Performance Analysis of Different Operators in Genetic Algorithm for Solving Continuous and Discrete Optimization Problems

Shilun Song[1], Hu Jin[1] and Qiang Yang[2,*]

[1]*Department of Electrical and Electronic Engineering, Hanyang University, Ansan, 15588, South Korea*

[2]*School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing 210044, China*

Keywords: Genetic Algorithm, Nonlinear Optimization, Traveling Salesman Problem.

Abstract: Genetic algorithm (GA), as a powerful meta-heuristics algorithm, has broad applicability to different optimization problems. Although there are many researches about GA, few works have been done to synthetically summarize the impact of different genetic operators and different parameter settings on GA. To fill this gap, this paper has conducted extensive experiments on GA to investigate the influence of different operators and parameter settings in solving both continuous and discrete optimizations. Experiments on 16 nonlinear optimization (NLO) problems and 9 traveling salesman problems (TSP) show that tournament selection, uniform crossover, and a novel combination-based mutation are the best choice for continuous problems, while roulette wheel selection, distance preserving crossover, and swapping mutation are the best choices for discrete problems. It is expected that this work provides valuable suggestions for users and new learners.

1 INTRODUCTION

Since Holland (Holland, 1992) came up with the first generation of GA in 1975, GA and its variants have been used in many fields, like traveling salesman problems(Wang et al., 2020), wireless sensor network(Zorlu et al., 2017), feature selection(Li et al., 2009)(Huang et al., 2010), etc. However, few research exists on discussing in detail the impact of different operators and parameters on the performance of GA.

As shown in figure 1, there are mainly three types of operators in GA: selection operator, crossover operator, and mutation operator. The selection operator mainly chooses promising parents for the crossover operator. The crossover operator aims to exchange the information of parent individuals to generate new offspring. The mutation operator is to generate new values of variables based on some rules. By repeatedly applying these three operators, the optima of problems may be found.

To comprehensively investigate the performance of different operators and parameters in GA, we conduct extensive experiments on both continuous and discrete problems. Specifically, on continuous problems, we use the nonlinear optimization (NLO) problems as the representatives to conduct experiments, while on discrete problems we use the traveling salesman problems (TSP) as the representatives.

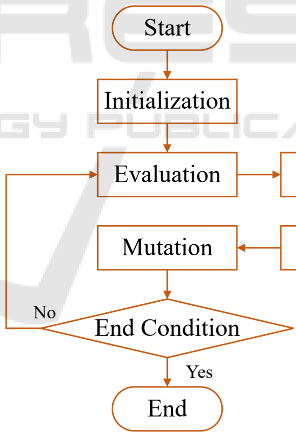


Figure 1: The overall framework of genetic algorithm (GA).

In the literature, there are some researches working on the comparison of selection operators (Zhong et al., 2005) (Chen et al., 2020), crossover operators (Pinho and Saraiva, 2020), and the combinations of crossover and mutation operators (Hildayanti et al., 2018) on NLO. For TSP problems, researchers have also developed many variants of GA from different perspectives, like improvement for selection operators (Yu et al., 2016), crossover operators (Freisleben

*Corresponding author.

Song, S., Jin, H. and Yang, Q.
Performance Analysis of Different Operators in Genetic Algorithm for Solving Continuous and Discrete Optimization Problems.
DOI: 10.5220/0010494005360547
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 536-547
ISBN: 978-989-758-509-8

and Merz, 1996), mutation operators (Zhou and Song, 2016), and the combination of the above two operators (Yu et al., 2011).

Different from existing studies, this paper aims to give a full-scale introduction and analysis of different operators and parameter settings on GA.

The remainder of this paper is organized as follows. In section 2, we discuss the continuous and discrete optimization problems. Then, in section 3, we show the involved genetic operators for the two types of problems in detail. In section 4, numerical experiments are conducted to show the performance of different operators. Finally, we give the conclusion in section 5.

2 CONTINUOUS AND DISCRETE OPTIMIZATION PROBLEMS

2.1 Continuous Optimization Problem

In the real world, NLO widely exists in many fields in different forms, like autonomous surface vehicles (Eriksen and Breivik, 2017), optimal placement problems (Almunif and Fan, 2017), and signal source localization (Ma et al., 2017), etc. This kind of problems become considerably difficult to solve as the dimensionality increases (Yang et al., 2020). Except for GA, there are also researches which apply other evolutionary algorithms to solve this type of problems, just like ant colony optimization (Yang et al., 2017b) and estimation of distribution algorithms (Yang et al., 2017a).

The target of NLO is expected to achieve the optimal solution under some restrictions, and the feasible solutions are taken from the continuous set. NLO can be noted as

$$\begin{aligned} & maximize/minimize\ f(\boldsymbol{x}), \\ & \qquad subject\ to\ \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

where $f(\boldsymbol{x})$ is a nonlinear objective function, and $\boldsymbol{X}$ is the domain of the target function . For the convenience of discussion, we note the feasible solution $\boldsymbol{x}$ as an M-dimensional vector $(x_1, x_2, \ldots, x_M)$.

Based on the property that whether there are multiple optima in the solution space, NLO is roughly categorized into two types: unimodal problems and multimodal problems.

- **Unimodal Problem.** There is only one optimum, namely the global optimum, for this kind of problems, as shown in figure 2. Such a property leads to that this type of problems is relatively easy to solve.

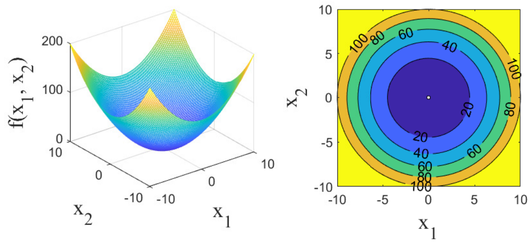


Figure 2: An example of unimodal functions.

- **Multimodal Problem.** As shown in figure 3, there are more than one local optima around the global one. This type of problems is more common and complex in the real world applications. Thus, we pay more attention to this type of functions.

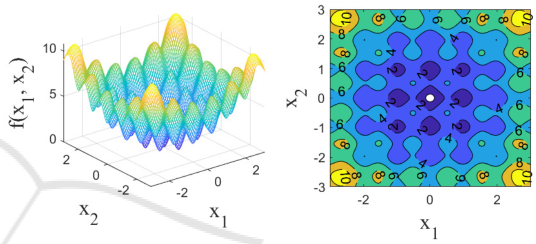


Figure 3: An example of multimodal functions.

2.2 Discrete Optimization Problem

TSP is a typical and complex kind of discrete problems. Many optimization problems in real-world applications could be modeled as TSP, like multi-bridge machining schedule (Li et al., 2017), 3-D printing (Ganganath et al., 2016) and unmanned aircraft systems (Xie et al., 2019).

Let $V = \{v_1, v_2, \ldots, v_M\}$ be the city set, where v_i represents the city indexed by i. The cost from city i to j is noted as c_{ij}. For simplicity, the problems in the following discussion are limited to symmetric TSP, which means that $c_{ij} = c_{ji}$ for $\forall i, j$.

Likewise, we note a route as $\boldsymbol{x} = (x_1, x_2, \ldots, x_M)$, whose total distance is noted as

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} c_{x_i x_{i+1}}$$

where $x_{M+1} = x_1$. The purpose is to search for the optimal route which brings the minimal cost. Figure 4 shows the optimal route for the *a280* instance. Our example data comes from TSPLIB (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html).

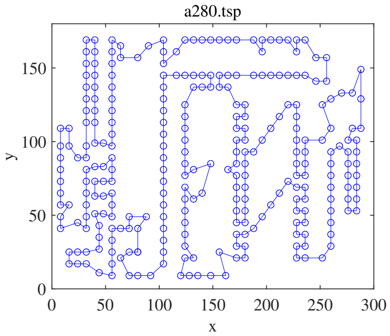


Figure 4: The optimal route of *a280*.

3 GENETIC OPERATORS

In this part, related evolutionary operators are discussed. Suppose there are N individuals that get involved in the evolution. The whole population is noted as $U = \{u_1, u_2, \ldots, u_N\}$. The fitness value of individual u_i is noted as α_i. All fitness values are listed in the fitness value vector $\Theta = (\alpha_1, \alpha_2, \ldots, \alpha_N)$.

3.1 Encoding Method

3.1.1 NLO Encoding

For NLO, each dimension of the chromosome is coded with a real number within the feasible range. As a consequence, each chromosome represents a feasible solution in the solution space.

3.1.2 TSP Encoding

The chromosome for TSP is set as a permutation of the cities. For example, a feasible solution of 9-city TSP is shown in figure 5.

9	4	7	5	3	6	1	8	2

Figure 5: A chromosome of GA when solving a TSP instance with 9 cities.

3.2 Initialization Operator

3.2.1 NLO Initialization

As for NLO, the genes are randomly sampled in the corresponding value ranges.

3.2.2 TSP Initialization

For TSP, the greedy method is used to initialize the routes as shown in algorithm 1.

Algorithm 1: TSP Initialization.

Input: the number of cities M.
Output: a new route $r_1 r_2 \ldots r_M$.
Initialization: an empty route
Randomly choose a city v'_1
$r_1 \leftarrow v'_1$
for $i = 2$ to M **do**
  Choose the city v'_i that is the nearest to v'_{i-1} and not included in the route yet.
  $r_i \leftarrow v'_i$
end for

3.3 Selection Operator

Considering that the selection operators can be commonly used in both continuous and discrete problems, there is no need to discuss them respectively. In this paper, the following two selection operators are concerned.

3.3.1 Roulette Wheel Selection (RWS)

This operator comes from the roulette game. In this selection, the individuals with larger fitness values have higher probabilities to be selected. The detailed process of the roulette wheel selection is shown in algorithm 2.

Algorithm 2: Roulette Wheel Selection (RWS).

Input: population $U = \{u_1, u_2, \ldots, u_N\}$, fitness value vector $\Theta = \{\alpha_1, \alpha_2, \ldots, \alpha_N\}$.
Output: selected parents U'.
1: **for** $i = 2$ to N **do**
2:   $\alpha_i \leftarrow \alpha_i + \alpha_{i-1}$
3: **end for**
4: **for** $i = 1$ to N **do**
5:   $\alpha_i \leftarrow \alpha_i / \alpha_N$
6: **end for**
7: $\alpha_0 \leftarrow 0$
8: $r \leftarrow rand(0,1)$. // Uniform$[0,1]$
9: Find k s.t. $r \in (\alpha_{k-1}, \alpha_k]$
10: Record individual u_k as a selected parent.
11: Go back to 8 until N parents are selected.

3.3.2 Tournament Selection (TS)

The main process of the tournament selection is shown in algorithm 3. Obviously, a larger n_{player} brings higher selection pressure to the population, where individuals with poorer performance have smaller probability to survive.

Algorithm 3: Tournament Selection (TS).

Input: population $U = \{u_1, u_2, \dots, u_N\}$, fitness value vector $\Theta = \{\alpha_1, \alpha_2, \dots, \alpha_N\}$.
Output: selected parents U'.

```
for i = 1 to N do
    Randomly select n_player individuals.
    Record the best individual as a selected parent.
end for
```

3.4 Crossover Operator

The crossover operators are used for parents to generate their children from the microcosmic perspective. The combination of genes from parents brings children different performance. We note p_x as the probability that an individual is selected. For every couple of selected parent individuals, a corresponding crossover operator is applied.

3.4.1 NLO Crossover

The crossover operators for NLO generate child individuals by letting the two chosen parents swap some corresponding genes.

Single Point Crossover (SPoC). For SPoC, a position of the chromosome except for the first one is randomly chosen, and all of the genes from that position to the end are swapped. The process in detail is shown in algorithm 4.

Algorithm 4: Single Point Crossover (SPoC).

Input: two parent chromosomes $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$.
Output: two child chromosomes.

```
Randomly choose a position k ∈ {2, ..., M}
for i = k to M do
    Swap x_i^[1] and x_i^[2]
end for
```

Uniform Crossover (UC). For UC, the algorithm checks every position and swaps the corresponding genes with probability p_{swap}. The process in detail is shown in algorithm 5.

3.4.2 TSP Crossover

In TSP, the chromosome is a complete route, which means the operators should not break the routing rule of TSP. Two common operators called distance preserving crossover (Freisleben and Merz, 1996) and single piece crossover (Kaur and Murugappan, 2008) are discussed as follow.

Algorithm 5: Uniform Crossover (UC).

Input: two parent chromosomes $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$.
Output: two child chromosomes.

```
for i = 1 to M do
    if rand() < p_swap then
        Swap x_i^[1] and x_i^[2]
    end if
end for
```

Distance Preserving Crossover (DPC). DPC generates a set O where every element o_i satisfies the following two conditions:

- o_i is a sub-route of both parent routes.
- o_i is not a sub-route of any common route piece in both parent routes.

Then it completes the child routes via the greedy strategy. The process of DPX is shown in figure 6

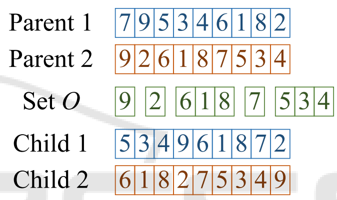


Figure 6: Distance preserving crossover.

Single Piece Crossover (SPiC). SPiC firstly selects a corresponding continuous piece of the chromosomes of the chosen parents, and then swaps the corresponding piece, just as shown in figure 7.

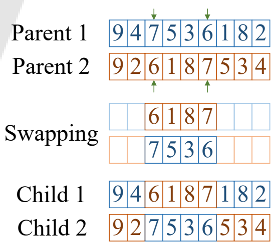


Figure 7: Single piece crossover.

3.5 Mutation Operator

3.5.1 NLO Mutation

In the mutation process, the algorithm checks all genes in all chromosomes and applies the mutation operator with probability p_m.

Uniform Mutation (UM). The chosen gene randomly switches to another value within the corresponding value range. This method is helpful for the algorithm to jump out of the local optimal areas, but might kill the good feasible solution.

Exponential Mutation (EM). It brings a mutation according to the fitness value of the individual. The main idea is that the individual with a higher fitness value should get a smaller mutation to keep the good property and the poor individuals should get a bigger mutation.

Let D_i be the value range of the chosen gene x_i, and $\hbar$ is a random value generator that generates 1 with probability 0.5 and generates -1 with probability 0.5. Besides we note $\mho(\lambda)$ as a random value generator whose value follows the exponential distribution with parameter λ. Obviously, with a higher λ, the $\mho_\lambda$ is closer to 0 expectedly.

Algorithm 6 shows the process of the exponential mutation operator for gene x_i, where factor a and b are parameters to control the mutation level.

Algorithm 6: Exponential Mutation (EM).

Input: gene x_i, fitness value vector Θ.
Output: mutated gene x_i'.

$\hat{\alpha} \leftarrow \max(\Theta)$
$\lambda \leftarrow b(\alpha_i/\hat{\alpha})^a$
$\delta \leftarrow \hbar\mho(\lambda)$ s.t. $x_i + \delta \in D_i$
$x_i' \leftarrow x_i + \delta \in D_i$

Combination Mutation (CM). Basically, the exponential mutation is a greedy strategy biased to good individuals. However, the eliminated individuals with poor performance may be also useful for the computation. Thus, we design a combination mutation operator that applies the uniform mutation and the exponential mutation with the same probability. This is supposed to decrease the influence of the greedy strategy.

3.5.2 TSP Mutation

The mutation for TSP should not break the completeness of a route, and operators bringing overturn are not supposed to be applied. In this paper, the swap mutation (SM) and the simple inversion mutation (SIM) are discussed.

Swap Mutation (SM). In this method, a probability p_m is used to select the mutation gene. The operator swaps the cities on the chosen genes and their corresponding nearest cities.

Simple Inversion Mutation (SIM). For every individual, once two positions on one of the route to be mutated are chosen, the operator inverts the route piece between the two positions, as shown in figure 8.

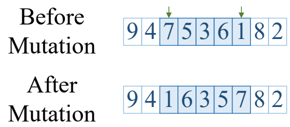


Figure 8: Simple inversion mutation.

3.6 Accelerator: Elitist Strategy

This operator is set to get involved in every calculation as default. As is known, due to the randomness, the solution structure of every individual can be changed with non-negligible probability, which means that even the optimal individual may not be kept and the searching speed of the algorithm gets slow.

The elitist selection acts as follows: in every generation, the best individual is selected and gets comparison with the best one in the last generation. If the current best individual performs better, it takes place of the best one in the last generation. If not, the best individual in the last generation takes place of the worst individual in this generation. Thus, the best individual until the current generation can always be kept in the population.

Actually, it is checked that the number of kept elites has no significant influence on the performance, which means keeping the best one in every generation is enough.

3.7 Local Search for TSP

As for TSP, local search is a necessary operator to improve the solution accuracy. In this paper, we just use 2-opt as the local search operator. It is also an accelerator for the computation of TSP. The 2-opt operator for an individual chromosome x_k is shown in algorithm 7.

3.8 The Complete GAs

Finally, we give the detailed process of the two complete GAs: NLO-GA and TSP-GA.

Algorithm 8 shows the detailed process of GA for NLO, and algorithm 9 shows the detailed process of GA for TSP. GEN_{MAX}, as the end controller, is the number of generations for evolution. Step 3 of both algorithms is used to record the individual with the highest fitness value. It is the preparation for the elitist selection. The overall difference between the two

Algorithm 7: 2-opt.

Input: chromosome $\boldsymbol{x}_k$, repeating time K.
Output: improved chromosome $\boldsymbol{x}_k$.

```
Compute the total distance f(x_k)
x'_k ← x_k
for i = 1 to N do
    Randomly select two cities v_i and v_j
    Swap v_i and v_j on x'_k
    if f(x'_k) < f(x_k) then
        x_k ← x'_k
    end if
end for
```

GAs is the local search for TSP. In the next section, we use these two schemes to test the performance of genetic operators.

Algorithm 8: NLO-GA.

Input: population size N, maximum generation GEN_{MAX}, n_{player} (if necessary), p_x, p_{swap} (if necessary), p_m.
Output: best individual u_{opt}.

```
Initialize the population
Evaluate
Keep the best
for generation = 1 to GEN_MAX do
    Select
    Crossover
    Mutate
    Evaluate
    Keep the best and do the elitist strategy
end for
```

4 NUMERICAL EXPERIMENTS

In this section, we conduct numerical experiments to compare genetic operators. Actually, it is hard to do the theoretical analysis on which is the best operator. Our purpose is to find the suitable operators that can obtain the best result under identical computation costs.

Firstly, we introduce the details about all of the experiments in subsection 4.1. With the numerical results, the comparisons of selection strategies, crossover strategies, and mutation strategies are given in subsection 4.2, 4.3, and 4.4, respectively.

4.1 Experiment Details

All algorithms are implemented in C language and run on PC with Windows 10, CPU Intel Core i7-8086K and RAM 64 GB.

Algorithm 9: TSP-GA.

Input: population size N, maximum generation GEN_{MAX}, n_{player} (if necessary), p_x, p_m (if necessary).
Output: best individual u_{opt}.

```
Initialize the population
Evaluate
Keep the best
for generation = 1 to GEN_MAX do
    Select
    Crossover
    Mutate
    Evaluate
    Keep the best and do the elitist strategy
    Local search
end for
```

4.1.1 Settings for NLO

For NLO, we select 4 functions shown in table 1 whose dimension can be easily expanded to high dimension.

Besides, for NLO-GA, parameter settings are listed in table 2. Due to the randomness of the algorithm, we use the average results over hundreds of independent runs. The *sample number* in the table means the repeating time.

4.1.2 Settings for TSP

For TSP, we select 9 instances of different sizes from TSPLIB, which are listed in table 3. Besides, for TSP-GA, some important and fixed parameter settings are listed in table 4.

4.2 Selection Strategies Comparision

4.2.1 Selection Strategies in NLO

In this part, we mainly compare the performance of RWS and TS on NLO. In the comparison, the SPoC and UM are used as default crossover and mutation operators. Under this setting, figures 18, 19, and 20 shows the performance of NLO-GA on 5, 15, and 30 dimensional problems.

4.2.2 Selection Strategies in TSP

In this part, we mainly compare the performance of RWS and TS on TSP. In comparison, the DPC and SM are used as default crossover and mutation operators. Under this setting, figures 9, 10, and 11 shows the performance of TSP-GA on 50, 100, and 200 size groups.

Table 1: NLO Instances.

Function	Type	Range	Optima
$f_1 = 1 + \sum_{i=1}^{D} x_i^2$	Unimodal	$[-100, 100]^D$	1
$f_2 = 1 + 20 + e - 20\exp\left\{-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}}\right\} - \exp\left\{\frac{\sum_{i=1}^{D}\cos(s\pi x_i)}{D}\right\}$	Multimodal	$[-32, 32]^D$	1
$f_3 = 1 + D - \sum_{i=1}^{D}\left\{\cos(2\pi x_i) - 0.01 * x_i^2\right\}$	Multimodal	$[-5.12, 5.12]^D$	1
$f_4 = 1 + \sum_{i=1}^{D}\left\{y_i^2 - 10\cos(2\pi y_i) + 10\right\}$, where $y_i = x_i$ if $\lvert x_i\rvert < 0.5$ and $y_i = \frac{round(2x_i)}{2}$ if $\lvert x_i\rvert \geq 0.5$.	Multimodal	$[-5.12, 5.12]^D$	1

Table 2: Parameter settings of NLO-GA.

Patameter	Value
Population size	40
Tournament player	10
p_{swap} for UC	0.5
Sample number	150

Table 3: TSP Instances.

Size Group	Instance	The Number of Cities
50	*att48*	48
	eil51	51
	berlin52	52
100	*kroC100*	100
	eil101	101
	lin105	105
200	*rat195*	195
	d198	198
	kroA200	200

4.2.3 Conclusion of Selection Strategies

According to the numerical results, we get the following conclusions for solving NLO and TSP:

- For NLO, TS performs better than RWS. The mutation probability p_m makes significiant influence. With high dimension, TS needs a low p_m to achieve the best performance.
- For TSP, RWS is better, and it needs a small crossover probability p_x to find a relatively short route.

4.3 Crossover Strategies Comparision

4.3.1 Crossover Strategies in NLO

In this part, we mainly compare the performance of SPoC and UC on NLO. Actually, it is checked that p_{swap} does not make obvious influence on the computation. In the comparison, the TS and UM are used as default selection and mutation operators. Under this setting, figures 21, 22, and 23 shows the performance of NLO-GA on 5, 15, and 30 dimensional problems.

Table 4: Parameter settings of TSP-GA.

Patameter	Value
Population size	60
Tournament player	15
p_m for SM	0.1
2-opt time	10
Sample number (group 50&100)	300
Sample number (group 200)	100

4.3.2 Crossover Strategies in TSP

In this part, we mainly compare the performance of DPC and SPiC on TSP. In comparison, the RWS and SM are used as default selection and mutation operators. Under this setting, figures 12, 13, and 14 shows the performance of TSP-GA on 50, 100, and 200 size groups.

4.3.3 Conclusion of Crossover Strategies

According to the numerical results, we get the following conclusions for solving NLO and TSP:

- For NLO, the difference with respect to the performance between the two crossover strategies is not obvious.
- For TSP, DPC works better, and it also needs a small crossover probability p_x to find a relatively short route.

4.4 Mutation Strategies Comparision

4.4.1 Mutation Strategies in NLO

In this part, we mainly compare the performance of UM and CM on NLO. In the comparison, the TS and SPoC are used as default selection and crossover operators. Under this setting, figures 24, 25, and 26 shows the performances inperformance of NLO-GA on 5, 15, and 30 dimensional problems.

4.4.2 Mutation Strategies in TSP

In this part, we mainly compare the performance of SM and SIM in TSP. In comparison, the RWS and DPC are used as default selection and crossover operators. Under this setting, figure 15, 16, and 17 shows the performance of TSP-GA on 50, 100, and 200 size groups.

4.4.3 Conclusion of Mutation Strategies

According to the numerical results, we get the following conclusions for solving NLO and TSP:

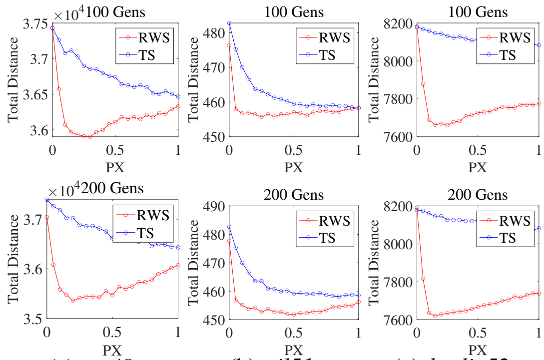
- For NLO, the combination mutation makes a marvel contribution to the computation, which also means that controlling the mutation level according to the fitness value is an efficient way to accelerate the computation. With a high dimension, the CM needs a low p_m to achieve the best performance. But, it always needs a high crossover probability to obtain promising no matter on low-dimensional problems or on high-dimensional problems.
- For TSP, the SM is proved to be an efficient operator to find a better route. And with a lower crossover probability, it works better.

5 CONCLUSIONS

In this paper, we mainly introduce the common genetic operators for typical continuous and discrete optimization problems. In addition, we come up with a combination mutation for NLO. Then for NLO and TSP, we use the numerical experiment to compare the performance of different types of operators. According to the results, we summarize some conclusions about the better usage of these operators, which can directly be used to solve the related problems.

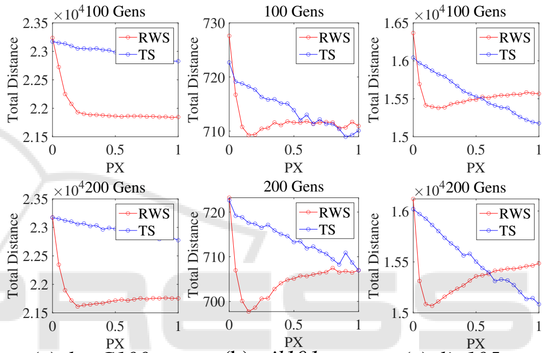
ACKNOWLEDGEMENTS

This research was supported in part by Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea(NRF-2019H1D3A2A01101977) and in part by '5G based IoT Core Technology Development Project' grant funded by the Korea government(MSIT) (No. 2020-0-00167, Core technologies for enhancing wireless connectivity of unlicensed band massive IoT in 5G+ smart city environment).



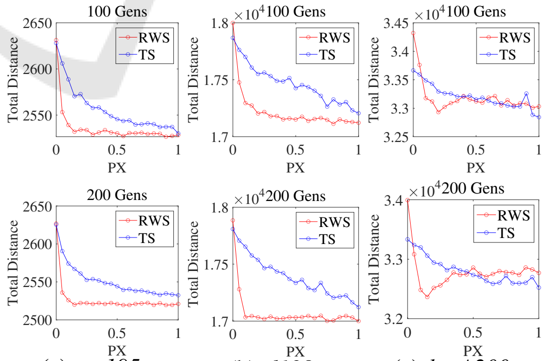
(a) *att48* (b) *eil51* (c) *berlin52*

Figure 9: Performance comparison between two selection strategies (namely, RWS and TS) on TSP instances with about 50 cities.



(a) *kroC100* (b) *eil101* (c) *lin105*

Figure 10: Performance comparison between two selection strategies (namely, RWS and TS) on TSP instances with about 100 cities.

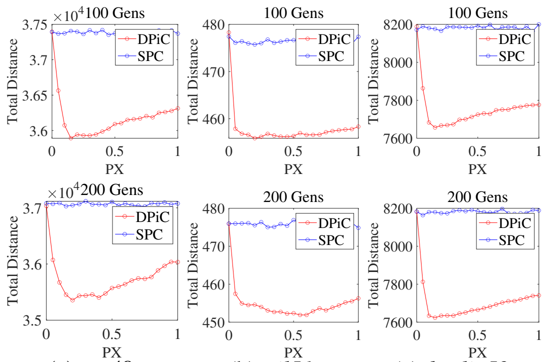


(a) *rat195* (b) *d198* (c) *kroA200*

Figure 11: Performance comparison between two selection strategies (namely, RTW and TS) on TSP instances with about 200 cities.

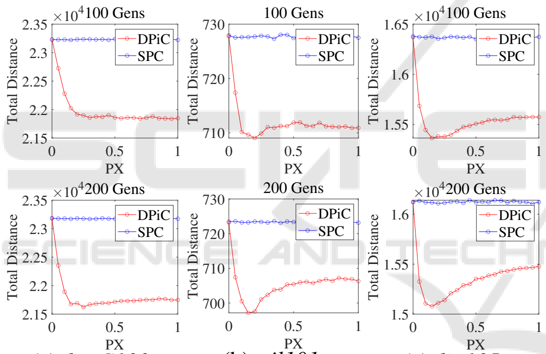
Besides, this work was supported in part by the National Natural Science Foundation of China under Grant 62006124 and 61873097, in part by the Natural Science Foundation of Jiangsu under Project

BK20200811, in part by the Natural Science Foundation of the Jiangshu Higher Education Institutions of China under Grant 20KJB520006, and in part by the Startup Foundation for Introducing Talent of NUIST.



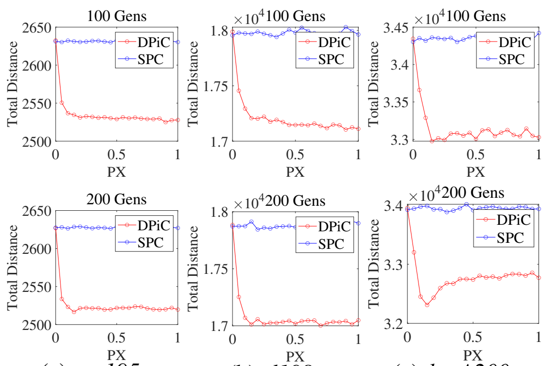
(a) *att48* (b) *eil51* (c) *berlin52*

Figure 12: Performance comparison between two crossover strategies (namely, DPiC and SPC) on TSP instances with about 50 cities.



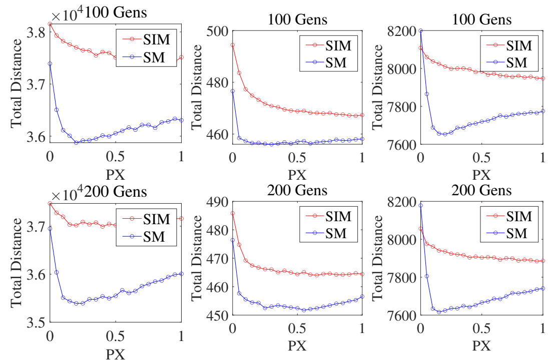
(a) *kroC100* (b) *eil101* (c) *lin105*

Figure 13: Performance comparison between two crossover strategies (namely, DPiC and SPC) on TSP instances with about 100 cities.



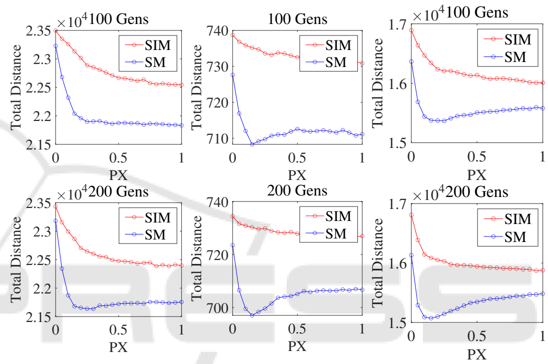
(a) *rat195* (b) *d198* (c) *kroA200*

Figure 14: Performance comparison between two crossover strategies (namely, DPiC and SPC) on TSP instances with about 200 cities.



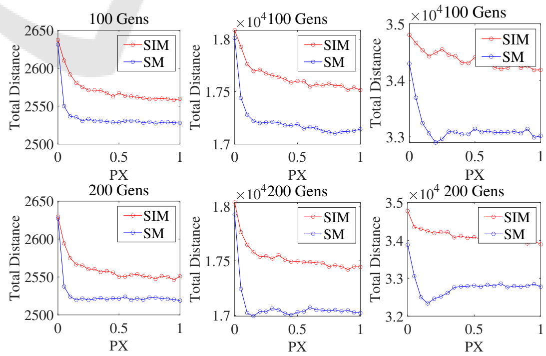
(a) *att48* (b) *eil51* (c) *berlin52*

Figure 15: Performance comparison between two mutation strategies (namely, SIM and SM) on TSP instances with about 50 cities.



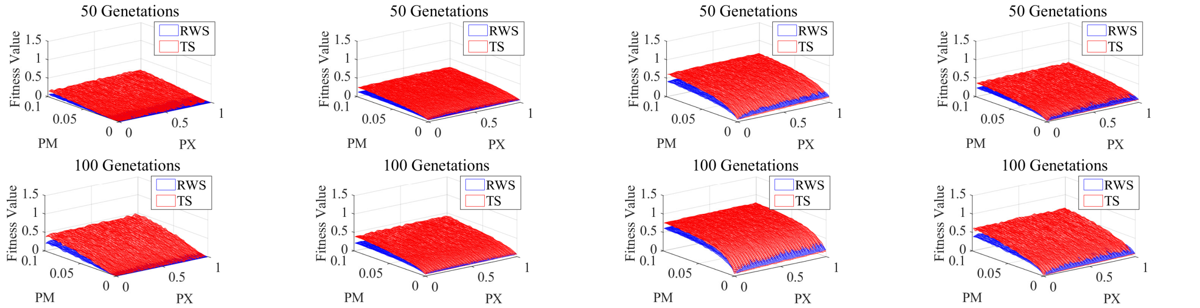
(a) *kroC100* (b) *eil101* (c) *lin105*

Figure 16: Performance comparison between two mutation strategies (namely, SIM and SM) on TSP instances with about 100 cities.



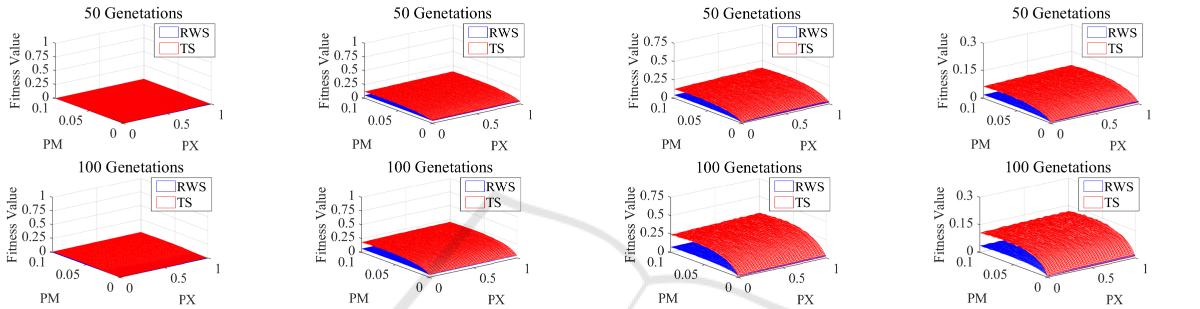
(a) *rat195* (b) *d198* (c) *kroA200*

Figure 17: Performance comparison between two mutation strategies (namely, SIM and SM) on TSP instances with about 200 cities.



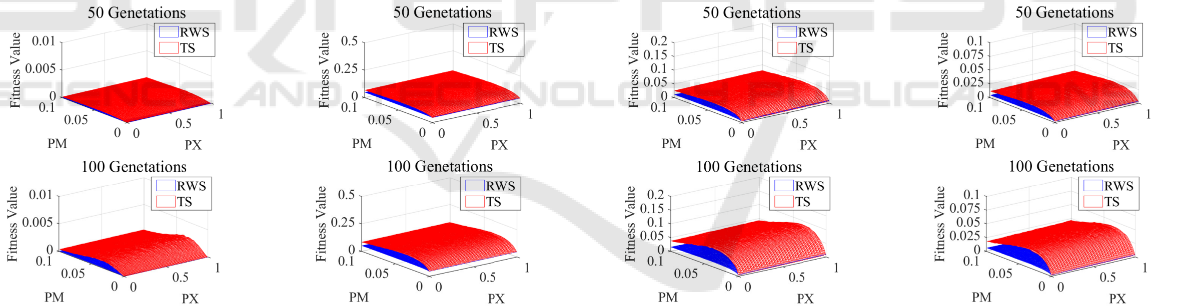
(a) $f_1, D = 5$ (b) $f_2, D = 5$ (c) $f_3, D = 5$ (d) $f_4, D = 5$

Figure 18: Performance comparison between two selection strategies (namely, RWS and TS) on 5-D NLO functions.



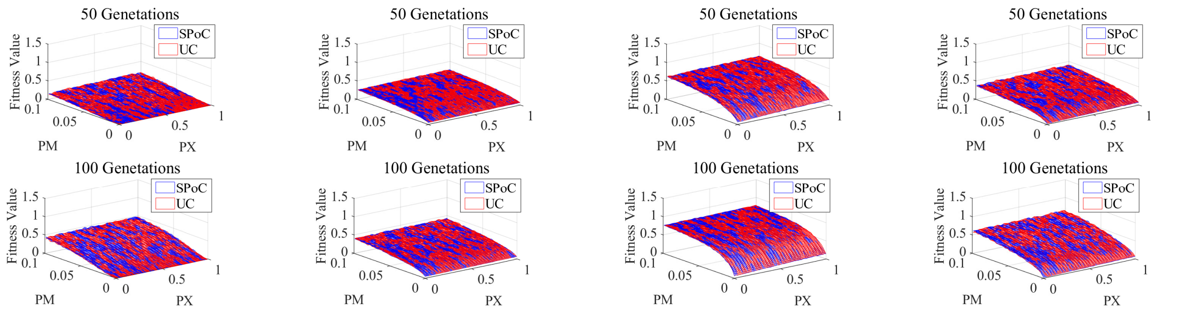
(a) $f_1, D = 15$ (b) $f_2, D = 15$ (c) $f_3, D = 15$ (d) $f_4, D = 15$

Figure 19: Performance comparison between two selection strategies (namely, RWS and TS) on 15-D NLO functions.



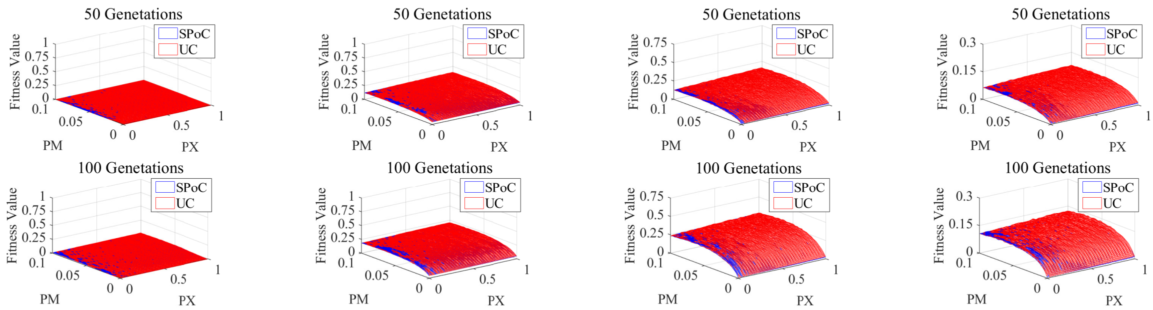
(a) $f_1, D = 30$ (b) $f_2, D = 30$ (c) $f_3, D = 30$ (d) $f_4, D = 30$

Figure 20: Performance comparison between two selection strategies (namely, RWS and TS) on 30-D NLO functions.



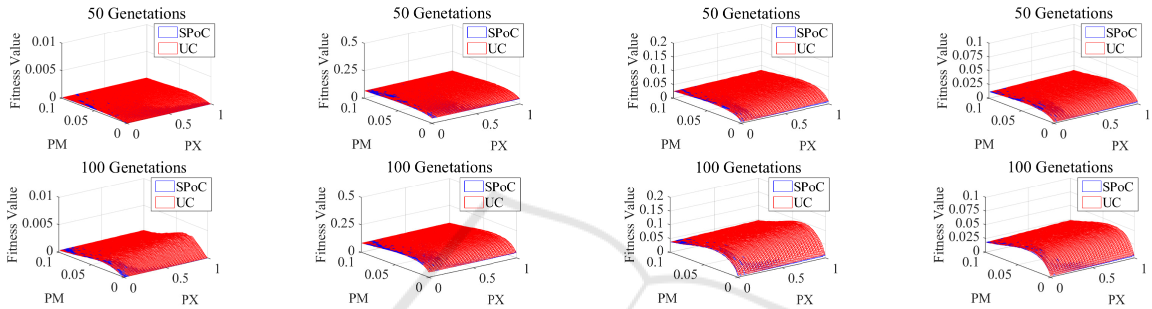
(a) $f_1, D = 5$ (b) $f_2, D = 5$ (c) $f_3, D = 5$ (d) $f_4, D = 5$

Figure 21: Performance comparison between two crossover strategies (namely, SPoC and UC) on 5-D NLO functions.



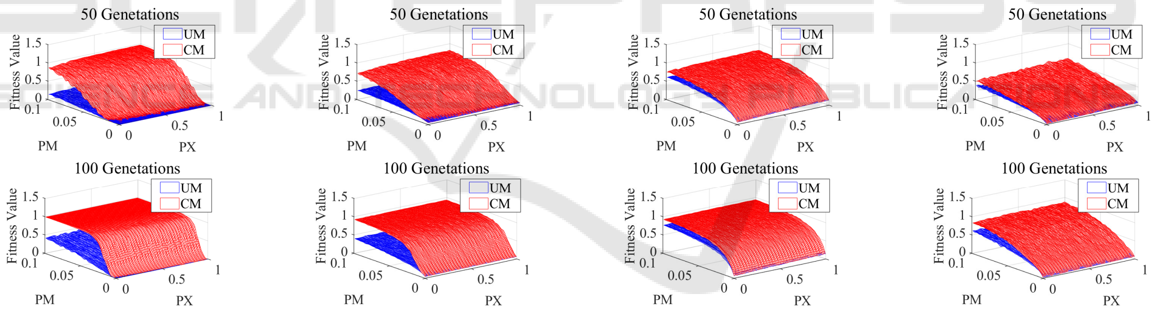
(a) $f_1, D = 15$ (b) $f_2, D = 15$ (c) $f_3, D = 15$ (d) $f_4, D = 15$

Figure 22: Performance comparison between two crossover strategies (namely, SPoC and UC) on 15-D NLO functions.



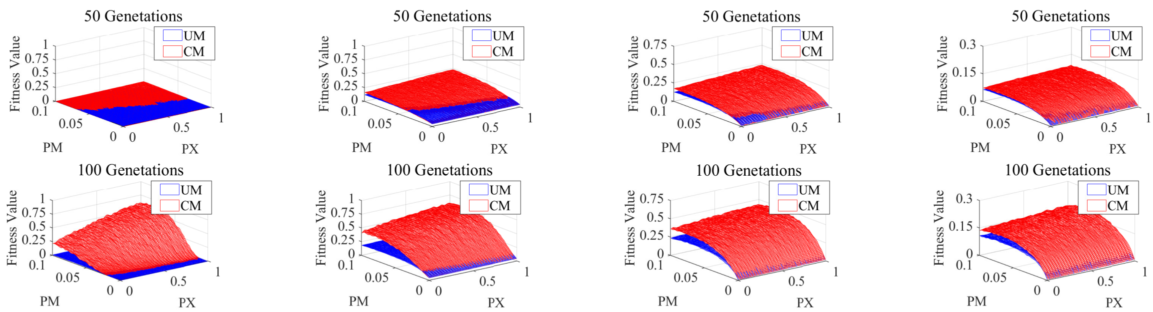
(a) $f_1, D = 30$ (b) $f_2, D = 30$ (c) $f_3, D = 30$ (d) $f_4, D = 30$

Figure 23: Performance comparison between two crossover strategies (namely, SPoC and UC) on 30-D NLO functions.



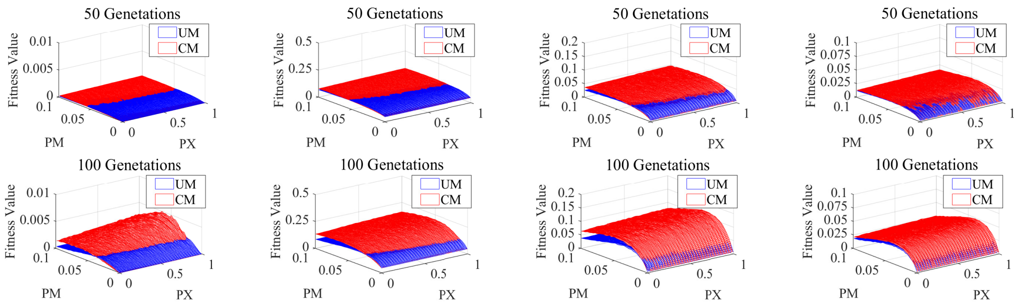
(a) $f_1, D = 5$ (b) $f_2, D = 5$ (c) $f_3, D = 5$ (d) $f_4, D = 5$

Figure 24: Performance comparison between two mutation strategies (namely, UM and CM) on 5-D NLO functions.



(a) $f_1, D = 15$ (b) $f_2, D = 15$ (c) $f_3, D = 15$ (d) $f_4, D = 15$

Figure 25: Performance comparison between two mutation strategies (namely, UM and CM) on 15-D NLO functions.



(a) $f_1, D = 30$ (b) $f_2, D = 30$ (c) $f_3, D = 30$ (d) $f_4, D = 30$

Figure 26: Performance comparison between two mutation strategies (namely, UM and CM) on 30-D NLO functions.

REFERENCES

Almunif, A. and Fan, L. (2017). Mixed integer linear programming and nonlinear programming for optimal pmu placement. In *North Amer. Power Symp.*, pages 1–6.

Chen, J. C., Cao, M., Zhan, Z. H., Liu, D., and Zhang, J. (2020). A new and efficient genetic algorithm with promotion selection operator. In *IEEE Trans. Cybern.*, pages 1532–1537.

Eriksen, B. H. and Breivik, M. (2017). Mpc-based mid-level collision avoidance for asvs using nonlinear programming. In *Proc. IEEE Conf. Control Technol. Appl.*, pages 766–772.

Freisleben, B. and Merz, P. (1996). A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proc. IEEE Int. Conf. Evol. Comput.*, pages 616–621.

Ganganath, N., Cheng, C., Fok, K., and Tse, C. K. (2016). Trajectory planning for 3d printing: A revisit to traveling salesman problem. In *Proc. Int. Conf. Control. Autom. Robot.*, pages 287–290.

Hildayanti, I. K., Soesanti, I., and Permanasari, A. E. (2018). Performance comparison of genetic algorithm operator combinations for optimization problems. In *Proc. Int. Seminar Res. Inf. Technol. Intell. Syst.*, pages 43–48.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

Huang, B., Buckley, B., and Kechadi, T.-M. (2010). Multi-objective feature selection by using nsga-ii for customer churn prediction in telecommunications. *Expert Syst. Appl.*, 37(5):3638 – 3646.

Kaur, D. and Murugappan, M. M. (2008). Performance enhancement in solving traveling salesman problem using hybrid genetic algorithm. In *Proc. Biennial Conf. North Amer. Fuzzy Inform. Process. Soc.*, pages 1–6.

Li, J., Meng, X., Zhou, M., and Dai, X. (2017). A two-stage approach to path planning and collision avoidance of multibridge machining systems. *IEEE Trans. Syst. Man Cybern. Syst.*, 47(7):1039–1049.

Li, Y., Zhang, S., and Zeng, X. (2009). Research of multi-population agent genetic algorithm for feature selection. *Expert Syst. Appl.*, 36(9):11570 – 11581.

Ma, F., Xu, Y., and Xu, P. (2017). A nonlinear programming based universal optimization model of tdoa passive location. In *Proc. Int. Conf. Intell. Syst. Knowl. Eng.*, pages 1–3.

Pinho, R. and Saraiva, F. (2020). A comparison of crossover operators in genetic algorithms for switch allocation problem in power distribution systems. In *Proc. IEEE Congr. Evol. Comput.*, pages 1–8.

Wang, Z., Fang, X., Li, H., and Jin, H. (2020). An improved partheno-genetic algorithm with reproduction mechanism for the multiple traveling salesperson problem. *IEEE Access*, 8:102607–102615.

Xie, J., Carrillo, L. R. G., and Jin, L. (2019). An integrated traveling salesman and coverage path planning problem for unmanned aircraft systems. *IEEE Control Syst*, 3(1):67–72.

Yang, Q., Chen, W., Gu, T., Zhang, H., Yuan, H., Kwong, S., and Zhang, J. (2020). A distributed swarm optimizer with adaptive communication for large-scale optimization. *IEEE Transactions on Cybernetics*, 50(7):3393–3408.

Yang, Q., Chen, W., Li, Y., Chen, C. L. P., Xu, X., and Zhang, J. (2017a). Multimodal estimation of distribution algorithms. *IEEE Transactions on Cybernetics*, 47(3):636–650.

Yang, Q., Chen, W., Yu, Z., Gu, T., Li, Y., Zhang, H., and Zhang, J. (2017b). Adaptive multimodal continuous ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 21(2):191–205.

Yu, F., Fu, X., Li, H., and Dong, G. (2016). Improved roulette wheel selection-based genetic algorithm for tsp. In *Proc. Int. Conf. Netw. Inf. Syst. Comput.*, pages 151–154.

Yu, Y., Chen, Y., and Li, T. (2011). A new design of genetic algorithm for solving tsp. In *Proc. Int. Joint Conf. Comput. Sci. Optim.*, pages 309–313.

Zhong, J., Hu, X., Zhang, J., and Gu, M. (2005). Comparison of performance between different selection strategies on simple genetic algorithms. In *Proc. IEEE Int. Conf. Comput. Intell. Modelling Control Automat. Int. Conf. Intell. Agents Web Technol. Internet Commerce*, volume 2, pages 1115–1121.

Zhou, H. and Song, M. (2016). An improvement of partheno-genetic algorithm to solve multiple travelling salesmen problem. In *Proc. IEEE/ACIS Int. Conf. Comput. Inf. Sci.*, pages 1–6.

Zorlu, O., Dilek, S., and Özsoy, A. (2017). Gpu-based parallel genetic algorithm for increasing the coverage of wsns. In *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, pages 640–647.