds 62.5% (five out of eight succeeds) outperforming RWS.

Figures 1.11 and 1.12 show the 99% performance graph of average values of RWS and RRWS (by drawing mean ± 3 STD). The band created by RRWS is narrower than the band created by RWS, which implies that the fluctuation of RRWS is less than that of RWS. RRWS is more reliable than RWS, and this is applicable to all the remaining test functions graphs. In a general overview of all the graphs, RRWS graphs looked more stable than RWS

graphs. Next section ends up with the conclusions.

6. CONCLUSIONS

Roulette wheel selection is easy to implement and mimics nature more faithfully and therefore is much more appealing. But it is slower than the ranked based roulette wheel selection in convergence to near the optimum solution. If good solution is discovered early, its fitness value dominates other fitness values. Then it will occupy majority portions of the mating pool. This will reduce the diversity in the mating pool and cause the GAs to converge to wrong solutions. Ranked roulette wheel selection overcomes this problem and increases the diversity. By using RRWS the GA becomes steadier and faster towards the optimum solutions than RWS.

RRWS is outperforming the conventional RWS in convergence, time, reliability, certainty, and more robustness. GA can help find very good solutions to difficult real-world problems.

REFERENCES

- [1] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [2] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, NY, 1989.
- [3] Aarts, E. and Korst, J. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley. Chichester 1989
- [4] Glover, F., Future Paths for integer programming and links to artificial intelligence, *Comp.and Operations Res.*, 5, 533-549, 1986.



APPENDIX A

Function name	Description
objfun1	De Jong's function 1
objfun1a	Axis parallel hyper-ellipsoid
objfun1b	Rotated hyper-ellipsoid
objfun2	Rosenbrock's valley (banana function)
objfun6	Rastrigin's function
objfun7	Schwefel's function
objfun8	Griewangk's function
objfun9	Sum of different power

Table 1

objfunc	First Generation Hit the Optimal Value	
	RWS	RRWS
1	>200	30
1a	>200	40
1b	>200	>200
2	>200	20
6	>200	>200
7	>200	>200
8	>200	90
9	>200	50

Table 2

objfunc	First Generation Hit the Optimal Value		Average of Resource Saving
	RWS	RRWS	
1	180	40	0.78
1a	>200	40	0.80
1b	>200	100	0.50
2	>200	60	0.70
6	>200	>200	0.00
7	>200	100	0.50
8	160	60	0.63
9	100	20	0.80

Table 3

Objfunc	First Generation Hit the Zero Deviation Value		Average of Resource Saving
	RWS	RRWS	
1	200	50	0.75
1a	>200	50	0.75
1b	>200	>200	0.00
2	>200	100	0.50
6	>200	>200	0.00
7	>200	>200	0.00
8	>200	60	0.70
9	110	20	0.82

Table 4

objfunc	First Generation Fit the Mean Graph		Percentage of outperforming
	RWS	RRWS	
1	180	40	0.8
1a	>200	40	0.8
1b	>200	>200	0
2	>200	80	0.6
6	>200	>200	0
7	>200	>200	0
8	160	60	0.625
9	100	20	0.8

Table 5

objfunc	First Generation Fit the Mean Graph		Percentage of outperforming
	RWS	RRWS	
1	200	60	0.70
1a	>200	50	0.75
1b	>200	>200	0.00
2	>200	80	0.60
6	>200	>200	0.00
7	>200	>200	0.00
8	180	60	0.56
9	100	20	0.80

Table 6

Obifun1

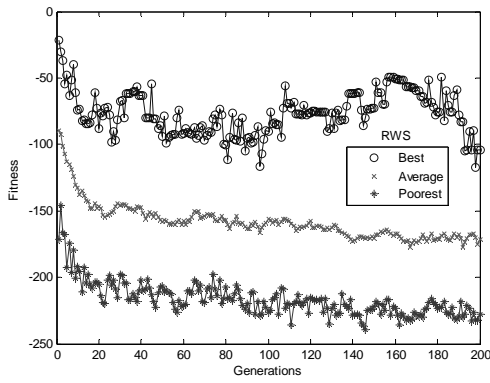


Figure 1.1

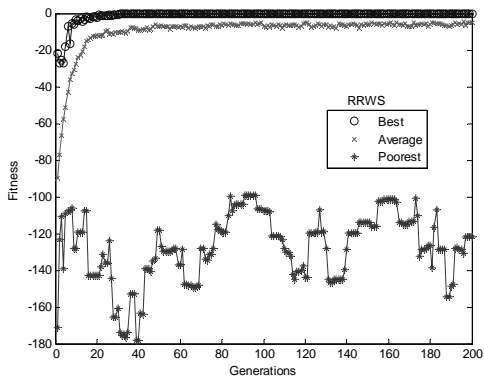


Figure 1.2

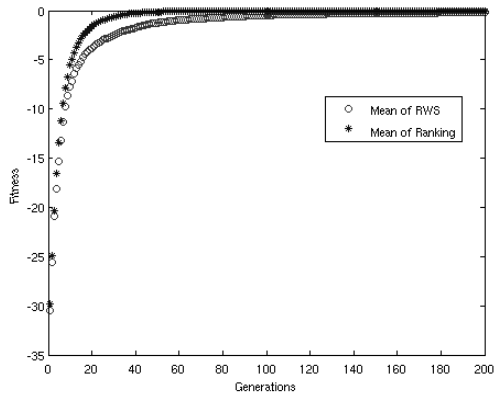


Figure 1.3

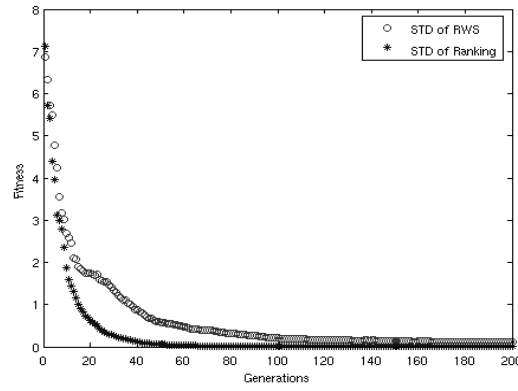


Figure 1.4

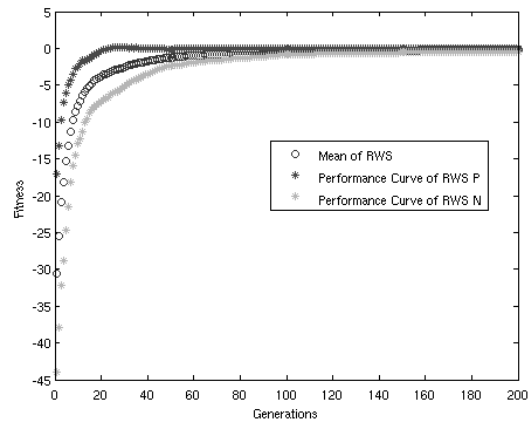


Figure 1.5

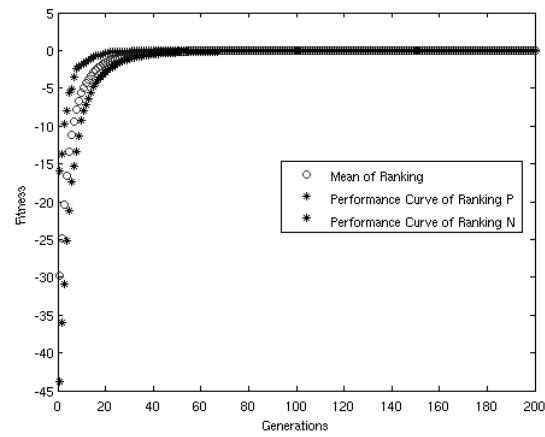


Figure 1.6

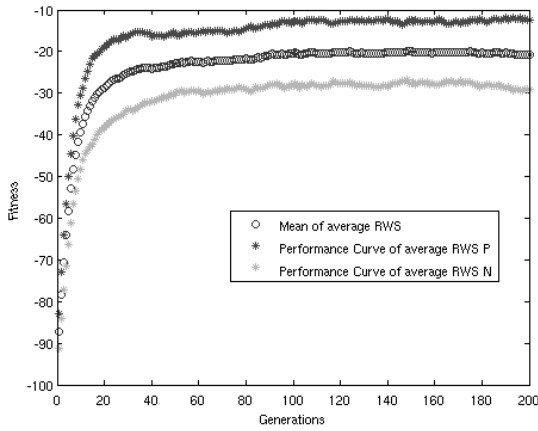


Figure 1.7

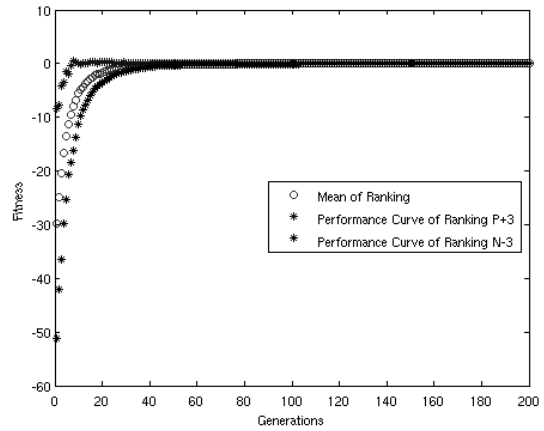


Figure 1.10

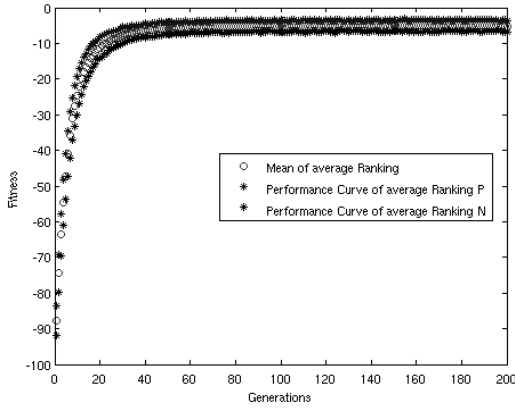


Figure 1.8

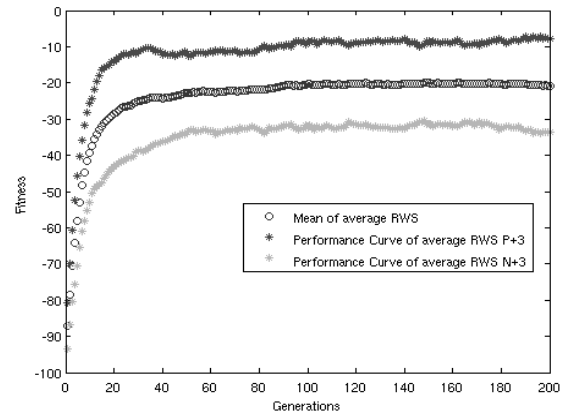


Figure 1.11

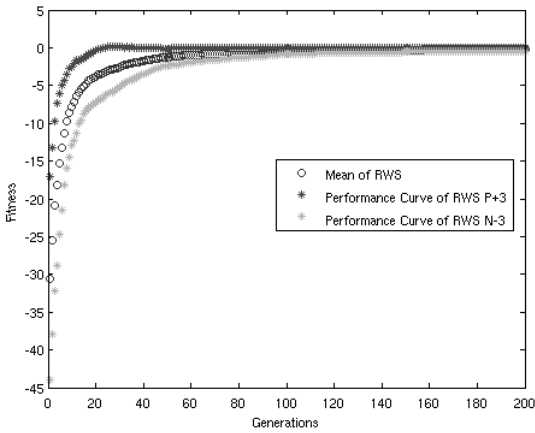


Figure 1.9

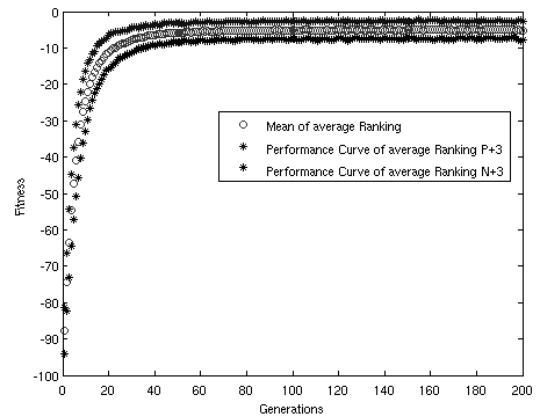


Figure 1.12

Objfun8

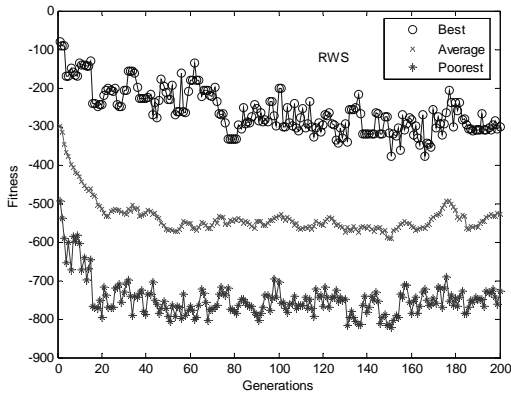


Figure 8.1

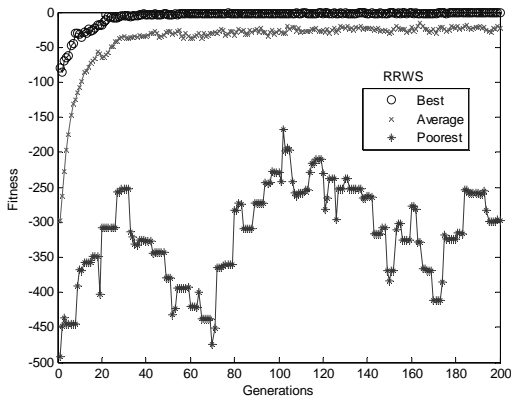


Figure 8.2

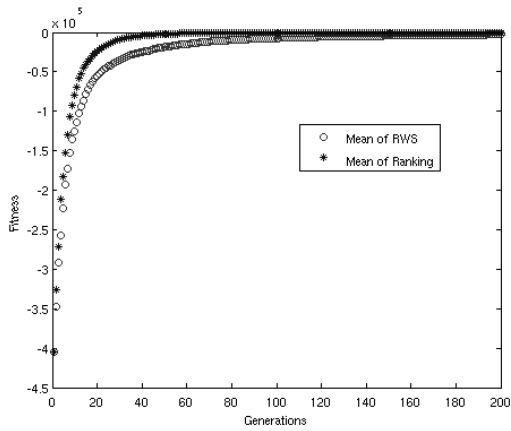


Figure 8.3

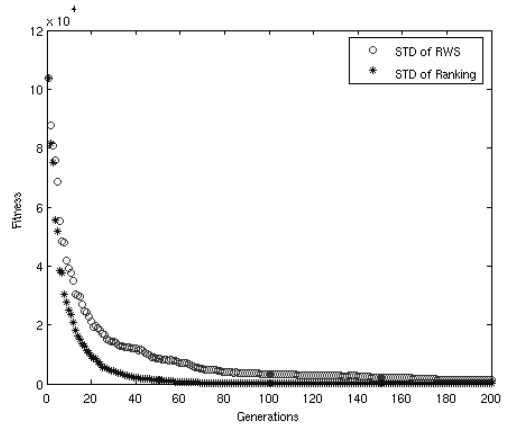


Figure 8.4

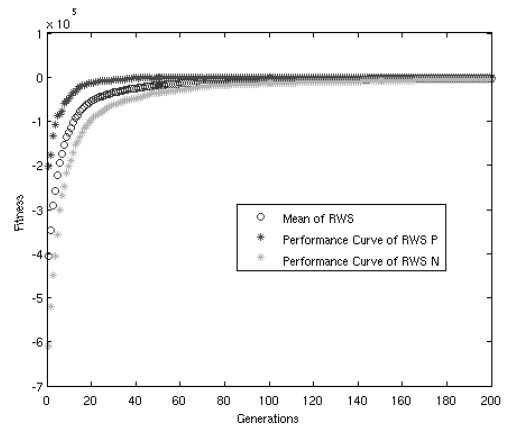


Figure 8.5

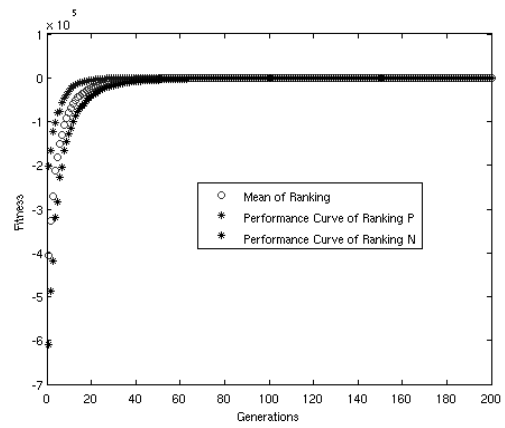


Figure 8.6

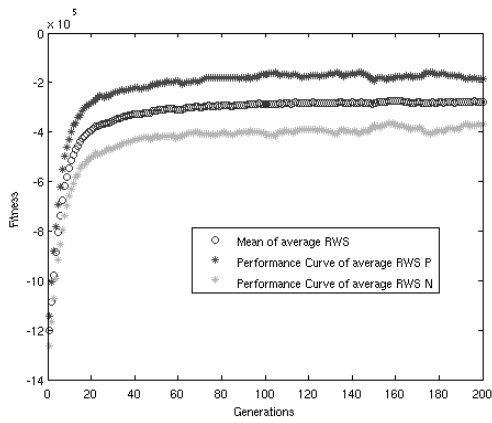


Figure 8.7

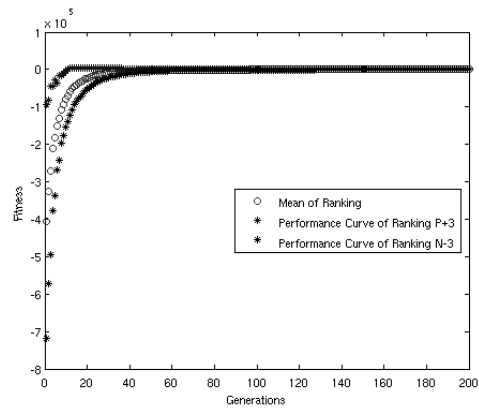


Figure 8.10

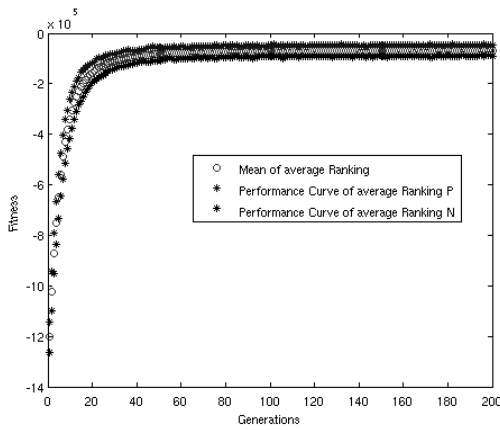


Figure 8.8

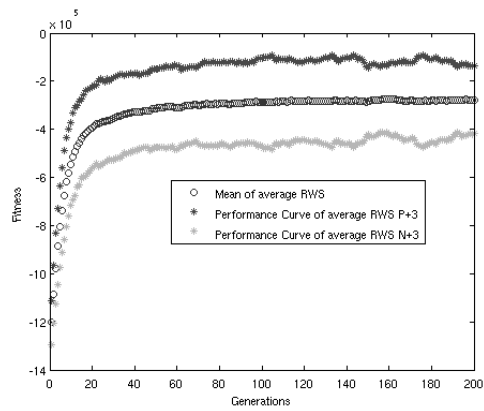


Figure 8.11

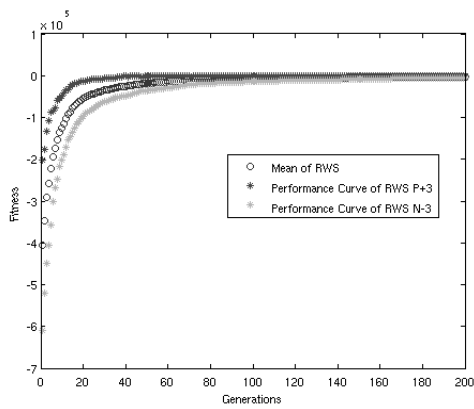


Figure 8.9

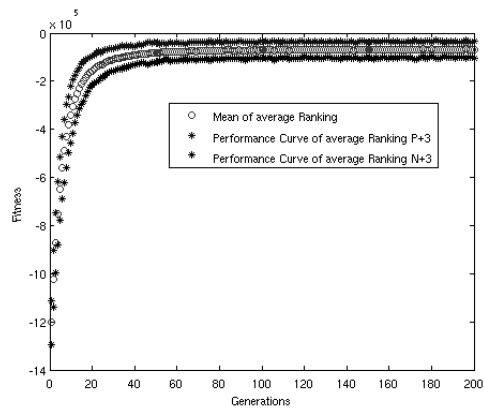


Figure 8.12