Chapter 4
Genetic Algorithm

4.1 Introduction

Genetic Algorithm (GA) is one of the first population-based stochastic algorithm proposed in the history. Similar to other EAs, the main operators of GA are selection, crossover, and mutation. This chapter briefly presents this algorithm and applies it to several case studies to observe its performance.

4.2 Inspiration

GA was inspired from the Darwinian theory of evolutionary [1, 2], in which the survival of fitter creature and their genes were simulated. GA is a population-based algorithm. Every solution corresponds to a chromosome and each parameter represents a gene. GA evaluates the fitness of each individual in the population using a fitness (objective) function. For improving poor solutions, the best solutions are chosen randomly with a selection (e.g. roulette wheel) mechanism. This operator is more likely to choose the best solutions since the probability is proportional to the fitness (objective value). What increases local optima avoidance is the probability of choosing poor solutions as well. This means that if good solutions be trapped in a local solution, they can be pulled out with other solutions.

The GA algorithm is stochastic, so one might ask how reliable it is. What makes this algorithm reliable and able to estimate the global optimum for a given problem is the process of maintaining the best solutions in each generation and using them to improve other solutions. As such, the entire population becomes better generation by generation. The crossover between individuals results in exploiting the 'area' between the given two parent solutions. This algorithm also benefits from mutation. This operator randomly changes the genes in the chromosomes, which maintains the

S. Mirjalili, *Evolutionary Algorithms and Neural Networks*, Studies
in Computational Intelligence 780, https://doi.org/10.1007/978-3-319-93025-1_4

diversity of the individuals in the population and increases the exploratory behavior of GA. Similar to the nature, the mutation operator might result is a substantially better solution and lead other solutions towards the global optimum.

4.3 Initial Population

The GA algorithm starts with a random population. This population can be generated from a Gaussian random distribution to increase the diversity. This population includes multiple solutions, which represent chromosomes of individuals. Each chromosome has a set of variables, which simulates the genes. The main objective in the initialisation step is to spread the solutions around the search space as uniformly as possible to increase the diversity of population and have a better chance of finding promising regions. The next sections discuss the steps to improve the chromosomes in the first population.

4.4 Selection

Natural selection is the main inspiration of this component for the GA algorithm. In nature, the fittest individuals have a higher chance of getting food and mating. This causes their genes to contribute more in the production of the next generation of the same species. Inspiring from this simple idea, the GA algorithm employs a roulette wheel to assign probabilities to individuals and select them for creating the next generation proportional to their fitness (objective) values. Figure 4.1 illustrate an example of a roulette wheel for six individuals. The details of these individuals are presented in Table 4.1.

It can be seen that the best individual (#5) has the largest share of the roulette wheel, while the worst individual (#4) has the lowest share. This mechanism simulates the natural selection of the fittest individual in nature. Since a roulette wheel is a stochastic operator, poor individuals have a small probability of participating in the creation of the next generation. If a poor solution is 'lucky', its genes move to the next generation. Discarding such solutions will reduce the diversity of the population and should be avoided.

It should be noted that the roulette wheel is one of the many selection operators in the literature [3–5]. Some of the other selection operators are:

- Boltzmann selection [6]
- Tournament selection [7]
- Rank selection [8]
- Steady state selection [9]
- Truncation selection [10]
- Local selection [11]

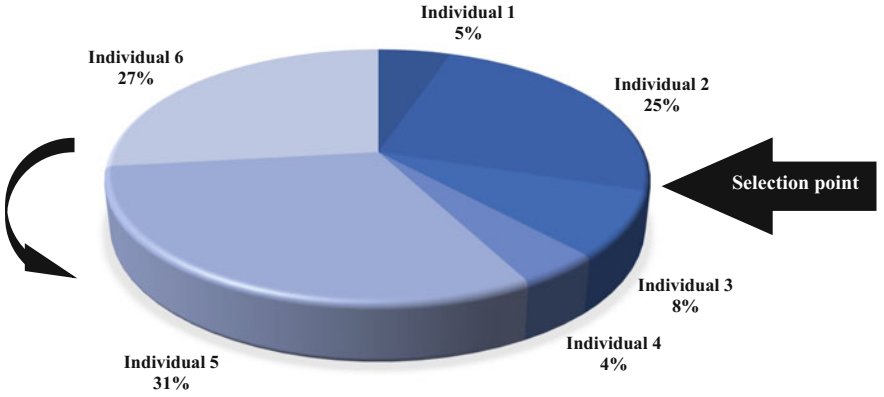

Fig. 4.1 Mechanism of the roulette wheel in GA. The best individual (#5) has the largest share of the roulette wheel, while the worst individual (#4) has the lowest share

Table 4.1 Details of the individuals in Fig. 4.1. The fittest individual is Individual #5

Individual number	Fitness value	% of Total
1	12	5
2	55	24
3	20	8
4	10	4
5	70	30
6	60	26
Total	227	100

- Fuzzy selection [12]
- Fitness uniform selection [13]
- Proportional selection [14]
- Linear rank selection [14]
- Steady-state reproduction [15]

4.5 Crossover (Recombination)

After selecting the individuals using a selection operator, they have to be employed to create the new generation. In nature, the chromosomes in the genes of a male and a female are combined to produce a new chromosome. This is simulated by combining two solutions (parent solutions) selected by the roulette wheel to produce two new solutions (children solutions) in the GA algorithm. There are different techniques for the crossover operator in the literature of which two (single-point and double-point [16]) are shown in Fig. 4.2.

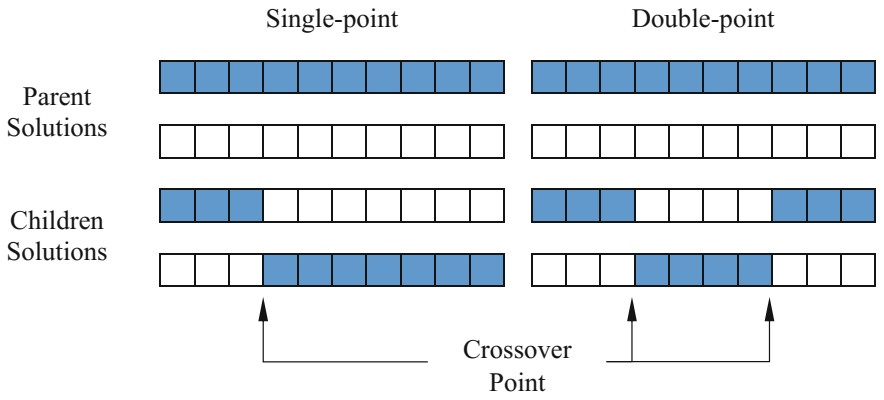

Fig. 4.2 Two popular crossover techniques in GA: single-point and double point. In the single-point cross over, the chromosomes of two parent solutions are swapped before and after a single point. In the double-point crossover, there are two cross over points and the chromosomes between the points are swapped only

In the single-point cross over, the chromosomes of two parent solutions are swapped before and after a single point. In the double-point crossover, however, there are two cross over points and the chromosomes between the points are swapped only. Other crossover techniques in the literature are:

- Uniform crossover [17]
- Half uniform crossover [18]
- Three parents crossover [19]
- Partially matched crossover [20]
- Cycle crossover [21]
- Order crossover [22]
- Position-based crossover [23]
- Heuristic cross over [24]
- Masked crossover [25]
- Multi-point crossover [26]

4.6 Mutation

The last evolutionary operator, in which one or multiple genes are altered after creating children solutions. The mutation rate is set to low in GA because high mutation rates convert GA to a primitive random search. The mutation operator maintains the diversity of population by introducing another level of randomness. In fact, this operator prevents solutions to become similar and increase the probability of avoiding local solutions in the GA algorithm. A conceptual example of this operator

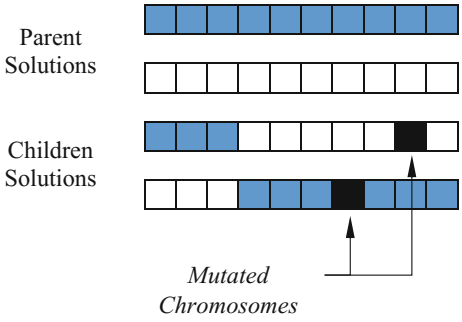

Fig. 4.3 Mutation operator alters one or multiple genes in the children solutions after the crossover phase

is visualised in Fig. 4.3. It can be seen in this figure that slight changes in the some of the randomly selected genes occur after the crossover (recombination) phase.

Some of the popular mutation techniques in the literature are:

- Power mutation [27]
- Uniform [28]
- Non-uniform [29]
- Gaussian [30]
- Shrink [31]
- Supervised mutation [32]
- Uniqueness mutation [33]
- Varying probability of mutation [34]

Taken together, most of EAs use the three evolutionary operators: selection, crossover, and mutation. These operators are applied to each generation to improve the quality of genes in the next generation. Another popular evolutionary operator is elitism [35], in which one or multiple best solutions are maintained and transferred without modification to the next generation. The main objective is to prevent such solutions (elites) from being degraded when applying the crossover or mutation operators.

The GA algorithm starts with a random population of individuals. Until the end of the end criterion, this algorithm improves the population using the above-mentioned three operators. The best solution in the last population is returned as the best approximation of the global optimum for a given problem. The rate of selection, crossover, and mutation can be changed or set to fix numbers during the optimisation. The next sections investigate the impact of changing such rates on the performance of GA.

4.7 Experiments When Changing the Mutation Rate

In this section several experiments are conducted to observe the impact of crossover and mutation rates on the performance of the GA algorithm. In the first experiment,

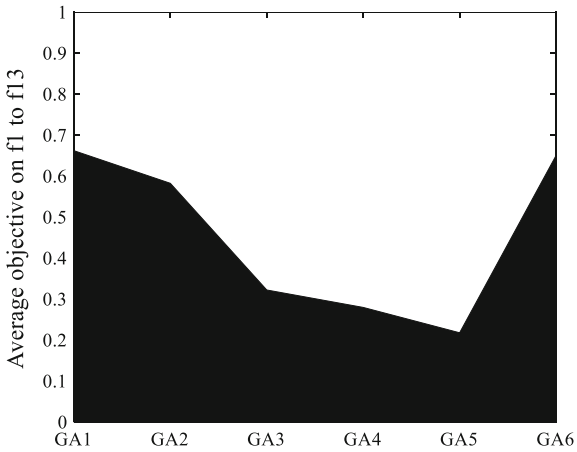

Fig. 4.4 Average objective value of 30 runs on all test functions for GA1 to GA6. The performance of the GA can be improved when the mutation is increased up to 0.4–0.6. However, this algorithm shows the worst average results when the mutation is very high. This is because the randomness of the search process increases proportional to the mutation rate

the mutation rate of GA is assigned with 0, 0.1, 0.2, 0.4, 0.6, 0.8 numbers to design GA1, GA2, GA3, GA4, GA5, and GA6 algorithms respectively. Note that the GA algorithm generates a random number in the interval of [0, 1], and if this number less than the mutation rate (if $r \leq \mu$ where r is the random number and μ is the mutation rate), the gene is mutated. This process is repeated for each chromosome and gene.

To compare the performance of GA when using different mutation rates, the function F1–F13 ([36]) is solved 30 independent runs and the statistical result are given in Table 4.2. To see how significantly the results of algorithms differ, Table 4.3 present the results of the Wilcoxon ranksum test conducted at 5% significance level.

To visually compare the overall performance of the GA algorithms, the average objective value of the solutions on all test functions are calculated and visualized in Fig. 4.4. It can be seen in the figure that the performance of the GA can be improved when the mutation is increased up to 0.4–0.6. However, this algorithm shows the worst average results when the mutation rate is very high. This is because the randomness of the search process increases proportional to the mutation rate. A lot of changes in the individuals, when the mutation rate is high, results in primitive random search and minimising the impact of selection and crossover operators. To observe the search pattern, Fig. 4.5 illustrates the search history (in the first two dimensions) of GA with 40 individuals and 5000 generations when solving the $F1$ test function.

This figure shows that the search history is sparse when the mutation is low. As a consequence, the accuracy and local search is high since parent solutions constantly share genes with a small level of mutations. By contrast, the coverage of landscape is high in the right subplot of Fig. 4.5. This shows that the solutions face changes a lot when the mutation rate is high that results in a more randomized search.

Table 4.2 There results of algorithms when changing the mutation rate. In this experiment the number of individuals is 40, and the maximum number of iterations is 500

Test function	GA1		GA2		GA3		GA4		GA5		GA6	
	mean	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*
F1	0.1366	0.4803	1.0000	1.0000	0.9636	0.1620	0.0000	0.6728	0.1394	0.0000	0.7018	0.8134
F2	1.0000	1.0000	0.1397	0.1203	0.4333	0.5596	0.0000	0.0000	0.0274	0.0226	0.5248	0.4507
F3	1.0000	1.0000	0.1300	0.0583	0.0000	0.1164	0.3640	0.0000	0.2171	0.0117	0.1336	0.4866
F4	1.0000	1.0000	0.6840	0.4524	0.2991	0.0913	0.0000	0.0000	0.4442	0.0405	0.4277	0.6498
F5	1.0000	1.0000	0.5890	0.9196	0.0000	0.3892	0.2202	0.5494	0.0853	0.0000	0.7966	0.3292
F6	1.0000	1.0000	0.1603	0.2800	0.0000	0.5967	0.3006	0.2959	0.4282	0.0000	0.4053	0.4448
F7	0.8532	0.6479	1.0000	0.7878	0.6110	0.0871	0.6421	0.1483	0.0000	0.0000	0.6544	1.0000
F8	1.0000	0.2423	0.7861	0.0000	0.1699	1.0000	0.4122	0.2979	0.0000	0.5469	0.3323	0.7173
F9	0.1404	0.7163	0.0000	1.0000	0.5839	0.6126	0.4022	0.2895	0.1289	0.4869	1.0000	0.0000
F10	1.0000	0.2586	0.2338	0.7589	0.0187	0.7901	0.0000	0.8643	0.2661	1.0000	0.9602	0.0000
F11	0.2344	0.4171	0.8521	1.0000	0.6126	0.4757	0.6601	0.3436	0.0000	0.2859	1.0000	0.0000
F12	0.2457	0.2652	1.0000	0.0000	0.0000	0.2413	0.0303	0.7846	0.8027	0.3148	0.7802	1.0000
F13	0.0000	0.4795	1.0000	0.3164	0.4995	1.0000	0.6068	0.1053	0.2952	0.5249	0.7392	0.0000

Table 4.3 P-values obtained after conducting the Wilcoxon ranksum text at 5% significance level for the algorithms in Table 4.2

Test function	GA1	GA2	GA3	GA4	GA5	GA6
F1	0.8501	0.2413	0.2413	N/A	0.7337	0.7337
F2	0.6232	0.6232	0.2413	N/A	0.4727	0.3447
F3	0.0452	0.5205	N/A	0.2730	0.5205	0.6232
F4	0.1041	0.3447	0.5708	N/A	0.6776	0.2123
F5	0.3447	0.4274	N/A	0.7913	0.8501	0.3447
F6	0.0257	0.5708	N/A	0.5708	0.2730	0.4274
F7	0.0058	0.0046	0.0113	0.0073	N/A	0.0376
F8	0.0173	0.0539	0.6232	0.2123	N/A	0.5205
F9	0.7913	N/A	0.9097	0.9097	0.6776	0.9698
F10	0.1405	0.9698	0.7913	N/A	0.9698	0.1212
F11	0.9698	0.4727	0.3847	0.4727	N/A	0.0757
F12	0.9698	0.0376	N/A	0.9097	0.1405	0.3847
F13	N/A	0.0312	0.2730	0.1041	0.2413	0.0757

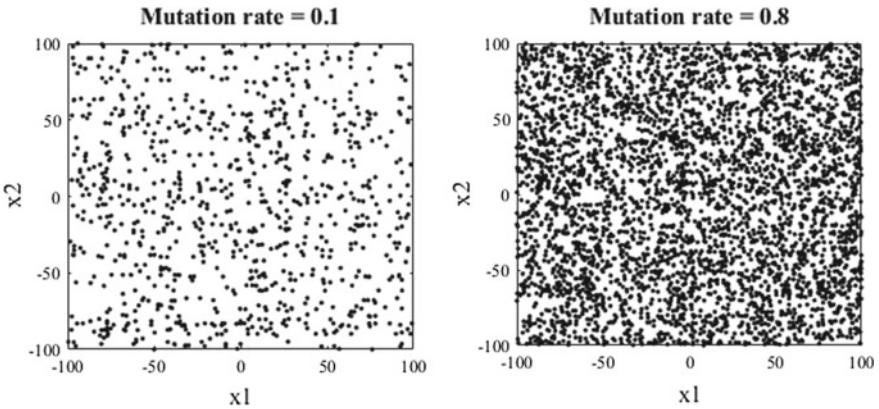

Fig. 4.5 Search history (in the first two dimensions) of GA with 40 individuals and 5000 generations when solving the F1 test function. The search history is sparse when the mutation is low. As the consequence, the accuracy and local search is high since parent solutions constantly share genes with a small level of mutations. By contrast, the coverage of landscape is high in the right subplot. This shows that the solutions face changes a lot when the mutation rate is high that results in a more randomized search

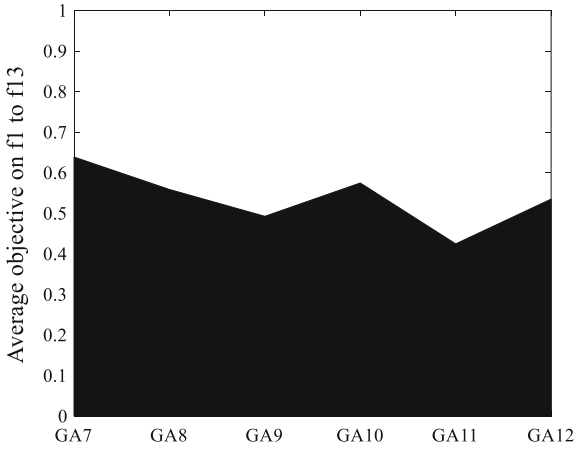

Fig. 4.6 Average objective value of 30 runs on all test functions for GA7 to GA12. The performance of GA does not degrade significantly when changing the crossover rate as opposed to the results in Fig. 4.4 when changing the mutation rate. The performance of GA is at best when the crossover rate is 0.9 (GA11)

4.8 Experiment When Changing the Crossover Rate

In the preceding section, the impact of the mutation rate on the performance of GA was investigated. In this section, the same experiments are conducted when changing the crossover rate: GA7 (0.5), GA8 (0.6), GA9 (0.7), GA10 (0.8), GA11 (0.9), GA12 (1). The average and standard deviation of 30 independent runs on F1–F13 are presented in Table 4.4. The p-values and average performance of GA on all test functions are shown in Table 4.5 and Fig. 4.6.

The results show that the performance of GA does not degrade significantly when changing the crossover rate as opposed to the results in the preceding section. Figure 4.6 shows that the performance of GA is at best when the crossover rate is 0.9 (GA11). High crossover rates allow creating more diverse children. As the main search mechanism, the crossover operator combines the best solutions to produce children. The higher and more frequently crossover, the higher probability of finding better children.

4.9 Conclusion

This section presented GA as one of the most popular evolutionary algorithm in the literature. After discussing the main evolutionary operators, several experiments were conducted to see the impact of changing controlling parameters on the performance of this algorithm. It was observed that the mutation rate should be tuned carefully since it can degrade the performance significantly when its value is high. It was also observed that the crossover rate is important in getting better results, but it can be set

Table 4.4 Experimental results when changing crossover rate: GA7 (0.5), GA8 (0.6), GA9 (0.7), GA10 (0.8), GA11 (0.9), GA12 (1). In this experiment the mutation was set to 0.6, the number of individuals are 40, and the maximum number of iterations are 500

Test function	GA7		GA8		GA9		GA10		GA11		GA12	
	mean	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*
F1	0.7508	0.7170	0.5297	1.0000	0.6734	0.4206	0.0000	0.0000	0.4046	0.5895	1.0000	0.0906
F2	1.0000	0.5773	0.4589	0.3578	0.5933	1.0000	0.0876	0.1178	0.0000	0.0000	0.1931	0.2154
F3	0.3315	0.0465	0.0569	1.0000	0.8908	0.4362	0.2770	0.6940	1.0000	0.3331	0.0000	0.0000
F4	0.0587	0.2214	0.9924	0.0000	0.5088	0.7690	0.9855	1.0000	0.0000	0.4858	1.0000	0.8390
F5	1.0000	0.4663	0.8640	0.0000	0.5640	0.1046	0.8677	0.6451	0.2526	1.0000	0.0000	0.7846
F6	0.8370	0.0601	0.8832	0.2771	0.3101	0.1238	1.0000	0.0000	0.3174	0.3928	0.0000	1.0000
F7	1.0000	0.0000	0.8933	0.3547	0.5416	0.3955	0.7573	0.2801	0.0000	0.5880	0.4453	1.0000
F8	0.6912	0.4536	0.0000	1.0000	0.7570	0.0000	0.5009	0.2983	0.3756	0.4908	1.0000	0.1683
F9	0.0945	1.0000	0.5832	0.7715	0.2822	0.0000	0.0000	0.8047	0.8402	0.8329	1.0000	0.5416
F10	0.4755	0.5376	0.8069	0.3889	0.9182	0.6203	1.0000	0.0000	0.0000	1.0000	0.6171	0.1449
F11	0.7502	0.3310	0.5221	0.2832	0.0665	0.0341	1.0000	0.0000	0.4164	1.0000	0.0000	0.7802
F12	0.4189	0.2010	0.0000	0.8621	0.2993	0.0000	0.5716	0.3662	0.9151	1.0000	1.0000	0.4007
F13	0.8967	0.0000	0.6746	0.0513	0.0000	0.7526	0.4259	0.3592	1.0000	1.0000	0.7011	0.8903

Table 4.5 P-values obtained after conducting the Wilcoxon ranksum text at 5% significance level for the algorithms in Table 4.4

Test function	GA7	GA8	GA9	GA10	GA11	GA12
F1	0.0257	0.3447	0.064	N/A	0.3075	0.0058
F2	0.0452	0.273	0.9698	0.9097	N/A	0.9097
F3	0.4727	0.6776	0.064	0.5708	0.0173	N/A
F4	0.7913	0.089	0.8501	0.1859	N/A	0.2123
F5	0.064	0.3447	0.4274	0.162	0.8501	N/A
F6	0.1041	0.1405	0.9097	0.0312	0.7337	N/A
F7	0.1041	0.273	0.4727	0.4727	N/A	0.7337
F8	0.1212	N/A	0.5205	0.7913	0.9698	0.1041
F9	0.9698	0.3447	0.7337	N/A	0.273	0.1212
F10	0.9698	0.5708	0.4274	0.7337	N/A	N/A
F11	0.3847	0.5205	0.9097	0.273	0.6232	N/A
F12	0.2123	N/A	0.2413	0.273	0.1859	0.0539
F13	0.064	0.1859	N/A	0.4274	0.089	0.2123

to high values without negative consequences as opposed to the mutation rate. For training NNs, a continuous version of GA should be employed as discussed in the preceding chapter (ACO). Due to the difficulty of training NN, both crossover and mutation rates will tuned.

References

1. Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine Learning*, *3*(2), 95–99.
2. Holland, J. H. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–73.
3. Genlin, J. (2004). Survey on genetic algorithm. *Computer Applications and Software*, *2*, 69–73.
4. Cant-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, *10*(2), 141–171.
5. Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 69–93). Elsevier.
6. Goldberg, D. E. (1990). A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, *4*(4), 445–460.
7. Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, *9*(3), 193–212.
8. Kumar, R. (2012). Blending roulette wheel selection & rank selection in genetic algorithms. *International Journal of Machine Learning and Computing*, *2*(4), 365.
9. Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 94–101). Elsevier.
10. Blickle, T., & Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4*(4), 361–394.

11. Collins, R. J., & Jefferson, D. R. (1991). *Selection in massively parallel genetic algorithms* (pp. 249–256). University of California (Los Angeles), Computer Science Department.
12. Ishibuchi, H., & Yamamoto, T. (2004). Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining. *Fuzzy Sets and Systems*, *141*(1), 59–88.
13. Hutter, M. (2002). Fitness uniform selection to preserve genetic diversity. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC'02* (Vol. 1, pp. 783–788). IEEE.
14. Grefenstette, J. J. (1989). How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the 3rd International Joint Conference on Genetic Algorithms (ICGA89)*.
15. Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9). Morgan Kaufmann Publishers.
16. Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *Computer*, *27*(6), 17–26.
17. Semenkin, E., & Semenkina, M. (2012). Self-configuring genetic algorithm with modified uniform crossover operator. In *International Conference in Swarm Intelligence* (pp. 414–421). Heidelberg: Springer.
18. Hu, X. B., & Di Paolo, E. (2007). An efficient genetic algorithm with uniform crossover for the multi-objective airport gate assignment problem. In *IEEE Congress on Evolutionary Computation, 2007 (CEC 2007)* (pp. 55–62). IEEE.
19. Tsutsui, S., Yamamura, M., & Higuchi, T. (1999). Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1* (pp. 657–664). Morgan Kaufmann Publishers Inc.
20. Bck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (2000). *Evolutionary computation 1: Basic algorithms and operators* (Vol. 1). CRC press.
21. Oliver, I. M., Smith, D., & Holland, J. R. (1987). Study of permutation crossover operators on the travelling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, July 28–31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates.
22. Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *IJCAI* (Vol. 85, pp. 162–164).
23. Whitley, D., Timothy, S., & Daniel, S. Schedule optimization using genetic algorithms. In D. Lawrence (Ed.) 351–357.
24. Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications* (pp. 160–168).
25. Louis, S. J., & Rawlins, G. J. (1991). Designer genetic algorithms: Genetic algorithms in structure design. In *ICGA* (pp. 53–60).
26. Eshelman, L. J., Caruana, R. A., & Schaffer, J. D. (1989). Biases in the crossover landscape. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 10–19). Morgan Kaufmann Publishers Inc.
27. Deep, K., & Thakur, M. (2007). A new mutation operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *193*(1), 211–230.
28. Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(4), 656–667.
29. Neubauer, A. (1997). A theoretical analysis of the non-uniform mutation operator for the modified genetic algorithm. In *IEEE International Conference on Evolutionary Computation* (pp. 93–96). IEEE.
30. Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. In *IEEE International Conference on Evolutionary Computation* (Vol. 1, p. 384). IEEE.
31. Tsutsui, S., & Fujimoto, Y. (1993). Forking genetic algorithm with blocking and shrinking modes (fGA). In *ICGA* (pp. 206–215).
32. Oosthuizen, G. D. (1987). Supergran: A connectionist approach to learning, integrating genetic algorithms and graph induction. In *Proceedings of the second International Conference on Genetic Algorithms and their Applications*, July 28–31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates.

33. Mauldin, M. L. (1984). Maintaining diversity in genetic search. In *AAAI* (pp. 247–250).
34. Ankenbrandt, C. A. (1991). An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 53–68). Elsevier.
35. Ahn, C. W., & Ramakrishna, R. S. (2003). Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, *7*(4), 367–385.
36. Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, *114*, 163–191.