Research Article

A Hybrid Cellular Genetic Algorithm for the Traveling Salesman Problem

Yanlan Deng, Juxia Xiong, and Qiuhong Wang

School of Mathematics and Physics, Guangxi University for Nationalities, Nanning 530006, China

Correspondence should be addressed to Juxia Xiong; xiongjuxia@gxun.edu.cn

Received 3 December 2020; Revised 22 January 2021; Accepted 28 January 2021; Published 9 February 2021

Academic Editor: Ali Asghar Rahmani Hosseinabadi

Copyright © 2021 Yanlan Deng et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The traveling salesman problem (TSP), a typical non-deterministic polynomial (NP) hard problem, has been used in many engineering applications. Genetic algorithms are useful for NP-hard problems, especially the traveling salesman problem. However, it has some issues for solving TSP, including quickly falling into the local optimum and an insufficient optimization precision. To address TSP effectively, this paper proposes a hybrid Cellular Genetic Algorithm with Simulated Annealing (SA) Algorithm (SCGA). Firstly, SCGA is an improved Genetic Algorithm (GA) based on the Cellular Automata (CA). The selection operation in SCGA is performed according to the state of the cell. Secondly, SCGA, combined with SA, introduces an elitist strategy to improve the speed of the convergence. Finally, the proposed algorithm is tested against 13 standard benchmark instances from the TSPLIB to confirm the performance of the three cellular automata rules. The experimental results show that, in most instances, the results obtained by SCGA using rule 2 are better and more stable than the results of using rule 1 and rule 3. At the same time, we compared the experimental results with GA, SA, and Cellular Genetic Algorithm (CGA) to verify the performance of SCGA. The comparison results show that the distance obtained by the proposed algorithm is shortened by a mean of 7% compared with the other three algorithms, which is closer to the theoretical optimal value and has good robustness.

1. Introduction

Traveling salesman problem (TSP) is one of the most common combinatorial optimization problems, and it has been widely used in this field. At the same time, many problems in the field of combinatorial optimization can be formulated as special TSP instances [1]. For example, resource optimization in the shortest path and shop scheduling problems are the most frequent TSP applications [2–4]. In Word Sense Disambiguation (WSD), problems can also be solved by describing WSD as a TSP variant [5]. In the global positioning system (GPS), TSP can also be used as the basis for finding route problems [6]. It can be seen that the application of TSP is extensive; however, TSP is an NP-hart problem, and the number of feasible solutions increases significantly with the increase of nodes, which brings great difficulty to the solution [7]. If the exact algorithm is used to solve TSP, it will take a long time, so the feasibility of using the exact algorithm to solve TSP is very low. On the other hand, although using an approximate solution to solve TSP problems cannot guarantee an optimal solution, it can reach a satisfactory solution in a very short time [8]. Thus, with the development of heuristic algorithms [9–14], many experts and scholars have begun to apply heuristic algorithms to solve TSP, which provides new ideas for solving TSP [15–21].

Genetic algorithm (GA) is an evolutionary algorithm proposed by Professor John H. Holland of the University of Michigan and his students from the late 1960s to the early 1970s [22]. GA uses biological evolution and genetic principles to imitate the generation and evolution process of all life and intelligence by drawing on the basis of biology. It is a bionic algorithm in a macroscopic sense and is applied to many optimization fields [23–26]. The coding method commonly used in genetic algorithms is binary coding, but most search space sequences do not correspond to feasible travel when binary coding is used. We can use ordinal coding and a partially mapped crossover (PMX) as the cross operator so that all the solutions obtained will be feasible solutions [27]. However, there are some disadvantages in using traditional GA to solve TSP, such as poor searchability...
and low convergence accuracy. On this basis, the improved GA also developed vigorously. Liu et al. [28] proposed an improved GA with decision cut-off algebra to solve TSP, which improves the convergence of the GA. Besides, Yu et al. [29] proposed an improved GA for solving TSP by a greedy algorithm, which used the greedy algorithm to generate the initial population and reduce the number of iterations. At the same time, the hybrid algorithm has also received the attention of many experts and scholars. For instance, Wang et al. [30] combined GA and SA, and proposed an improved simulated annealing genetic algorithm for solving TSP, improving the local search capabilities. Also, Zhang et al. [31] combined the GA and Firefly algorithm proposing a Firefly GA algorithm to solve TSP and prevent the algorithm from falling into local optimality. Moreover, Tao et al. [32] combined the GA and Ant Colony (AC) algorithm proposing a dynamic Ant Colony genetic algorithm to solve TSP, which improved the reininsertion of offspring to improve stability.

Cellular genetic algorithm (CGA) is a hybrid of cellular automata and genetic algorithms. It has been applied to various decision-making problems [33]. However, the application of the CGA to TSP is very limited at present. To study the application of the CGA in TSP and improve GA’s optimization performance, this paper proposes a hybrid Cellular Genetic Algorithm with SA (SCGA). We improve TSP optimization performance by combining the cellular genetic algorithm and the simulated annealing algorithm. The contributions of this paper are as follows:

(i) The paper successfully applies the cellular genetic algorithm to solve TSP and proposed a hybrid Cellular Genetic Algorithm combined with a simulated annealing algorithm. The experimental results show that the hybrid cellular genetic algorithm’s optimization performance is better than GA, SA, and CGA.

(ii) The optimization performance of the three cellular automata rules in the SCGA is examined through experiments and concluded that cellular automata rules with a moderate total number of living cells are beneficial to algorithm optimization.

The rest of this article is arranged as follows. Section 2 is a brief introduction to the preliminary knowledge that will be used, including the traveling salesman problem model, the basic content of genetic algorithm (GA), simulated annealing (SA) algorithm, and cellular automata (CA). In Section 3, the proposed algorithm and its steps are explained in detail. Section 4 reveals the simulation results and discussions. Section 5 summarizes the full text and proposes the next step.

2. Preliminaries

2.1. Traveling Salesman Problem (TSP). TSP is a commonly used test benchmark for optimization methods, which can be simply described as follows.

Given the coordinates of \( n \) cities, starting from any city, passing through the remaining cities only once, and finally returning to the starting city, there are \( n! \) routes. We need to choose the shortest route from these \( n! \) routes. In the symmetric Euclidean TSP, city \( A \)'s distance to city \( B \) is equal to the distance from city \( B \) to city \( A \). According to the description of TSP, the following mathematical model can be established [29]:

\[
R = (N_1, N_2, \ldots, N_n), \quad \min f(R),
\]

\[
f(R) = \sum_{i=1}^{n-1} d(N_i, N_{i+1}) + d(N_n, N_1),
\]

\[
d(N_i, N_{i+1}) = d(N_{i+1}, N_i).
\]

We use the city number as the city mark. The solution \((1, 2, 3, \ldots, n)\) represents a trip starting from city 1, along city 2, city 3, until city \( n \), and finally back to city 1. \((N_1, N_2, \ldots, N_n)\) is a permutation of \((1, 2, 3, \ldots, n)\), and \(d(N_i, N_{i+1})\) represents the Euclidean distance between city \( N_i \) and city \( N_{i+1} \).

2.2. Genetic Algorithm (GA). Genetic algorithm is one of the universal algorithms in the optimization field. The essence of GA is an efficient, parallel, and global search method. It can automatically acquire and accumulate search knowledge during the search process and adaptively control the search process to find the optimal solution [27, 34]. The traditional GA is mainly composed of population and population size, coding method, selection strategy, genetic operator, and stopping criterion [35–37]. The population is composed of chromosomes or individuals, and each individual corresponds to a feasible solution to the problem. The population size refers to the number of individuals in the population, which is a subset of the problem’s feasible region. Generally speaking, the larger the number of individuals in the population is, the more feasible solutions it contains, and the better the optimal solution will be. However, as the number of individuals in the population increases, the algorithm’s computation time will also increase, which affects the performance of the algorithm. Therefore, it is necessary to select an appropriate population size. The coding method is also called the gene expression method. The chromosomes in the population are composed of genes. Correctly coding the chromosomes is the basic and important work of GA. According to the mathematical model of TSP, we can choose the ordinal encoding as the encoding method, and each chromosome is composed of a nonrepetitive ordinal number \((1, 2, 3, \ldots, n)\). The selection strategy is the process of selecting parents from the current population. We can use the commonly used roulette as the selection operator. The pseudocode of the roulette operator is shown in Algorithm 1, and the steps of the roulette operator are as follows:

(i) Step 1. Calculate the number \(NP\) of individuals to be selected according to the selection probability \(P_s\).

(ii) Step 2. Calculate the fitness value of each individual according to the following equation:
**Mathematical Problems in Engineering**

\[ f_k = \frac{1}{\sum_{i=1}^{n-1} d(N_i, N_{i+1}) + d(N_n, N_1)}. \]

(iii) Step 3. Calculate the cumulative probability of an individual \( P_k \):

\[ P_k = \frac{\sum_{i=1}^{k} f_i}{\sum_{j=1}^{n} f_j}. \]

(iv) Step 4. Randomly generate the number \( r \) in \([0, 1]\); if \( P_{i-1} < r \) and \( r < P_i \), select the \( i^{th} \) individual. Repeat \( NP \) times to get the selected \( NP \) individuals.

Genetic operators mainly include crossover and mutation, which is the most important part of the GA. We choose partially mapped crossover (PMX) and reverse order mutation to hide the infeasible solutions in the execution process. Reverse order mutation refers to reversing the order of the numbers between two tangent points. Therefore, the numbers will not change, and there will be no duplicate numbers. Figure 1 is a schematic diagram of PMX, and the steps of PMX are as follows:

*Step 1.* Randomly generate two cut points.

*Step 2.* Exchange the part between two individual tangent points.

*Step 3.* Determine the mapping relationship.

*Step 4.* Restore the legitimacy of the individual according to the mapping relationship.

The stopping criterion generally uses the maximum number of iterations, and the algorithm stops when the maximum number of iterations is reached. Algorithm 2 shows the pseudocode of the GA for solving TSP.

### 2.3. Simulated Annealing (SA) Algorithm

In 1953, Metropolis et al. [38] first proposed the simulated annealing algorithm (SA). In 1983, Kirkpatrick et al. [39] successfully used SA in combinatorial optimization problems, especially large-scale problems. SA is a general random search algorithm, which is an extension of the local search algorithm. It is derived from the simulation of the annealing process in thermodynamics [40]. Because modern SA can effectively solve NP complexity problems, avoid falling into local optima, and overcome initial value dependence, it has been widely used in engineering, production scheduling, control engineering, machine learning, and other fields [41–43]. SA mainly includes state expression, neighborhood definition, thermal equilibrium, and cooling control [39]. State expression is to describe the system’s energy state in a mathematical form, and one state corresponds to one solution of the problem. State expression is the main work of SA, which directly determines the neighborhood structure and size. A reasonable state expression method can reduce computational complexity and improve algorithm performance. Consistent with genetic algorithms, nonrepeated ordinal numbers are used to represent solutions. The neighborhood definition’s starting point is to ensure that the neighborhood’s solutions are distributed as much as possible in the entire solution space. The nature of the problem usually determines the definition of the neighborhood. Consistent with the mutation operator in GA, we use reverse order to form neighbor solutions. To replace the neighbor solution with the original solution is determined by the Metropolis criterion. Algorithm 3 shows the pseudocode of the Metropolis criterion. The thermal equilibrium is equivalent to the isothermal process in the physical annealing, which is the internal cycle process in SA. To ensure the equilibrium state, the number of internal cycles should be large enough, leading to an increase in computation time. The number of internal cycles is the length of the Markov chain L. Cooling control is the outer cycle process of SA. Unlike GA, SA uses a temperature drop to control the iteration of the algorithm. The algorithm stops when the temperature drops to the end temperature. The pseudocode of SA for solving TSP is shown in Algorithm 4, and the steps of SA for solving TSP are as follows:

*Step 1.* Set initial parameters (including initial temperature, Markov chain length, cooling rate, and termination temperature.), and randomly generate an initial solution.

*Step 2.* Using the reverse order method to generate a neighbor solution.

*Step 3.* Move the neighbor according to Metropolis criterion.

*Step 4.* Judge whether the thermal equilibrium is reached; if the thermal equilibrium is not reached, go to Step 2.

*Step 5.* Lower the temperature. If the temperature reaches the end temperature, the algorithm will stop; otherwise, go to Step 2.

### 2.4. Cellular Automata (CA)

In 1948, Stanislaw M. Ulam and von Neumann proposed cellular automata (CA). CA is a simplified mathematical model for describing complex phenomena in nature [44]. CA reflects the law of self-replication of life through relatively simple rules. The realization of self-replication of life on the computer has brought a new revolution in computing and simulation fields. As a discrete model of a complex system, CA is an important experimental method for studying dynamic interaction and spatiotemporal evolution [45–47]. CA is composed of four parts: cell, cellular space, cellular neighbor type, and cellular automata rule [44].

The choice of cellular neighbor’s type and cellular automata rule is a crucial part that affects CA’s performance. Here, the Moore neighborhood is utilized. The Moore neighborhood consists of a central cell (the one which is to be updated) and its eight geographical neighbors, north, west, south, east, north-east, north-west, south-west, and south-east, that is a total of nine cells [48]. Each central cell can be alive (\( s = 1 \)) or dead (\( s = 0 \)). If permutations and combinations are used to describe neighbors’ states, many unnecessary duplicate
data will be generated. Therefore, there is no need to
enumerate the states and positions of specific neighbors;
just calculate the total number of cellular states in
neighbors [49]. The sum of neighbors’ states is \( K \); so, \( K \)
can be any value between 0 and 8. Cellular automata rules
[50] are as follows:

**Algorithm 1:** The pseudocode of the roulette operator.

```
Roulette operator
Begin
   Input: population, selection probability \( P_c \)
   Calculate \( NP, f_k, P_k, (k = 1, 2, \ldots, n) \)
   for \( i = 1 \) to \( NP \)
      if \( \text{rand} < P_i \) and \( \text{rand} > P_{i-1} \)
         select the \( i^{th} \) individual
   End if
End for
Output: the selected individual.
End
```

**Algorithm 2:** The pseudocode of genetic algorithm (GA) for solving TSP.

```
Genetic algorithm (GA) for solving TSP
Begin
   Input: city coordinates: \( X = \{x_1, x_2, \ldots, x_n\} \), \( Y = \{y_1, y_2, \ldots, y_n\} \), selection probability \( P_s \), crossover probability \( P_c \), mutation probability \( P_m \), the maximum number of iterations (MaxIter)
   Initialization: using ordinal coding to generate the initial population, number of iterations \( k = 1 \)
   while \( k \leq \text{MaxIter} \)
      Selection
      Crossover
      Mutation
      New population
      Save the optimal result in the iteration
      \( k = k + 1 \)
   End while
   Output: the optimal result: \( R = \{N_1, N_2, \ldots, N_n\} \), \( f(R) \)
End
```

**Algorithm 3:** The pseudocode of the Metropolis criterion.

```
Metropolis criterion
Begin
   Input: city coordinates: \( X = \{x_1, x_2, \ldots, x_n\} \), \( Y = \{y_1, y_2, \ldots, y_n\} \). Solution \( S_1 \) and neighbor solution \( S_2 \), current temperature \( T \)
   Calculate \( f(S_1) \) and \( f(S_2) \) by equation (2)
   \( \Delta f = f(S_2) - f(S_1) \), \( P = \exp(-\Delta f/T) \)
   if \( \Delta f < 0 \) or \( P > \text{rand} \)
      \( S_1 = S_2 \)
   End if
Output: \( S_1 \)
End
```
3. A Hybrid Cellular Genetic Algorithm (SCGA)

This section describes the main contents of SCGA in detail, including the dynamic evolution of cellular space, genetic operations, elitist strategy, and the steps of SCGA.

3.1. Dynamic Evolution of Cellular Space. A state matrix in the cellular space is composed of each cell state in the cellular space. In the iteration, each cell in the cellular space is used as the central cell, and the state of the central cell is updated using the cellular automata rules to form a new state matrix.

The pseudocode of the dynamic evolution of the cellular space is shown in Algorithm 5. Given a 10 × 10 cellular space arbitrarily, each cell state is randomly generated to form an initial state matrix (an n-th order square matrix N composed of 0 and 1), as shown in Figure 2. The cellular automata rule determines each cell’s updated state, for example, if the cell in the third row and the third column is the central cell. The central cell state is 1, and there are four cells in the neighboring cells with states equal to 1. According to rule 1, the central cell update state will be 0, while the central cell updated state will be 1, according to rule 2 or rule 3. It can be seen that the updated cellular state changes according to different cellular automata rules. Because dead cells do not perform genetic operations, we only consider the number of living cells. Figure 3 shows that when different cellular automata rules are selected, the total number of living cells will differ. Therefore, if rule 1 is chosen, the total number of living cells will eventually remain at about 10%, while if rule 2 is selected, the total number of living cells will eventually remain at about 38%. On the other hand, if rule 3 is selected, the total number of living cells will eventually remain in a range of 50%. The total number of living cells in the cellular space will affect the optimization performance of the algorithm. Aiming at the TSP, we will compare the optimal solutions obtained by applying these three rules in the hybrid cellular genetic algorithm through experiments.

3.2. Genetic Operations. SCGA puts the individual as a cell into the cellular space, selects the living cell, and forms the parents with the best cell among neighbors. Then, the parents perform crossover and mutation operations. If the cellular state is dead, no crossover and mutation operations are performed. Algorithm 6 shows the pseudocode for the genetic operations.

Assuming that the number of cities is 10 and the number of populations is 100, the corresponding cellular space is a 2D grid of 10 × 10 units. The algorithm works as follows:

(i) Randomly generate 100 individuals to form the initial population and associate the individuals with the cells one-to-one

(ii) If Figure 2 is used as the state matrix of the cellular space at the initial moment, the individual is represented by the letter. The number after the letter represents the state of the cell, as shown in Figure 4
(iii) Take the individual V in the third row and second column as the central cell
(iv) In the current iteration, the cellular state is living (state 1)
(v) Tables 1 and 2 show that the individual with the shortest distance among neighbors is AE, so choose AE and V and perform crossover and mutation operations as parents:

Firstly, the parents can get the offspring individuals O1 and O2 after PMX. Perform the reverse mutation on O1 and O2 to get the individuals O3 and O4, as shown in Table 3. The new individuals obtained after genetic manipulation are O3 and O4; the best progeny is O3, and O3 is better than V; so O3 replaces V to complete the genetic operation.

3.3. Elitist Strategy. The living central cell and the best neighbor get offspring after crossover and mutation operations. If the distance between the optimal progeny and central cell is not compared, and the optimal progeny directly replaces the central cell, the inferior solution may enter the population. If the central cell’s distance is better than the optimal progeny distance, and the central cell is not replaced, it may fall into a local optimum. In order to solve
Genetic operations
Begin
Input: An n-th order square matrix \( N \) composed of 0 and 1, cellular space,
city coordinates: \( X = \{x_1, x_2, \ldots, x_n\}, Y = \{y_1, y_2, \ldots, y_n\} \)
Calculate the total distance \( f_i(R) \) of each individual in the cellular space by Equation (2)
for \( i = 1 \) to \( n^2 \)
if \( N(i) = 1 \)
Select the \( i \)-th individual and the individual with smallest total distance among the 8 neighbors
Crossover
Mutation
New the \( i \)-th individual
End if
End for
Output: New cellular space
End

Algorithm 6: The pseudocode for genetic operations.

| A0 | B1 | C1 | D0 | E1 | F1 | G1 | H1 | I0 | J0 |
|----|----|----|----|----|----|----|----|----|----|
| K1 | L1 | M0 | N1 | O0 | P0 | Q0 | R0 | S1 | T0 |
| U0 | V1 | W1 | X0 | Y1 | Z0 | AA | AB0| AC1| AD0|
| AE0| AF0| AG0| AH1| AI0| AJ1| AK0| AL1| AM0| AN0|
| AO1| AP0| AQ1| AR1| AS1| AT1| AU0| AV0| AW1| AX1|
| AY0| AZ0| BA1| BB1| BC0| BD0| BE0| BF0| BG1| BH0|
| BI0| BJ0| BK1| BL1| BM1| BN0| BO0| BP1| BQ0| BR0|
| BS1| BT1| BU0| BV0| BW0| BX1| BY0| BZ0| CA1| CB0|
| CC0| CD1| CE0| CF0| CG0| CH0| CI0| CJ1| CK1| CL1|
| CM0| CN0| CO0| CP1| CQ0| CR0| CS1| CT0| CU1| CV0|

Figure 4: Cell and state matrix.

Table 1: Central cell and neighbors.

| Cell | Route | Distance | Cell | Route | Distance | Cell | Route | Distance |
|------|-------|----------|------|-------|----------|------|-------|----------|
| K    | 34697181025 | 316.0711 | U    | 62489315710 | 344.9062 | AE    | 78193510426 | 244.1511 |
| L    | 89461052713 | 262.5796 | V    | 43271068159 | 290.6597 | AF    | 23846109751 | 267.6944 |
| M    | 48127391065 | 260.2487 | W    | 24857361910 | 299.5755 | AG    | 11048657293 | 288.2223 |

Table 2: City coordinates.

| Number | x    | y    |
|--------|------|------|
| 1      | 37   | 52   |
| 2      | 49   | 52   |
| 3      | 52   | 64   |
| 4      | 20   | 26   |
| 5      | 40   | 30   |
| 6      | 21   | 47   |
| 7      | 17   | 63   |
| 8      | 31   | 62   |
| 9      | 52   | 33   |
| 10     | 51   | 21   |

Table 3: Individuals in crossover and mutation process.

| Cell | Route | Distance | Cell | Route | Distance |
|------|-------|----------|------|-------|----------|
| V    | 43271068159 | 290.6597 | O2   | 93271068415 | 296.2896 |
| AE   | 78193510426 | 244.1511 | O3   | 18493510267 | 258.7903 |
| O1   | 48193510267 | 264.5371 | O4   | 94861072315 | 286.1712 |
the above problems, SCGA sets an elite individual retention strategy after the crossover and mutation operations. When the optimal progeny’s obtained distance after the crossover and mutation operation is less than the distance of the center cell, the center cell is directly replaced with the optimal progeny. Otherwise, after performing the simulated annealing operation on the optimal progeny, new offspring are obtained, and the central cell is replaced with the new offspring. Algorithm 7 shows the pseudocode of the elitist strategy. Through the elitist strategy, the living central cell can be optimized as much as possible to avoid falling into the local optimum.

3.4. Process of SCGA. SCGA, as explained before, has three main contents: the dynamic evolution of cellular space, genetic operations, and elitist strategy. In this section, we will introduce the main steps of applying SCGA to solve TSP. Algorithm 8 shows the pseudocode of the proposed hybrid Cellular Genetic Algorithm with SA (SCGA) and its required steps for the TSP solution.

(i) Step 1. Population initialization:
   (a) Initialize the following parameters:
   (i) Crossover probability $P_c$,
   (ii) Mutation probability $P_m$,
   (iii) Cooling rate $r$,
   (iv) Initial temperature $T_0$,
   (v) Population size $Num$,
   (vi) The size of cellular space Num. $C$,
   (vii) Maximum iteration times Maxiter,
   (viii) Markov chain length $L$,
   (ix) Termination temperature $T_t$.
   (b) Randomly generate the initial population as cellular space and state matrix (an $n$-th order square matrix $N$ composed of 0 and 1).

(ii) Step 2. Calculate the city distance matrix.

(iii) Step 3. Calculate the total distance of individuals in the population.

(iv) Step 4. Take each cell as the central cell in turn.
   (c) If the central cell state is 1, then it is the central cell and the best cell from neighbors to form the parents. Otherwise, skip to Step 7.

(v) Step 5. Perform crossover and mutation operations on the parents to get the offspring.

(vi) Step 6. If the distance of the optimal progeny is longer than the distance of the central cell, the simulated annealing algorithm is used to optimize the optimal progeny and replace the central cell after optimization. Otherwise, the optimal progeny will directly replace the central cell.

(vii) Step 7. Update the state matrix according to the cellular automata rule.

(viii) Step 8. Calculate the distance of individuals in the current population and store the best individuals.

(ix) Step 9. Judge the stop condition.

(d) Stop when the maximum number of iterations is reached. Otherwise, skip to Step 4.

4. Experimental Results and Discussion

Here, instances in the TSPLIB test library are selected, and GA, SA, CGA, and SCGA (cellular automata rules 1–3) are used to solve these instances. All of the proposed algorithms are implemented using MATLAB R2018b using a laptop equipped with Windows 10 Ultimate 64 bit operating system, processor Intel Core i7-10510U CPU, 12 GB of RAM, and 1.80 GHz processor speed.

4.1. Experiments Settings. 13 instances from TSPLIB are selected for simulation experiments. To avoid the influence of randomness, the results of the experiments on each algorithm for each instance were taken independently over 10 repeated times. The result is expressed by the route distance obtained from the experiment. The basic parameters of SCGA include crossover probability $P_c = 0.9$, mutation probability $P_m = 0.05$, cooling rate $r = 0.9$, and initial temperature $T_0 = 100$. The other parameters are shown in Table 4, where Num, Num. C, Maxiter, $L$, and $T_t$ represent population size, the size of cellular space, maximum iteration times, Markov chain length, and termination temperature values which are identified. The basic parameters of GA are initialized as the population size $Num = 100$, generation gap GGAP = 0.9, and maximum iteration times Maxiter = 10000. The basic parameters of SA are also initialized as Markov chain length $L = 500$ and termination temperature $T_t = 0.001$. The basic parameters of CGA are assigned initial values as mutation probability $P_m = 0.5$ and maximum iteration times Maxiter = 10000. The other parameters that are not mentioned are the same as those of SCGA.

4.2. Comparison of Cellular Automata Rules. In order to test the optimization of the three cellular automata rules, the experiment is performed using rule 1, rule 2, and rule 3 in SCGA using the same parameters applied in the previous experiments. Figure 5 shows the optimal paths of eil51, pr107, bier127, and kroa200. Table 5 shows the best solutions, the worst solutions, the mean value, and the standard deviation of 10 repeated experiments. The boldface in the table represents the optimal value. As can be seen, in the instances of att48, berlin52, chn31, oliver30, pr107, and pr152, the best solutions obtained using rule 1, rule 2, and rule 3 are equal. In the instances of bier127, ch130, and kroa100, the best solutions obtained using rule 2 and rule 3 are also equal, and both are better than the best solutions obtained using rule 1. In the instances of eil51 and eil76, the best solutions obtained using rule 2 are better than the best solutions obtained using rule 1 and rule 3. Only in the instance of pr136, the best solutions obtained by using rule 1 and rule 3 are better than the best solutions obtained using rule 2. In the instance of kroa200, the best solution obtained using rule 3 is better than the best solution obtained using rule 2. In most instances, the worst solutions obtained using rule 2 are better than the worst solutions obtained using rule
1 and rule 3. Only in the instances of kroa100 and pr136, the worst solutions obtained using rule 1 and rule 3 are better than the worst solutions obtained using rule 2. At the same time, in 10 repeated experiments, the mean value and the standard deviation computed using rule 2 are better than the mean value and the standard deviation computing using rule Elitist strategy

Begin
Input: Central cell $S_1$ and new individual $S_2$
City coordinates: $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$
Calculate $f(S_1)$ and $f(S_2)$ by Equation (2)
if $f(S_1) > f(S_2)$
$S_1 = S_2$
else
Optimize $S_2$ using simulated annealing algorithm
$S_1 = S_2$
End if
Output: $S_I$
End

Algorithm 7: The pseudocode of elitist strategy.

The hybrid cellular genetic algorithm (SCGA) for solving TSP

Begin
Input: city coordinates: $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, $P_c$, $P_m$, $r$, $T_0$, Num, Num. C, Maxiter, $L$, $T_t$
Initialization: using ordinal coding to generate randomly the initial population as cellular space, generate randomly an n-th order square matrix $N$ composed of 0 and 1 as state matrix, number of iteration $k = 1$
while $k \leq$ Maxiter
for $i = 1$ to $n^2$
if $N(i) = 1$
Genetic operations
Elitist strategy
End if
Save the optimal result in the iteration
End for
$k = k + 1$
End while
Output: the optimal result: $R = \{N_1, N_2, \ldots, N_n\}$, $f(R)$
End

Algorithm 8: The pseudocode of the hybrid cellular genetic algorithm (SCGA) for solving TSP.

| Instance      | Num | Num.C | Maxiter | $L$  | $T_t$ |
|---------------|-----|-------|---------|------|-------|
| att48         | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| berlin52      | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| bier127       | 1600| $40 \times 40$ | 100     | 500  | 0.001 |
| ch130         | 1600| $40 \times 40$ | 100     | 500  | 0.001 |
| chn31         | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| el51          | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| el76          | 900 | $30 \times 30$ | 100     | 500  | 0.01  |
| kroa100       | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| kroa200       | 1600| $40 \times 40$ | 100     | 500  | 0.001 |
| oliver30      | 400 | $20 \times 20$ | 50      | 100  | 0.01  |
| pr107         | 1600| $40 \times 40$ | 50      | 500  | 0.001 |
| pr136         | 1600| $40 \times 40$ | 100     | 500  | 0.001 |
| pr152         | 1600| $40 \times 40$ | 100     | 500  | 0.001 |

Table 4: The parameters of instances.
The optimal path of eil51
(a)

The optimal path of pr107
(b)

The optimal path of bier127
(c)

The optimal path of kroa200
(d)

Figure 5: The optimal paths: (a) eil51; (b) pr107; (c) bier127; (d) kroa200.

Table 5: Comparison of cellular automata rules.

| Instance | Opt | Rule | Best | Worst | Mean | StD |
|----------|-----|------|------|-------|------|-----|
| att48    | 3352 | Rule 1 | 33523.7085 | 33555.2765 | 33530.0221 | 12.6272 |
|          |      | Rule 2 | 33523.7085 | 33523.7085 | 33523.7085 | 0.0000 |
|          |      | Rule 3 | 33523.7085 | 33555.2765 | 33526.8653 | 9.4704 |
| berlin52 | 7452 | Rule 1 | 7544.3659  | 7544.3659  | 7544.3659  | 0.0000 |
|          |      | Rule 2 | 7544.3659  | 7544.3659  | 7544.3659  | 0.0000 |
|          |      | Rule 3 | 7544.3659  | 7544.3659  | 7544.3659  | 0.0000 |
| bier127  | 118282 | Rule 1 | 118822.0753 | 118822.0753 | 118822.0753 | 0.0000 |
|          |      | Rule 2 | 118293.5238 | 118293.5238 | 118293.5238 | 0.0000 |
|          |      | Rule 3 | 118293.5238 | 118293.5238 | 118293.5238 | 0.0000 |
| ch130    | 15377 | Rule 1 | 6225.4433  | 6225.4433  | 6225.4433  | 0.0000 |
|          |      | Rule 2 | 6183.0251  | 6183.0251  | 6183.0251  | 0.0000 |
|          |      | Rule 3 | 6183.0251  | 6205.6532  | 6189.9083  | 10.3112 |
| chn31    | 15377 | Rule 1 | 15377.7113 | 15377.7113 | 15377.7113 | 0.0000 |
|          |      | Rule 2 | 15377.7113 | 15377.7113 | 15377.7113 | 0.0000 |
|          |      | Rule 3 | 15377.7113 | 15377.7113 | 15377.7113 | 0.0000 |
| eil51    | 426  | Rule 1 | 430.4455   | 436.7443   | 433.2549   | 1.9062  |
|          |      | Rule 2 | 428.8718   | 432.1911   | 429.9723   | 1.1064  |
|          |      | Rule 3 | 430.4455   | 434.1932   | 432.0974   | 1.1403  |
It can be seen that, in most instances, a better and more stable optimal solution can be obtained by using rule 2 in SCGA. In other words, the optimization performance of cellular automata rules with a moderate total number of living cells is giving better results than other rules.

4.3. Comparison of Other Algorithms. In order to verify the optimization of SCGA, SCGA, CGA, GA, and SA are applied to the instances. The experimental results are shown in Tables 6–8.

As shown in Table 6, based on the instances of kroa200, bier127, pr136, and ch130, the distance of the SCGA best solutions is shorter than the distance of the GA best solutions by 9.73%, 8.64%, 8.25%, and 5.80%, respectively. Besides, based on the instances of kroa200, pr136, bier127, eil76, and ch130, the distance of the SCGA best solutions is shorter than the distance of the SA best solutions by 15.79%, 8.86%, 6.69%, 8.25%, 5.14%, and 5.02%, respectively. In the instances of kroa200 and pr136, the distance of the best solutions obtained by using SCGA is shorter than the distance of the best solutions obtained by using CGA by 7.42% and 5.43%, respectively. The SCGA best solutions are better than the GA best solutions in these 13 instances. Only in the instances of oliver30 and chn31, the distance of the SCGA best solutions is equal to the CGA best solutions’ distance.

Moreover, in the instance of oliver30, the SCGA best solution’s distance is equal to the distance of the SA best solution. Therefore, the SCGA worst solutions are better than the GA, SA, and CGA worst solutions in these 13 instances.

It is shown in Table 7, based on the instances of kroa200, pr136, bier127, kroa100, ch130, berlin52, ch130, and pr107, that the SCGA solution mean distance is shorter than the GA mean distance solutions by 26.30%, 12.16%, 11.07%, 10.15%, 8.98%, 8.90%, and 8.71%, respectively. In the instances of kroa200, pr136, pr152, bier127, ch130, and kroa100, the SCGA mean distance is shorter than the SA distance mean value by 17.92%, 10.61%, 8.87%, and 8.03%, respectively. In the instances of kroa200 and pr136, the distance of the SCGA mean value is shorter than the CGA mean distance by 9.60% and 6.88%, respectively. At the same time, the standard deviations of the SCGA are the smallest.

This paper uses SCGA to solve these instances and accounts for four decimal places. So, there will be a slight error when comparing with the known optimal solution. In Table 8, the calculation equations for the best solution error and the mean value error are as follows:

$$\text{Best. Err} = \frac{\text{Best} - \text{Opt}}{\text{Opt}} \quad (7)$$

$$\text{Mean. Err} = \frac{\text{Mean} - \text{Opt}}{\text{Opt}}$$

It is shown in Table 8, in the instances of chn31, pr107, and pr152, that the SCGA best solution errors are about 0%. Only in the instances ch130 and eil76, the SCGA best solution errors are greater than 1%.

From Tables 6–8, we conclude the following.

The optimal solution, the worst solution, the mean value, and the standard deviation obtained by solving the TSP instances using our proposed algorithm (SCGA) are better than GA, SA, and CGA. Using SCGA to solve the TSP instances gives the closest values to the theoretical optimal value. Figure 6 shows the box plots of 10 independents runs of instances kroa100, pr152, att48, and eil76. The box plots of SCGA always locate the lowest position in four instances. Furthermore, these box plots of SCGA are close to the straight lines, which indicate that SCGA has the smallest
Table 6: The best solution and the worst solution.

| Instance | Opt | Metric | GA   | SA   | CGA  | SCGA  |
|----------|-----|--------|------|------|------|-------|
| att48    | 33522 | Best | 34149.3743 | 34013.2819 | 33961.1106 | 33523.7085 |
|          |       | Worst | 36386.8892 | 36208.3069 | 34609.5047 | 33555.2765 |
| berlin52 | 7542 | Best | 7899.0083  | 7684.5829  | 7812.5665  | 7544.3659  |
|          |       | Worst | 8538.6311  | 8240.4600  | 8522.4864  | 7544.3659  |
| bier127  | 118282 | Best | 128515.2927 | 126213.835 | 123577.8803 | 118293.5238 |
|          |       | Worst | 135685.9343 | 132724.0594 | 127274.5241 | 118822.0753 |
| ch130    | 6110 | Best | 6541.9890   | 6493.4978   | 6350.0191  | 6183.0251  |
|          |       | Worst | 6997.0840   | 6901.1848   | 6559.8327  | 6225.4433  |
| chn31    | 15377 | Best | 15487.2062  | 15450.5079  | 15377.7113 | 15377.7113 |
|          |       | Worst | 16265.9926  | 16097.4187  | 15891.3431 | 15377.7113 |
| eil51    | 426  | Best | 437.6323    | 439.5980    | 438.5469   | 428.8718   |
|          |       | Worst | 464.2338    | 458.5167    | 459.2966   | 436.7443   |
| eil76    | 538  | Best | 570.9642    | 575.2977    | 562.0046   | 547.1706   |
|          |       | Worst | 608.9891    | 619.4997    | 584.2479   | 552.0326   |
| kroa100  | 21282 | Best | 41893.2866  | 36233.3604  | 33755.3014 | 29865.9464 |
|          |       | Worst | 24611.4693  | 24663.0452  | 23080.9158 | 21767.5565 |
| kroa200  | 29368 | Best | 32405.9486  | 34196.6537  | 31724.8527 | 29533.0620 |
|          |       | Worst | 41078.4393  | 45161.7422  | 44579.1881 | 44301.6837 |
| pr107    | 44303 | Best | 45983.4393  | 45161.7422  | 44579.1881 | 44301.6837 |
|          |       | Worst | 50006.8782  | 49392.8367  | 47261.2359 | 44301.6837 |
| pr136    | 96772 | Best | 104784.6839 | 105372.1700 | 102047.9935 | 96795.4030 |
|          |       | Worst | 114883.2612 | 110519.6460 | 106732.0575 | 97181.1904 |
| pr152    | 73682 | Best | 75801.2620  | 76992.7749  | 74888.9388 | 73683.6406 |
|          |       | Worst | 81017.3352  | 83021.7641  | 78170.3976 | 73683.6406 |

Table 7: The mean value and standard deviation.

| Instance | Opt | Metric | GA   | SA   | CGA  | SCGA  |
|----------|-----|--------|------|------|------|-------|
| att48    | 33522 | Mean  | 35198.2428 | 34873.5741 | 34341.4737 | 33526.8653 |
|          |       | StD | 775.3089  | 742.4642  | 218.9470  | 775.3089  |
| berlin52 | 7542 | Mean  | 8221.6318  | 7964.2032  | 7967.7532  | 7544.3659  |
|          |       | StD | 175.0246  | 166.5235  | 108.1287  | 108.1287  |
| bier127  | 118282 | Mean | 131581.2788 | 128983.7461 | 125262.8867 | 118469.7076 |
|          |       | StD | 2448.7082  | 2325.0307  | 171.7538  | 171.7538  |
| ch130    | 6110 | Mean  | 6751.4105  | 6749.0506  | 6467.4978  | 6199.5853  |
|          |       | StD | 147.3786  | 125.4857  | 70.3231  | 70.3231  |
| chn31    | 15377 | Mean | 454.4414   | 446.8963   | 443.9726   | 431.7748   |
|          |       | StD | 9.1465    | 6.2678     | 5.7699    | 1.9750    |
| eil51    | 426  | Mean  | 590.0867   | 589.7360   | 570.9942   | 549.4681   |
|          |       | StD | 13.1262   | 11.3760    | 6.3239    | 1.6914    |
| eil76    | 538  | Mean  | 23584.7465 | 23095.5248 | 22203.6352 | 21379.2726 |
|          |       | StD | 674.8707   | 803.5809   | 451.3302  | 127.6113   |
| kroa100  | 21282 | Mean | 37475.0045 | 34987.5192 | 32519.8526 | 29671.5496 |
|          |       | StD | 2563.0760  | 589.8787   | 536.6872  | 120.7940   |
| kroa200  | 29368 | Mean | 443.1219   | 424.0513   | 425.4695   | 423.7406   |
|          |       | StD | 19.0721   | 0.5317     | 1.4590    | 0.0000    |
| oliver30 | 420  | Mean  | 48158.4584 | 47656.8289 | 45861.5128 | 44301.6837 |
|          |       | StD | 1505.6024  | 1540.8192  | 833.1544  | 0.0000    |
| pr107    | 44303 | Mean | 108845.5545 | 107333.6988 | 103717.2423 | 97042.3108 |
|          |       | StD | 2690.4693  | 1450.6517  | 1462.0469 | 151.2063   |
| pr136    | 96772 | Mean | 78641.4797 | 80403.5388 | 76394.6319 | 73683.6406 |
|          |       | StD | 1537.2679  | 1712.5745  | 907.1630  | 0.0000    |
Table 8: The best solution error and the mean value error.

| Instance | Metric | GA (%) | SA (%) | CGA (%) | SCGA (%) |
|----------|--------|--------|--------|---------|----------|
| att48    | Best.Err | 1.87   | 8.01   | 1.31    | 0.01     |
|          | Mean.Err  | 5.00   | 4.03   | 2.44    | 0.01     |
| berlin52 | Best.Err  | 4.73   | 9.26   | 3.59    | 0.03     |
|          | Mean.Err  | 9.01   | 5.60   | 5.65    | 0.03     |
| bier127  | Best.Err  | 8.65   | 12.21  | 4.48    | 0.01     |
|          | Mean.Err  | 11.24  | 9.05   | 5.90    | 0.16     |
| ch130    | Best.Err  | 7.07   | 12.95  | 3.93    | 1.20     |
|          | Mean.Err  | 10.50  | 10.46  | 5.85    | 1.47     |
| chn31    | Best.Err  | 0.72   | 4.91   | 1.24    | 0.00     |
|          | Mean.Err  | 2.63   | 2.09   | 1.24    | 0.00     |
| eil51    | Best.Err  | 2.73   | 7.63   | 2.95    | 0.67     |
|          | Mean.Err  | 6.68   | 4.91   | 4.22    | 1.36     |
| eil76    | Best.Err  | 6.13   | 15.15  | 4.46    | 1.70     |
|          | Mean.Err  | 9.68   | 9.62   | 6.13    | 2.13     |
| kroa100  | Best.Err  | 3.30   | 15.89  | 1.23    | 0.02     |
|          | Mean.Err  | 10.65  | 8.52   | 4.33    | 0.46     |
| kroa200  | Best.Err  | 10.34  | 23.38  | 8.03    | 0.56     |
|          | Mean.Err  | 27.60  | 19.13  | 10.73   | 1.03     |
| oliver30 | Best.Err  | 1.12   | 1.22   | 0.89    | 0.89     |
|          | Mean.Err  | 5.51   | 0.96   | 1.30    | 0.89     |
| pr107    | Best.Err  | 3.79   | 11.49  | 0.62    | 0.00     |
|          | Mean.Err  | 8.70   | 7.57   | 3.52    | 0.00     |
| pr136    | Best.Err  | 8.28   | 14.21  | 5.45    | 0.02     |
|          | Mean.Err  | 12.48  | 10.91  | 7.18    | 0.28     |
| pr152    | Best.Err  | 2.88   | 12.68  | 1.64    | 0.00     |
|          | Mean.Err  | 6.73   | 9.12   | 3.68    | 0.00     |

Figure 6: Continued.
5. Conclusions

In order to solve the TSP effectively, a hybrid Cellular Genetic Algorithm with SA (SCGA) is proposed. The algorithm combines the principles of GA, CA, and SA to improve the optimization GA’s performance. Experimental results show that SCGA is superior to GA, SA, and CGA in solving the TSP. It proves that SCGA has high optimization performance and robustness when solving the TSP.

We studied the optimization performance of three cellular automata rules for the TSP solution. In addition to the three cellular automata rules studied in this paper, other cellular automata rules may have higher optimization performance. We combined the simulated annealing algorithm and the cellular genetic algorithm to enhance the TSP solution’s optimization performance. In addition to the simulated annealing algorithm, the cellular genetic algorithm was combined with other algorithms and effectively enhanced the TSP solution’s optimization performance. Common traveling salesman problems include symmetric traveling salesman problem, asymmetric traveling salesman problem, multiperson traveling salesman problem, and multitarget traveling salesman problem. This paper studied only the symmetric traveling salesman problem based on the Euclidean distance. The optimization performance of the asymmetric traveling salesman problem and other traveling salesman problems has not been verified. In addition, it is not clear how to apply the hybrid cellular genetic algorithm to other optimal combination problems, such as knapsack problems, production scheduling problems, and clustering. These issues are worthy of our further discussion, and they will be the focus of our future work.

Data Availability

Datasets used to support the findings of this study are available from http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (Grant no. 61772006), the Special Fund for Bagui Scholars of Guangxi (Grant no. 2017), the Science and Technology Major Project of Guangxi (Grant no. AA17204096), the General Program of Guangxi Natural Science Foundation (Grant no. 2019GXNSFAA185033), and the Talent Introduction Project of Guangxi University for Nationalities (Grant no. 2020KJQD05). And, the authors would like to express their gratitude to EditSprings (https://www.editsprings.com/) for the expert linguistic services provided.

References

[1] G. Gutin and A. P. Punnen, The Traveling Salesman Problem and its Variations, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
[2] J. E. Beasley and N. Christofides, “An algorithm for the resource constrained shortest path problem,” Networks, vol. 19, no. 4, pp. 379–394, 1989.
[3] R. D. Franceschi, M. Fischetti, and P. Toth, “A new ILP-based refinement heuristic for vehicle routing problems,” Mathematical Programming, vol. 105, no. 2-3, pp. 471-499, 2006.
[4] M. Widmer and A. Hertz, “A new heuristic method for the flow shop sequencing problem,” European Journal of Operational Research, vol. 41, no. 2, pp. 186–193, 1989.
[5] K. H. Nguyen and C. Y. Ock, “Word sense disambiguation as a traveling salesman problem,” Artificial Intelligence Review, vol. 40, no. 64, pp. 405–427, 2013.

[6] C. Papalitsas, P. Karakostas, T. Andronikos et al., “Combinatorial GVNS (general variable neighborhood search) optimization for dynamic garbage collection,” Algorithms, vol. 11, no. 38, 2018.

[7] C. H. Papadimitriou, “The euclidean travelling salesman problem is NP-complete,” Theoretical Computer Science, vol. 4, no. 3, pp. 237–244, 1977.

[8] G. Laporte, “The traveling salesman problem: an overview of exact and approximate algorithms,” European Journal of Operational Research, vol. 59, no. 2, pp. 231–247, 1992.

[9] Y. J. Song, D. Q. Wu, W. Deng et al., “Multi-population parallel co-evolutionary differential evolution for parameter optimization,” Energy Conversion and Management, vol. 228, p. 2021.

[10] Y. J. Song, D. Q. Wu, A. W. Mohamed et al., “Enhanced success history adaptive DE for parameter optimization of photovoltaic models,” Complexity, vol. 2021, Article ID 6660115, 14 pages, 2021.

[11] W. Deng, J. J. Xu, X. Z. Gao et al., “An enhanced MSIQDE algorithm with novel multiple strategies for global optimization problems,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020.

[12] W. Deng, J. Xu, Y. Song, and H. Zhao, “Differential evolution algorithm with wavelet basis function and optimal mutation strategy for complex optimization problem,” Applied Soft Computing, vol. 100, Article ID 106724, 2020.

[13] W. Deng, J. Xu, H. Zhao, and Y. Song, “A novel gate resource allocation method using improved PSO-based QEA,” IEEE Transactions on Intelligent Transportation Systems, p. 1, 2020.

[14] W. Deng, J. Xu, Y. Song, and H. Zhao, “An effective improved co-evolution ant colony optimisation algorithm with multi-strategies and its application,” International Journal of Bio-Inspired Computation, vol. 16, no. 3, pp. 158–170, 2020.

[15] L. Babel, “New heuristic algorithms for the dubins traveling salesman problem,” Journal of Heuristics, vol. 26, no. 4, pp. 503–530, 2020.

[16] K. Yang, X. You, S. Liu, and H. Pan, “A novel ant colony optimization based on game for traveling salesman problem,” Applied Intelligence, vol. 50, no. 12, pp. 4529–4542, 2020.

[17] Y. Dong and Z. Huang, “An improved noise quantum annealing method for TSP,” International Journal of Theoretical Physics, vol. 50, no. 12, 2020.

[18] J. W. Ohlmann and B. W. Thomas, “A compressed-annealing heuristic for the traveling salesman problem with time windows,” Inform Journal on Computing, vol. 19, no. 1, pp. 80–90, 2017.

[19] Y. Zhou, Q. Luo, H. Chen, A. He, and J. Wu, “A discrete invasive weed optimization algorithm for solving traveling salesman problem,” Neurocomputing, vol. 151, pp. 1227–1236, 2015.

[20] M. Mavrovouniotis, F. M. Muller, and S. Yang, “Ant colony optimization with local search for dynamic traveling salesman problems,” IEEE Transactions on Cybernetics, vol. 47, no. 7, pp. 1743–1756, 2017.

[21] G. Pan, K. Li, A. Ouyang, and K. Li, “Hybrid immune algorithm based on greedy algorithm and delete-cross operator for solving TSP,” Soft Computing, vol. 20, no. 2, pp. 555–566, 2016.

[22] J. H. Holland and H. John, “Genetic algorithms and the optimal allocation of trials,” Siam Journal on Computing, vol. 2, no. 2, pp. 88–105, 1973.

[23] L. Yao, W. A. Sethares, and D. C. Kammer, “Sensor placement for on-orbit modal identification via a genetic algorithm,” AIAA Journal, vol. 31, no. 10, pp. 1922–1928, 2012.

[24] K. Shinichi, T. Daisuke, A. Masanori et al., “Dynamic modeling of genetic networks using genetic algorithm and s-system,” Bioinformatics, vol. 5, pp. 643–650, 2003.

[25] R. Martínez, O. Castillo, and L. T. Aguilar, “Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mobile robot using genetic algorithms,” Information Sciences, vol. 179, no. 13, pp. 2158–2174, 2009.

[26] P. Ahmadi and I. Dincer, “Exergoenvironmental analysis and optimization of a cogeneration plant system using multimodal genetic algorithm (mga),” Energy, vol. 35, no. 12, pp. 5161–5172, 2010.

[27] J.-Y. Potvin, “Genetic algorithms for the traveling salesman problem,” Annals of Operations Research, vol. 63, no. 3, pp. 337–370, 1996.

[28] H. H. Liu, C. Cui, and J. Chen, “An improved genetic algorithm for solving travel salesman problem,” Transactions of Beijing Institute of Technology, vol. 33, no. 4, pp. 390–393, 2013.

[29] Y. Y. Yu, Y. Chen, and T. Y. Li, “Improved genetic algorithm for solving TSP,” Control and Decision, vol. 29, no. 8, pp. 1483–1488, 2014.

[30] Y. N. Wang and H. W. Ge, “Improved simulated annealing genetic algorithm for solving TSP problem,” Computer Engineering and Applications, vol. 46, no. 5, pp. 44–47, 2010.

[31] L. Y. Zhang, Y. Gao, and T. Fei, “Firefly genetic algorithm for traveling salesman problem,” Computer Engineering and Design, vol. 40, no. 7, pp. 1939–1944, 2019.

[32] L. H. Tao, Z. N. Ma, P. T. Shi et al., “Dynamic ant colony genetic algorithm based on TSP,” Machinery Design & Manufacture, vol. 12, pp. 147–149+154, 2019.

[33] X. Li, J. Wu, and X. Li, “Cellular genetic algorithms,” in Theory of Practical Cellular Automation, Springer, Singapore, 2018.

[34] M. Srinivas and L. M. Patnaik, “Genetic algorithms: a survey,” IEEE Transactions on Systems, Man, and Cybernetics, vol. 3, no. 4, pp. 344–355, 1991.

[35] L. Feng and Y. Qi-wen, “Improved genetic operator for genetic algorithm,” Journal of Zhejiang University-SCIENCE A, vol. 3, no. 4, pp. 431–443, 2002.

[36] A. S. Wu and I. Garibay, “The proportional genetic algorithm: gene expression in a genetic algorithm,” Genetic Programming and Evolvable Machines, vol. 3, pp. 157–192, 2002.

[37] P. Karthikeyan, S. Baskar, and A. Alphones, “Improved genetic algorithm using different genetic operator combinations (GOCs) for multicast routing in ad hoc networks,” Soft Computing, vol. 17, no. 9, pp. 1563–1572, 2013.

[38] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” The Journal of Chemical Physics, vol. 21, no. 6, pp. 1087–1092, 1953.

[39] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, no. 4596, pp. 671–680, 1983.

[40] N. Leite, F. Melício, and A. C. Rosa, “A fast simulated annealing algorithm for the examination timetabling problem,” Expert Systems with Applications, vol. 122, pp. 137–151, 2019.

[41] W.-H. Chen and B. Srivastava, “Simulated annealing procedures for forming machine cells in group technology,” European Journal of Operational Research, vol. 75, no. 1, pp. 100–111, 1994.

[42] E. Aarts and J. Korst, “Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial problems,””
optimization and neural computing,” *Siam Review*, vol. 12, no. 2, p. 323, 2006.

[43] P. Laarhoven, J. M. Van, and A. J. K. Lenstra, “Job shop scheduling by simulated annealing,” *Operations Research*, vol. 40, pp. 113–125, 1992.

[44] V. Neumann, *Theory of Self-Reproduction Automata*, University of Illinois Press, Urbana, IL, USA, 1966.

[45] S. Wolfram, “Statistical mechanics of cellular automata,” *Reviews of Modern Physics*, vol. 55, no. 3, pp. 601–644, 1983.

[46] S. Wolfram, “Computation theory of cellular automata,” *Communications in Mathematical Physics*, vol. 96, no. 1, pp. 15–57, 1984.

[47] E. Jen, “Global properties of cellular automata,” *J Stat Phys*, vol. 43, p. 1986.

[48] B. Chopard, “Cellular automata modeling of physical systems,” in *Encyclopedia of Complexity and Systems Science*, R. Meyers, Ed., Springer, New York, NY, USA, 2009.

[49] S. W. Wei and Q. H. Hu, “An improved cellular genetic algorithm with evolutionary rules,” *Computer Integrated Manufacturing Systems*, vol. 36, no. 8, pp. 247–252, 2019.

[50] Y. M. Lu, M. Li, and L. Li, “The cellular genetic algorithm with evolutionary rule,” *Acta Electronica Sinica*, vol. 38, no. 7, pp. 1603–1607, 2010.