A Novel Evolutionary Algorithm with Column and Sub-Block Local Search for Sudoku Puzzles

Chuan Wang, Bing Sun, *Student Member*, *IEEE*, Ke-Jing Du, Jian-Yu Li, *Member*, *IEEE*, Zhi-Hui Zhan, *Senior Member*, *IEEE*, Sang-Woon Jeon, *Member*, *IEEE*, Hua Wang, *Senior Member*, *IEEE*, and Jun Zhang, *Fellow*, *IEEE*

***Abstract*— Sudoku puzzles are not only popular intellectual games but also NP-hard combinatorial problems related to various real-world applications, which have attracted much attention worldwide. Although many efficient tools, such as evolutionary computation (EC) algorithms, have been proposed for solving Sudoku puzzles, they still face great challenges with regard to hard and large instances of Sudoku puzzles. Therefore, to efficiently solve Sudoku puzzles, this paper proposes a genetic algorithm (GA)-based method with a novel local search technology called local search-based GA (LSGA). The LSGA includes three novel design aspects. First, it adopts a matrix coding scheme to represent individuals and designs the corresponding crossover and mutation operations. Second, a novel local search strategy based on column search and sub-block search is proposed to increase the convergence speed of the GA. Third, an elite population learning mechanism is proposed to let the population evolve by learning the historical optimal solution. Based on the above technologies, LSGA can greatly improve the search ability for solving complex Sudoku puzzles. LSGA is compared with some state-of-the-art algorithms at Sudoku puzzles of different difficulty levels and the results show that LSGA performs well in terms of both convergence speed and success rates on the tested Sudoku puzzle instances.**

***Index Terms*—Sudoku puzzle; genetic algorithm; combinatorial optimization problems; evolutionary computation; local search**

Manuscript received ….; revised ….; accepted ….. This work was supported in part by the National Natural Science Foundations of China under Grant 62176094, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, in part by the National Research Foundation of Korea under Grant NRF-2022H1D3A2A01093478, and in part by the Korea Institute of Ocean Science and Technology under Grant PE99732. (*Corresponding authors: Zhi-Hui Zhan and Jun Zhang*)

C. Wang is with the College of Software, Henan Normal University, Xinxiang 453007, China.

B. Sun is with the College of Computer and Information, Henan Normal University, Xinxiang 453007, China.

J. Y. Li and Z.-H. Zhan are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, the Pazhou Laboratory, Guangzhou 510330, China, and the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

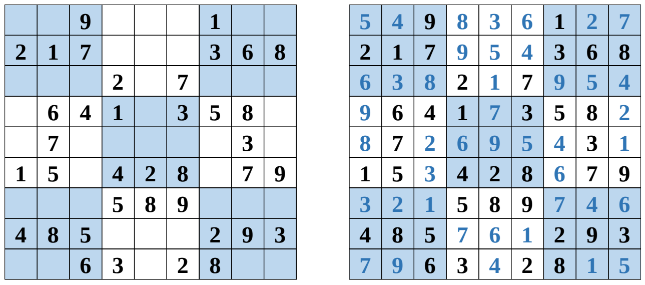
K. J. Du and H. Wang are with the Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, VIC 8001, Australia.

S.-W. Jeon is with the Department of Electronics and Communication Engineering, Hanyang University, Ansan 15588, South Korea.

J. Zhang is with the Zhejiang Normal University, Jinhua 321004, China and with the Hanyang University, Ansan 15588, South Korea.

I. Introduction

SUDOKU is a popular logic-based combinatorial puzzle game for people of all ages, it was invented in 1979 and was officially named “Sudoku” in 1984 [1]. The typical Sudoku is composed of 81 cells (a 9×9 grid), as shown in Fig. 1. Fig. 1(a) is a Sudoku puzzle with several given numbers, and Fig. 1(b) is the solution to this puzzle. Moreover, with rapid development, more complex high-dimensional Sudoku puzzles have appeared in recent years, with dimensions of 16×16, 25×25, and even 100×100.



		9				1		
2	1	7				3	6	8
			2		7			
	6	4	1		3	5	8	
	7						3	
1	5		4	2	8		7	9
			5	8	9			
4	8	5				2	9	3
		6	3		2	8		

(a) Sudoku puzzle

5	4	9	8	3	6	1	2	7
2	1	7	9	5	4	3	6	8
6	3	8	2	1	7	9	5	4
9	6	4	1	7	3	5	8	2
8	7	2	6	9	5	4	3	1
1	5	3	4	2	8	6	7	9
3	2	1	5	8	9	7	4	6
4	8	5	7	6	1	2	9	3
7	9	6	3	4	2	8	1	5

(b) Solution to the puzzle

Fig. 1. Example of 9×9 Sudoku puzzle and its solution.

The rules of Sudoku are as follows: The game begins with several given numbers in an $N \times N$ grid. Then, the player must fill in the empty cells with numbers 1 to N in such a way that no number appears twice in the same row, column, or sub-blocks. Sudoku puzzles are simple in form and definition, but it is not easy to find solutions [2]. In 2003, Takayuki Yato and Takahiro Seta proved that solving Sudoku is an NP-hard problem [3]. Generally, the factors for evaluating the difficulty of a Sudoku puzzle include the dimension of the problem, percentage and distribution of the given numbers, and time cost to solve the Sudoku by a baseline solver. To evaluate the difficulty of Sudoku, some typical tools have been developed, including SUDOKUSAT, Sudoku Explainer (SE), and Hoduku Explainer [4]. Currently, the most used tool is SE, which can give a corresponding SE score. Generally, a higher SE score indicates that the Sudoku puzzle is more difficult. For example, the Sudoku puzzles can be divided into levels of easy, medium, hard, evil, and even more difficult.

Nowadays, Sudoku is not only a game but also a kind of core problem in many real-world applications in daily life and industrial engineering, such as in data encryption [5], radar waveform design [6], and education [7]. For example, Jana *et al.* [5] proposed a video steganography technique that hid encrypted data in videos by a Sudoku-based reference matrix. This technique shows good performance in resisting fault attacks if the Sudoku puzzle can be solved efficiently. Li *et al.* [8] applied retracing extended Sudoku to image data-hiding technology. As the retracing extended Sudoku is a kind of Sudoku containing multiple solutions, which imposes high demands on the robustness of the algorithm for solving the Sudoku puzzle. To improve the efficiency of photovoltaic systems, Horoufiany et al. [9] proposed a Sudoku-based arrangement rule to avoid mutual shading between fixed photovoltaic arrays and obtained the optimal arrangement by solving the corresponding Sudoku puzzle by a genetic algorithm (GA). Moreover, the Sudoku is also studied as a representative of the exact cover problem [10]. Therefore, the Sudoku puzzle widely exists in various applications. The development of algorithms for solving Sudoku has not only of academic research significance, but also helpful for real-world applications, which has attracted increasing attention.

So far, the existing algorithms for solving Sudoku puzzles can be divided into mathematical algorithms [11] and heuristics algorithms [12]. The exact algorithms are faster at solving Sudoku puzzles, but lack portability [13]. Therefore, as a type of heuristics algorithm, GA has gained widespread attention due to its powerful search ability and versatility. During the past decade, some studies have shown the great potential of GA in solving Sudoku puzzles [14]. However, the GA-based methods still have some shortcomings. When solving difficult Sudoku puzzles, some GA-based methods still need to take a long time to solve or even be unsolvable [15]. Therefore, developing a more efficient method to solve Sudoku puzzles remains a challenge.

In this paper, we propose an improved GA with local search (LSGA) to effectively and efficiently solve Sudoku puzzles. Specifically, the LSGA has three novel designs. First, we adopt the matrix-based encoding for Sudoku, and based on this encoding scheme, the crossover and mutation operations in LSGA are designed. Second, we present a novel local search mechanism based on column search and sub-block search to increase the convergence speed of the GA. Third, to avoid being trapped in local optimal solutions, an elite population learning mechanism is proposed to randomly replace poor individuals with new individuals, which is very effective when solving difficult Sudoku puzzles. To illustrate the efficiency of the proposed LSGA, we evaluate it on Sudoku puzzles at different difficulty levels and compare it with some state-of-the-art approaches.

The rest of the paper is organized as follows: Section II reviews studies on solving Sudoku puzzles in recent years. Then, in Section III, the matrix-based GA is elaborated, and the effectiveness and efficiency of the proposed LSGA are illustrated by extensive experiments in Section IV. Finally, conclusions are given in Section V.

II. Related Work

The charm of Sudoku is that it is easy to learn but difficult to master. Therefore, it has received much attention since it was first published in the newspaper "*Times*" [16]. Fig. 2 shows the structure of a standard Sudoku puzzle. To summarize the definition, the $N\times N$ ($\sqrt{N}$ is an integer greater than 0) Sudoku puzzle must satisfy the following constraints:

(1) Unique solution restriction: A Sudoku puzzle has only one unique solution.

(2) The rule of rows: All 1 to N numbers in each row should appear and not be repeated.

(3) The rule of columns: All 1 to N numbers in each column should appear and not be repeated.

(4) The rule of sub-blocks: All 1 to N numbers in each $\sqrt{N}\times\sqrt{N}$ sub-block should appear and should not be repeated.

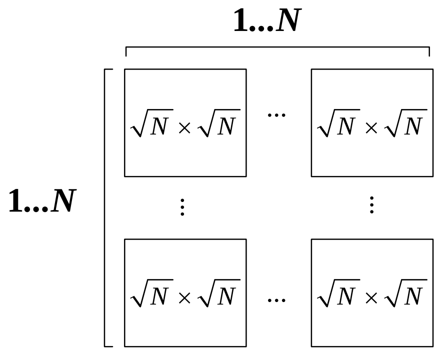


Fig. 2. Structure of standard $N\times N$ Sudoku puzzle.

Many researchers have tried tackling Sudoku through different methods. A widely used method is dancing links [17], which is a brute force algorithm. This method transforms Sudoku puzzles into exact cover problems and employs the backtracking method to solve them. However, brute force algorithms cannot handle those high-dimensional Sudoku puzzles at an acceptable time and memory cost [18].

To overcome the shortage of brute force algorithms, many heuristic approaches have been reported in the literature to solve Sudoku. For example, Sevkli *et al*. [19] proposed two novel models based on the variable neighborhood search (VNS) algorithm to solve Sudoku: Unfiltered-VNS and Filtered-VNS. The experiments showed that Filtered-VNS can obtain better solution quality than Unfiltered-VNS for easy- and medium-level puzzles, while Unfiltered-VNS performs better in solving hard-level puzzles. Betar *et al*. [20] introduced an improved hill-climbing algorithm called the β-Hill-Climbing algorithm, which could escape local optima by using a random operator. Experimental results showed that the β-Hill-Climbing algorithm can find solutions within a very short time under the best parameter configuration.

Traditional Sudoku solution methods are ineffective in solving complex and high-dimension Sudoku puzzles because the Sudoku puzzles have a huge search space [21]. Evolutionary computation algorithms, such as GA [22]-[24], ant colony optimization (ACO) [25]-[27], particle swarm optimization (PSO) [28]-[32], differential evolution [34]-[36], and estimation of distribution algorithms [37] have shown promising performance in solving Sudoku puzzles and many other complex or real-world problems [38]-[41]. For example, Mantere and Koljonen [15] adopted GA to solve the Sudoku puzzle, but this work could not effectively solve difficult Sudoku puzzles. Pathak *et al*. [42] proposed the wisdom of a crowd aggregate function for Sudoku puzzles, which can effectively prevent GA from being trapped in local optima. Deng *et al*. [43] proposed an efficient hybrid algorithm for Sudoku. In this hybrid algorithm, the improved GA can produce more abundant individuals to participate in crossover. Then, they combined PSO with GA, so that the population could better evolve towards the optimal solution. Lloyd *et al*. [16] adopted ACO to solve high-dimensional Sudoku puzzles and compared the effect of the percentage of given numbers on the difficulty of Sudoku.

Summarizing the above algorithms, we can conclude that the search ability and convergence speed of the algorithms are key indicators for solving Sudoku puzzles, because of the huge search space and unique solutions of Sudoku puzzles. In this paper, we present an improved GA, called LSGA, which adopts the new local search method designed for Sudoku puzzles and a new elite population learning mechanism to solve Sudoku puzzles more effectively and efficiently.

III. Proposed LSGA Method

A. Representations and Initialization

When solving Sudoku puzzles, it is necessary to encode the possible solutions into data structures, which facilitate the evolutionary operations of the GA. For example, Katya *et al*. [44] recorded the given numbers and solutions of Sudoku with two strings of length N^2, and a string of length N to record the numbers waiting to be selected in each row. Mantere *et al*. [15] used two arrays of N^2 numbers to represent the Sudoku solution: one represented the solution, and the other recorded the position of the given numbers.

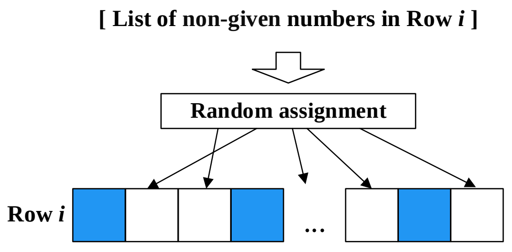


Fig. 3. Initialization stage: non-given numbers in row *i* are randomly assigned to empty spaces.

In our algorithm, we adopt two matrices to represent a chromosome, one matrix is the major matrix that records the numbers in each position of the Sudoku grids, while another matrix is the associated matrix that records where each position is occupied by a number. Specifically, the number in row i and column j in Sudoku is recorded at position (i, j) of the major matrix. Meanwhile, for the associated matrix, if there is a given number at a position, then the corresponding value of the associated matrix is "1"; otherwise, it is "0." This coding method facilitates the implementation of crossover operation and local search operation.

The initialization stage is an important process of the GA. To reduce the complexity of the puzzle, all non-given numbers in each row are randomly assigned to the empty spaces, as shown in Fig. 3. Therefore, the initial solutions will satisfy the row rules of Sudoku.

B. Fitness Function

The goal of solving a Sudoku puzzle is to find the solution where each number occurs only once on each row, column, and sub-block. Therefore, when evaluating an individual, we count how many rows, columns, and sub-blocks are incorrect (i.e., not including all the numbers from 1 to N). Therefore, the fitness of the optimal solution is 0, which means that the rows, columns, and sub-blocks of this individual satisfy all rules of Sudoku.

As the individuals generated by the initialization already satisfy the constraints of the row rule, LSGA only needs to optimize the numbers in the columns and sub-blocks without breaking the row rule. In summary, the fitness of each individual can be evaluated by

$$F = \sum_{i=1}^{N} c_i + \sum_{j=1}^{N} s_j \qquad (1)$$

where N is the dimension of the Sudoku puzzle, and c_i indicates whether the i-th column satisfies the rule of Sudoku. That is, c_i equals 0 if the i-th column satisfies the rule and equals 1 if it does not. Correspondingly, s_j represents whether the j-th sub-block satisfies the limitations. F is the sum of r_i and s_j. When F equals 0, it indicates that the algorithm finds the optimal solution.

C. Crossover and Mutation

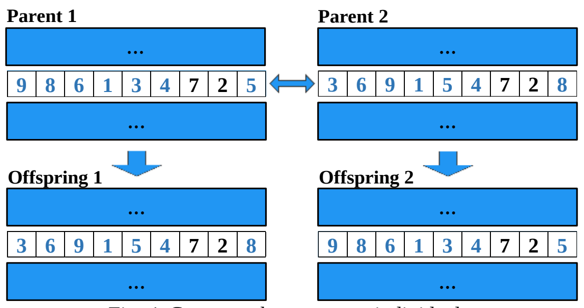


Fig. 4. Crossover between two individuals.

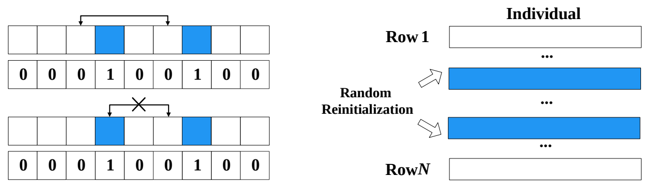
Crossover operations emphasize the exchange of genes among individuals. In LSGA, the crossover operator is performed by rows. Specifically, the parents are selected based on the individual crossover rate $PC1$. Subsequently, based on the row crossover rate $PC2$, the same rows from the parents are selected to participate in the swap operation. An example of the crossover is shown in Fig. 4. In this figure, two parents are selected based on $PC1$. Then, the same rows of the two parents are selected to swap based on $PC2$. Because the given numbers in the same row are in the same position, such as "2" and "7" in Figure 4, the swap operation will not change the original Sudoku puzzle.

The pseudocode of the crossover operation is shown in **Algorithm 1**. In lines 2-3 of **Algorithm 1**, we select two individuals based on the individual crossover rate $PC1$. Subsequently, in lines 4-8, rows are selected to swap based on the row crossover rate $PC2$. Finally, in line 10, the offspring of the crossover are preserved.

Mutation is an important operation for the population to explore the solution space and helps populations escape local optima. Here are two different mutation strategies to help the GA improve its exploration capabilities: swap mutation and reinitialization mutation.

Algorithm 1: Pseudocode of crossover

```
Input: population, individual crossover rate PC1, row crossover rate PC2
 1: For each individual in the population:
 2:     If rand1<PC1: // rand1 is a random variable in [0,1]:
 3:         Select the second parent from the population randomly;
 4:         For each row in the individual:
 5:             If rand2<PC2: // rand2 is a random variable in [0,1]:
 6:                 Parents exchange the selected rows;
 7:             End If
 8:         End For
 9:     End If
10:     Save the offspring to the new population;
11:End For
Output: new population
```



(a) Swap mutation (b) Reinitialization mutation
Fig. 5. Two designed mutation operations: (a) swap mutation; (b) reinitialize mutation.

The swap mutation operation is performed as a swap of two positions inside random rows to ensure that each row satisfies constraint (2) of Sudoku (as mentioned in Section II). The associated matrix is used to check if the position is appropriate for mutation. If the value is “1”, this position is occupied by a given number and this corresponding position is illegal to exchange; thus, the given numbers will not be changed during the mutation. The probability of the swap is determined by the swap mutation rate *PM*1. As shown in Fig. 5(a), the above mutation is legal, while the below mutation is illegal.

Algorithm 2: Pseudocode of mutation

```
Input: population, swap mutation rate PM1 and reinitialization mutation rate PM2
 1: For each individual in the population:
 2:     For each row in the individual:
 3:         If rand1<PM1: // rand1 is a random variable in [0,1]:
 4:             If the number of non-given numbers>=2:
 5:                 Select two non-given numbers to exchange positions;
 6:             End If
 7:         End If
 8:         If rand2<PM2: // rand2 is a random variable in [0,1]:
 9:             Reinitialize the row;
10:         End If
11:     End For
12: End For
Output: new population
```

The reinitialization mutation performs the mutation by reinitializing the distribution of the random rows. As shown in Fig. 5(b), the number of given numbers is retained while the non-given numbers are randomly assigned to the empty space at random. The reinitialization mutation can help the algorithm jump out of the local optima better than the swap mutation. However, a high mutation probability for reinitialization mutation will slow the convergence of the algorithm, so the reinitialization mutation rate *PM*2 is a value smaller than 0.1 and the fitness of individuals is the worst.

The pseudocode of the mutation operator is shown in **Algorithm 2**. In lines 3-7 of **Algorithm 2**, rows are selected to participate in the swap mutation based on the *PM*1. In lines 4-6, here is a judgment on the feasibility of the swap. If there is only one non-given number in a row, this row cannot participate in the swap. In lines 8-10, rows are reinitialized based on *PM*2.

D. Column and Sub-Block Local Search

Many studies have shown that local search is an effective technique for improving the convergence speed of the algorithm [45]. Therefore, we design a new novel local search method in LSGA for solving Sudoku puzzles. It has two components: column local search and sub-block local search.

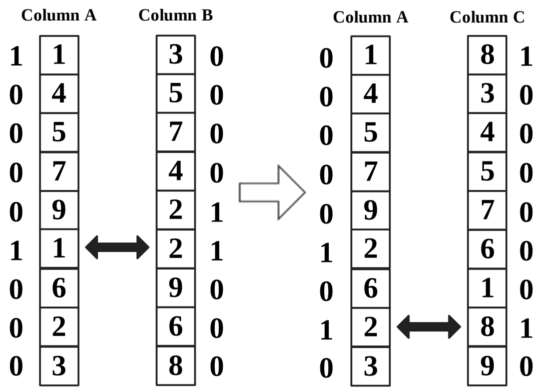


Fig. 6. Example of column local search. Repeat numbers are marked as 1, and others are marked as 0.

The first component is the column local search, which is designed to eliminate the repeating numbers in columns. First, count all columns that do not meet the rules (called illegal columns). We define the set *C* to record these columns. Then, each illegal column is randomly paired with the other columns in *C*, which will be swapped if the repeat numbers are in the same row and none of them are in each other’s column. For example, Fig. 6 depicts a part of the solution to a 9×9 Sudoku puzzle, where we use 1 to mark the position of the repeated number. According to the rules of Sudoku, the number “1” is the repeat number in Column A and “2” is the repeat number in Column B. Therefore, both columns all have repeat numbers in the 6-th row, so we can exchange “1” and “2” to make Column B a legal column. Then, Column A continues to swap with Column C, which could make both of them meet the column rules.

The second component is the sub-block local search. Similar to the column local search, the sub-block local search swaps the repeat number in the same row. First, it counts all sub-blocks that do not meet the rules (called illegal sub-blocks). We define the set *S* to record these sub-blocks. Then, each illegal sub-block is randomly paired with the other sub-blocks in *S*, swapping them if the repeat numbers are in the same row and none of them is in each other's sub-block. For example, in Fig. 7, Sub-block A and Sub-block B both have repeated numbers, one of which is "9" and the other is "8". Therefore, we can exchange them on the same row to make both sub-blocks satisfy the Sudoku rules.

Algorithm 3: Pseudocode of local search

```
Input: population
 1: For each individual in population:
 2:    Record all illegal columns (sub-blocks) in the set $C$ ($S$);
 3:    For each column (sub-block) in $C$ ($S$):
 4:       Randomly select another column (sub-block) from $C$ ($S$);
 5:       If the repeat numbers are in the same row:
 6:          If repeat numbers do not exist in both columns (sub-blocks):
 7:             Swap these repeat numbers;
 8:          End If
 9:       End If
10:    End For
11: End For
Output: new population
```

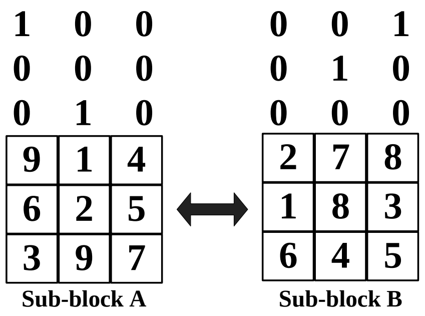


Fig. 7. Example of sub-block local search. Repeat numbers are marked as 1, and others are marked as 0.

In summary, the basic idea of local search is to make the columns and sub-blocks on both sides gradually satisfy the rules of Sudoku by exchanging repeated values. **Algorithm 3** describes the basic framework of the local search.

E. Elite Population Learning

As the local optimal solution and the global optimal solution of Sudoku puzzles are very different, it is difficult for the GA to jump out of the local optima. Thus, a learning mechanism based on elite populations is proposed to avoid the GA falling into local optima. The elite population is a queue structure, that records the best individuals of each generation and updates them with new optimal individuals. In elite population learning, the worst individuals in the population are replaced by a random individual x_{random} from the elite population or are reinitialized. We define the probability P_b to control this process.

The replacement operation is as follows:

$$x_{worst} = \begin{cases} x_{random}, & \text{if } rand() < P_b \\ init(), & \text{otherwise} \end{cases} \tag{2}$$

$$s.t. \quad P_b = \frac{Maxfx - fx(x_{random})}{Maxfx} \tag{3}$$

where x_{worst} is the worst individual, *Maxfx* is the fitness of x_{worst}, x_{random} is a randomly selected elite individual with fitness is $fx(x_{random})$, *rand*() outputs a random variable in (0,1), and *init*() is the initialization function.

According to Eq. (2), the worst individual in each generation has only two choices: to be replaced or to be reinitialized. Therefore, in most cases, the algorithm tends to search toward the current optimal solution via replacement but still explores new search directions via reinitialization. Thus, LSGA can balance exploration and exploitation.

F. Overall LSGA Method

Integrating the above techniques using GA, the developed LSGA is outlined in **Algorithm 4**. In detail, the individuals are generated through initialization in line 1. Then, the population is optimized by evolutionary operations in lines 4-6. Subsequently, in lines 7-8, local search operations are applied to speed up the convergence of the algorithm. Then, the fitness of individuals is evaluated in line 9 and the elite population learning strategy is executed in line 10. The algorithm iteratively repeats the above operations until the optimal solution is found or the maximal number of generations is reached.

Algorithm 4: Pseudocode of LSGA

```
Input: maximum number of generations $FES_{max}$, population.
 1: Initialize population;
 2: Evaluate population;
 3: While ($count \leq FES_{max}$) do:
 4:     Tournament selection;
 5:     Crossover;
 6:     Mutation;
 7:     Column local search;
 8:     Sub-block local search;
 9:     Evaluate population;
10:     Elite population learning;
11:     Reserve the best individual as gbest;
12:     If $fx(gbest)$==0:
13:        Break;
14:     End If
15: End While
16: Obtain the best solution gbest and its fitness $fx(gbest)$;
Output: $fx(gbest)$ and gbest
```

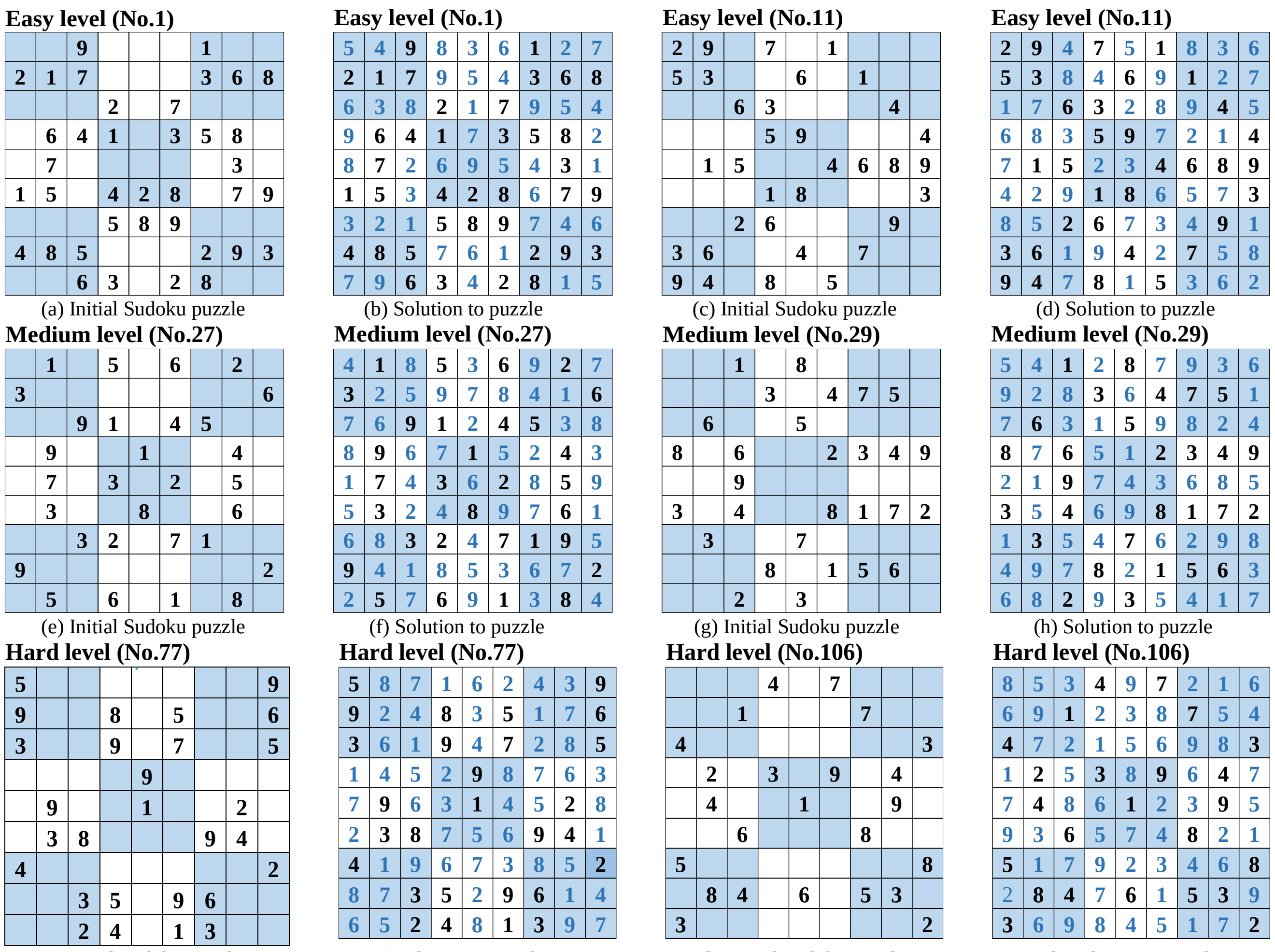
IV. Experimental Studies

A. Comparisons with State-of-the-Art Methods

To illustrate the performance of LSGA, we compare it with state-of-the-art algorithms, including the node-based coincidence algorithm named NB-COIN [13], the preserve building blocks GA named GA-I [46], and the GA with local optima handling named GA-II [18]. To make a fair comparison, the population size is set to 150, while all algorithms run 1×10^4 generations. The parameter settings of LSGA are listed in Table I.

TABLE I. Parameters in LSGA

Parameter	Value
Population size	150
Elite population size	50
Individual crossover rate *PC*1	0.2
Row crossover rate *PC*2	0.1
Swap mutation rate *PM*1	0.3
Reinitialization mutation rate *PM*2	0.05
Tournament size	2



(a) Initial Sudoku puzzle (b) Solution to puzzle (c) Initial Sudoku puzzle (d) Solution to puzzle

(e) Initial Sudoku puzzle (f) Solution to puzzle (g) Initial Sudoku puzzle (h) Solution to puzzle

(i) Initial Sudoku puzzle (j) Solution to puzzle (k) Initial Sudoku puzzle (l) Solution to puzzle

Fig. 8. Six Sudoku puzzles and their solutions.

In the experiments, six classic Sudoku puzzles, which are also solved by the compared algorithms NB-COIN, GA-I, and GA-II, are selected. These Sudoku puzzles cover three difficulty levels (i.e., easy, medium, and hard), as shown in Fig. 8. Each algorithm runs 100 times on each puzzle, where *Succ_Count* is the number of runs among the 100 runs that can find the optimal solutions within 1×10^4 generations, and *Avg_Gen* is the average number of generations required to find the optimal solution. Note that to ensure the fairness of the comparison, the experimental results of the compared algorithms are obtained directly from their original papers. Table II lists the experimental results.

TABLE II. RESULTS OF PROPOSED LSGA AND OTHER METHODS FOR SOLVING SUDOKU ON SIX DIFFERENT SUDOKU PUZZLES

Puzzle ID	LSGA		NB-COIN [13]		GA-I [46]		GA-II [18]	
	Succ_Count	*Avg_Gen*	*Succ_Count*	*Avg_Gen*	*Succ_Count*	*Avg_Gen*	*Succ_Count*	*Avg_Gen*
Easy 1	**100**	**2**	**100**	**2**	**100**	62	**100**	46
Easy 11	**100**	6	**100**	**4**	**100**	137	**100**	88
Medium 27	**100**	**23**	**100**	130	**100**	910	**100**	188
Medium 29	**100**	**57**	**100**	1196	**100**	3193	**100**	357
Hard 77	**100**	**254**	**100**	2710	**100**	9482	**100**	702
Hard 106	**100**	**1269**	**100**	2341	96	26825	**100**	1791

TABLE III. RESULTS OF PROPOSED LSGA AND OTHER METHODS FOR SOLVING SUDOKU ON THREE SUPER DIFFICULT SUDOKU PUZZLES

Puzzle ID	LSGA		GA-III [14]		GPU-GA [47]		GA-I [46]	
	Succ_Count	*Avg_Gen*	*Succ_Count*	*Avg_Gen*	*Succ_Count*	*Avg_Gen*	*Succ_Count*	*Avg_Gen*
SD1	**100**	**424**	**100**	10993	**100**	9072	98	25257
SD2	**100**	**538**	**100**	22036	**100**	13481	90	40365
SD3	**100**	**3926**	96	27384	**100**	22799	62	62283

Super Difficult Sudoku (SD1)

7	9							3
							6	
8		1			4			2
		5						
3			1					
	4				6	2		9
2				3				6
	3		6		5	4	2	1

(a) Initial Sudoku puzzle

Super Difficult Sudoku (SD1)

7	9	2	5	6	8	1	4	3
4	5	3	2	1	9	8	6	7
8	6	1	3	7	4	9	5	2
6	2	5	8	9	3	7	1	4
3	7	9	1	4	2	6	8	5
1	4	8	7	5	6	2	3	9
2	8	4	9	3	1	5	7	6
9	3	7	6	8	5	4	2	1
5	1	6	4	2	7	3	9	8

(b) Solution to puzzle

AI Escargot (SD2)

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		

(c) Initial Sudoku puzzle

AI Escargot (SD2)

1	6	2	8	5	7	4	9	3
5	3	4	1	2	9	6	7	8
7	8	9	6	4	3	5	2	1
4	7	5	3	1	2	9	8	6
9	1	3	5	8	6	7	4	2
6	2	8	7	9	4	1	3	5
3	5	6	4	7	8	2	1	9
2	4	1	9	3	5	8	6	7
8	9	7	2	6	1	3	5	4

(d) Solution to puzzle

Super difficult Sudoku (SD3)

					3		1	7
	1	5			9			8
	6							
1					7			
		9				2		
			5					4
							2	
5			6			3	4	
3	4		2					

(e) Initial Sudoku puzzle

Super difficult Sudoku (SD3)

2	9	4	8	6	3	5	1	7
7	1	5	4	2	9	6	3	8
8	6	3	7	5	1	4	9	2
1	5	2	9	4	7	8	6	3
4	7	9	3	8	6	2	5	1
6	3	8	5	1	2	9	7	4
9	8	6	1	3	4	7	2	5
5	2	1	6	7	8	3	4	9
3	4	7	2	9	5	1	8	6

(f) Solution to puzzle

Fig. 9. Three super difficult Sudoku puzzles and their solutions.

From Table II, we can see that in these six Sudoku puzzles, LSGA, NB-COIN, and GA-II all obtain the final results in all 100 runs, while GA-I in Hard 106 only finds the solution in 96 runs. Thus, LSGA, NB-COIN, and GA-II are better than GA-I. Subsequently, compared with NB-COIN, the performance of LSGA is worse than NB-COIN when solving easy-level puzzles. Furthermore, as NB-COIN depends on probability distributions to generate solutions for Sudoku, NB-COIN is less influenced by local optimal solutions than other GAs. Specifically, both LSGA and the other comparison algorithms solve Hard 106 with more generations than Hard 77, but the performance of NB-COIN on the two puzzles is not very different. By analyzing the solution process, we found that there is a very competitive local optimal solution in the Hard 106. This local optimal solution has only two columns that do not conform to the rules of Sudoku, but its structure is different from the best solution. As a result, the GAs can easily fall into this local optimum. In general, comparing the results of solving medium-level, and hard-level Sudoku puzzles, the average number of generations of LSGA is less than GA-I, GA-II, and NB-COIN. Therefore, the performance of LSGA in solving Sudoku puzzles is very competitive.

Next, we conduct an experiment on three so-called super difficult Sudoku puzzles named Super Difficult-1 (SD1), AI-Escargot (SD2), and Super Difficult-2 (SD3) selected from [14]. These are shown in Fig. 9. Among these three puzzles, the AI Escargot is one of the most difficult Sudoku puzzles in the world [46]. Table III shows the comparison result of LSGA and some other algorithms that have successfully solved these super difficult puzzles: GA-III [14], GPU-GA [47], and GA-I [46]. Each algorithm runs 100 times on each puzzle, where *Succ_Count* is the solution success rate among the 100 runs within 1×10^4 generations, and *Avg_Gen* is the average number of generations required to find the optimal solution.

From Table III, we see that LSGA and GPU-GA can solve all puzzles with 100% success rate, while GA-I and GA-III cannot. Moreover, both LSGA and GPU-GA require fewer number of generations than the other algorithms in each puzzle, and LSGA requires the fewest. Furthermore, we evaluate the difficulty of SD1, SD2, and SD3 with the help of the Sudoku Explainer (SE) and get scores of 7.2, 10.5, and 2.8, respectively, which means, for the Sudoku solving methods that have been recorded in SE (like WXYZ-Wing, Swampfish, ALS-Wing, etc.), SD2 is very difficult to solve, but SD3 is much simpler. However, the experimental results in Table III are different. Compared with SD1 and SD2, LSGA uses much more generations to solve the SD3, this situation not only occurs in LSGA, but also in other compared GAs in Table III. Therefore, we consider that if some known methods for solving Sudoku can be introduced into GA, the efficiency of solving difficult Sudoku puzzles could be greatly improved.

B. *Statistical Performance on Open Sudoku Puzzles*

To illustrate the statistical performance of LSGA on more Sudoku puzzles, we conduct experiments based on a large number of Sudoku puzzles selected from the open website www.websudoku.com. We select Sudoku puzzles from four difficulty levels: Easy, Medium, Hard, and Evil. In each difficulty level, 30 puzzles are randomly selected. Therefore, totally 120 puzzles are adopted for testing. The details of all the 120 puzzles are provided in the Supplemental_Material. The configurations of LSGA are the same as those in Table I. LSGA runs 10 times on each puzzle and the average performance of the 10 runs is calculated and given as *Avg_Gen* in Table S.I in the Supplemental_Material. Then, the mean performance of the LSGA on all the 30 puzzles (i.e., the 30 *Avg_Gen* values) in each difficulty level is given as *Mean_Gen* in the last row of Table S.I. Moreover, all the *Mean_Gen* values of the four difficulty levels and other statistical values are reported in Table IV. For example, in the second row of Table IV for all the 30 puzzles in easy level, the average number of generations needed by LSGA to obtain the optimal solution to each puzzle among the 10 runs is calculated, and then the maximal average number and the minimal average number among the 30 puzzles are given as *Max_Gen* and *Min_Gen*. Moreover, the mean of the 30 average numbers is given as *Mean_Gen* and the *Mean_Succ_Rate* is the success rate of LSGA in solving all the 30 puzzles in all the 10 runs.

From Table IV, we see that LSGA efficiently solves all puzzles. As the difficulty level increases, the number of generations needed by LSGA to obtain the optimal solution also increases exponentially, especially for the puzzles of the evil level. Therefore, we conduct a further investigation on the factors affecting the performance of the LSGA in the following part.

TABLE IV. RESULTS OF PROPOSED LSGA FOR SOLVING SUDOKU PUZZLES AT WWW.WEBSUDOKU.COM

Level	*Mean_Succ_Rate*	*Mean_Gen*	*Max_Gen*	*Min_Gen*
Easy	100%	4.8	9.4	2.7
Medium	100%	17.3	32.8	6.1
Hard	100%	70.4	151.1	19.6
Evil	100%	107.6	449.5	22.6

C. Further Investigation and Discussion

To further study the factors affecting the performance of LSGA, we decide to rate and generate Sudoku puzzles by using Sudoku Explainer (SE). The score of a Sudoku puzzle in SE is determined by the complexity of the skills required to solve it. The more complex the skills required to solve a Sudoku puzzle, the higher SE score it will get, which can determine the difficulty level of the Sudoku puzzle accordingly. This type of evaluation is very effective for players and is widely used [4]. Therefore, we use SE to generate Sudoku puzzles with 7 difficulty levels, each difficulty level containing 10 different Sudoku puzzles. These 7 levels are called Easy, Medium, Hard, Superior, Fiendish, Super, and Advance, and their SE score intervals are [1.0, 1.2], [1.3, 1.5], [1.6, 2.6], [2.7, 3.9], [4.0, 5.9], [6.0, 6.9], and [7.0, 8.0], respectively. The details of all the 70 puzzles are given in the Supplemental_Material. The LSGA is adopted to solve these Sudoku puzzles. Similar to the experiments in Section IV-B, each puzzle is solved 10 times and the average number of generations needed by LSGA to obtain the optimal solution of the 10 runs is calculated. Then, we can obtain 10 average results on each difficulty level (i.e., there are 10 puzzles and each puzzle has an average result). The details of these 10 average results are given in Table V and their distribution is plotted as Box in Fig. 10. There are 7 columns in Table V and 7 Boxes in Fig. 10 for 7 difficulty levels. Moreover, we also look into the number of given numbers in all the 70 puzzles. Fig. 11 shows the average number of generations needed by LSGA to solve Sudoku puzzles with different given numbers. For example, the first bar means that, there may be several puzzles among the 70 puzzles that are with 23 given numbers, then the average number of generations needed by LSGA to solve each of these several puzzles among the 10 runs is calculated, and at last the mean of these several average values is calculated, which is 177.

As shown in Fig. 10 and Table V, we can conclude that the required generations for LSGA to solve Sudoku are not significantly affected by difficulty levels. For example, the generations required to solve most Sudoku puzzles at the Hard and Superior levels are less than that at the Medium level. That is, LSGA inherits the problem-independent characteristics of GA and is more general for Sudoku puzzles. Moreover, as shown in Fig. 11, we can conclude that there is a correlation between the difficulty of solving Sudoku puzzles and the number of given numbers. More given numbers can give LSGA more help in finding a solution, because the given numbers can effectively reduce the search space and eliminate interference solutions. However, the relationship between the given numbers and difficulty is not strictly linear or exponential. That is, there exists the situation that some Sudoku puzzles with more given numbers but are more difficult to be solved because these given numbers do not provide enough clues to determine the non-given numbers. For example, in Section IV-A, Sudoku puzzle Hard 106 (with 24 given numbers) is more difficult than Sudoku puzzle SD2 (with 23 given numbers), because some Sudoku puzzles like Hard 106 have many local optimal solutions and their given numbers cannot effectively help LSGA to escape from the local optimal solutions.

TABLE V. AVERAGE GENERATIONS NEEDED BY LSGA TO SOLVE EACH SUDOKU PUZZLE WITH DIFFERENT DIFFICULTY LEVELS

Level* / Puzzle	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	42.8	130.9	96.3	88.4	847.1	145.2	147.7
2	67.1	68.1	11.1	36	135.7	107	63.3
3	3.1	19.6	21.5	94.1	36.1	28.4	78.6
4	197.4	166.3	22	253.7	21.1	10.2	150.3
5	30.1	91.5	34.7	211.9	17.5	97.9	84.7
6	9.6	241.2	21.6	24.3	30.9	92.5	315.7
7	24.7	80.7	70.5	62.1	258.4	46.4	88.8
8	9.9	247.3	64.2	11.5	54.5	24.9	43.3
9	12.7	44.7	155.9	111.9	102.4	53.6	50
10	102.8	237.4	140.4	100.1	248.6	975.2	145.4
Mean	50.02	132.77	63.82	99.4	175.23	158.13	116.78

*The level 1 to 7 means Easy, Medium, Hard, Superior, Fiendish, Super, and Advance, respectively.

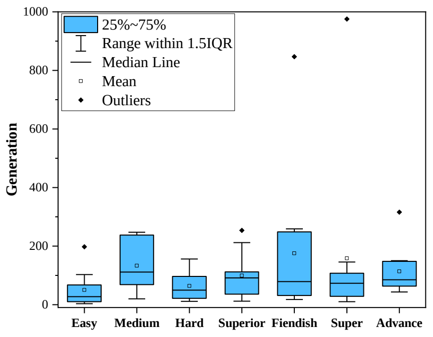


Fig. 10. Distributions of the average generations needed by LSGA to solve Sudoku puzzles with different difficulty levels.

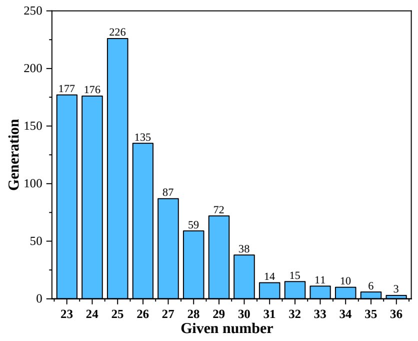


Fig. 11. Mean generations needed by LSGA to solve Sudoku puzzles with different given numbers.

V. Conclusion

In this paper, we propose an improved GA with a local search (named LSGA) for Sudoku. In particular, we adopt a matrix-based encoding GA and devise mutation and crossover operators for this coding scheme. Then, to improve the convergence speed of LSGA, a local search method incorporating column and sub-block search is proposed. Finally, by comparing with GA-based algorithms in different dimensions and levels of Sudoku puzzles, LSGA successfully solves all of these puzzles and shows good performance.

LSGA can also be applied to solve other types of Sudoku puzzles such as Mini Sudoku and Ring Sudoku. However, for Sudoku variants such as Killer Sudoku and Kakuro Sudoku [48], the initialization and local search strategies need to be redesigned because the local search in LSGA is designed for regularly shaped sub-blocks. Furthermore, for puzzles without sub-blocks, such as those of Futoshiki and Takuzu [49], the column local search strategy is still applicable. Therefore, LSGA deserves further research to better solve other Sudoku puzzles. Although our algorithm is successful in solving many Sudoku puzzles, there is still room for improvement. For example, with the help of manual Sudoku solving methods, such as direct hidden pair and fish methods [50], humans can easily find numbers in the irrational position and adjust them, whereas LSGA needs several, tens, or even hundreds of generations to achieve the same results. Therefore, in the future, we can improve the performance of LSGA by combining it with other Sudoku-solving methods.

References

[1] J.-P. Delahaye, “The science behind SUDOKU,” *Sci. Am.*, vol. 294, no. 6, pp. 80-87, Jun. 2006.

[2] X. Qi, G. Li, N. Wang, X. Wang, and L. Wen, “Method Study on Solving Sudoku Problem,” in *Int. Conf. Data Sci. and Bus. Anal.*, Istanbul, Turkey. 2019, pp. 269-271.

[3] T. Yato and T. Seta, “Complexity and completeness of finding another solution and its application to puzzles,” *Comp. Sci.*, vol. E86-A, no.5, pp. 1052-1060, May. 2003.

[4] M. Henz and H.-M. Truong, “SudokuSat—A Tool for Analyzing Difficult Sudoku Puzzles,” *Tools Appl. Artifi. Intell.*, vol 166. Springer, pp. 25-35, 2009.

[5] S. Jana, A. K. Maji, and R. K. Pal, “A novel SPN-based video steganographic scheme using Sudoku puzzle for secured data hiding,” *Innovations Syst. Soft. Eng.*, vol. 15, no. 1, pp. 65-73, Jan. 2019.

[6] R. M. Nicholas, D. B. Travis, and M. N. Ram, “Sudoku based phase-coded radar waveforms,” *Radar Sensor Technol.*, vol. 11742, pp. 117420L, April, 2021. [Online]. doi: 10.1117/12.2588316.

[7] S. Jose and R. Abraham, “Influence of Chess and Sudoku on cognitive abilities of secondary school students,” *Iss. and Ideas in Educ.*, vol. 7, no. 1, pp. 27-34, Jul. 2019.

[8] X. Li, Y. Luo, and W. Bian, “Retracing extended sudoku matrix for high-capacity image steganography,” *Multimedia Tools Appl.*, vol. 80, no. 12, pp. 18627-18651, Feb. 2021.

[9] M. Horoufiany and R. Ghandehari, “Optimization of the Sudoku based reconfiguration technique for PV arrays power enhancement under mutual shading conditions,” *Solar Energy*, vol. 159, pp. 1037-1046, Jan. 2018.

[10] M. Harrysson and H. Laestander, “Solving Sudoku efficiently with Dancing Links”, *Exp. Math.*, vol. 23, pp. 190-217, Dec. 2014.

[11] J. Gunther and T. Moon, “Entropy minimization for solving Sudoku,” *IEEE Trans. Signal Proces.*, vol. 60, pp. 508-513, Jan. 2012.

[12] L. Clementis, “Advantage of parallel simulated annealing optimization by solving Sudoku puzzle,” in *Emerg. Trends Robot. Intel. Syst.*, Kosice, Slovakia, 2015, pp. 207-213.

[13] K. Waiyapara, W. Wattanapornprom, and P. Chongstitvatana, “Solving Sudoku puzzles with node based Coincidence algorithm,” in *Joint Conf. Comput. Sci. Softw. Eng.*, Khon Kaen, Thailand, May. 2013, pp. 11-16.

[14] M. Becker and S. Balci, “Improving an evolutionary approach to Sudoku puzzles by intermediate optimization of the population,” *Inform. Sci. Appl.*, Daegu, Korea, Jul. 2019, pp. 369-375.

[15] T. Mantere and J. Koljonen, “Solving and rating sudoku puzzles with genetic algorithms,” *Artificial Intel. Conf. Soiety*, Espoo, Finland, Oct. 2006, pp. 86-92.

[16] H. Lloyd and M. Amos, “Solving Sudoku with ant colony optimization,” *IEEE Trans. Games*, vol. 12, no.3, pp. 302-311, Sept. 2020.

[17] D. E. Knuth, “Dancing links,” *Millenial Perspect. Comput. Sci.*, Palgrave Macmillan, Basingstoke, USA: Boston, 2000, pp. 187-214.

[18] F. Gerges, G. Zouein, and D. Azar, “Genetic algorithms with local optima handling to solve Sudoku puzzles,” in *Int. Conf. Comput. Artif. Intell.*, Chengdu, China, Mar. 2018, pp. 19-20.

[19] A. Z. Sevkli and K. A. Hamza, “General variable neighborhood search for solving Sudoku puzzles: Unfiltered and filtered models,” *Soft Comput.*, vol. 23, no.15, pp. 6585-6601, Aug. 2019.

[20] M. A. Al-Betar, M. A. Awadallah, A. L. Bolaji, and B. O. Alijla, “β-Hill Climbing Algorithm for Sudoku Game,” in *Int. Confe. Inform. Commun.* Tech., Gaza, Palestine, May. 2017, pp. 84-88.

[21] J. Horn, “Solving a large sudoku by co-evolving numerals,” in *Genet. and Evolut. Comput. Conf. Companion*, New York, United States, Jul. 2017, pp. 29-30.

[22] J. -Y. Li, Z. -H. Zhan, H. Wang and J. Zhang, “Data-Driven Evolutionary Algorithm With Perturbation-Based Ensemble Surrogates,” *IEEE Trans. Cyber.*, vol. 51, no. 8, pp. 3925-3937, Aug. 2021.

[23] Z. H. Zhan, *et al.*, “Matrix-based evolutionary computation,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 6, no. 2, pp. 315-328, Apr. 2022.

[24] J. -Y. Li, Z. -H. Zhan, C. Wang, H. Jin and J. Zhang, “Boosting Data-Driven Evolutionary Algorithm With Localized Data Generation,” *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 923-937, Oct. 2020.

[25] X. Zhang, Z. -H. Zhan, W. Fang, P. Qian and J. Zhang, “Multipopulation Ant Colony System With Knowledge-Based Local Searches for Multiobjective Supply Chain Configuration,” *IEEE Trans. Evolut. Comput.*, vol. 26, no. 3, pp. 512-526, June 2022.

[26] L. Shi, Z. H. Zhan, D. Liang, and J. Zhang, “Memory-based ant colony system approach for multi-source data associated dynamic electric vehicle dispatch optimization,” *IEEE Trans. Intell. Transp.*, early access, doi: 10.1109/TITS.2022.3150471.

[27] J. Y. Li, *et al.*, “A multipopulation multiobjective ant colony system considering travel and prevention costs for vehicle routing in COVID-19-like epidemics,” *IEEE Trans. Intell. Transp.*, early access, doi: 10.1109/TITS.2022.3180760.

[28] J. R. Jian, Z. G. Chen, Z. H. Zhan, and J. Zhang, “Region encoding helps evolutionary computation evolve faster: A new solution encoding scheme in particle swarm for large-scale optimization,” *IEEE Trans. Evolut. Comput.*, vol. 25, no. 4, pp. 779-793, Aug. 2021.

[29] J. Y. Li, Z. H. Zhan, R. D. Liu, C. Wang, S. Kwong, and J. Zhang, “Generation level parallelism for evolutionary computation: A pipeline-based parallel particle swarm optimization,” *IEEE Trans. Cybern.*, vol. 51, no. 10, pp. 4848-4859, Oct. 2021.

[30] X. F. Liu, Z. H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang, “Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization,” *IEEE Trans. Evolut. Comput.*, vol. 23, no. 4, pp. 587-602, Aug. 2019.

[31] Z. J. Wang, *et al.*, “Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling,” *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715-2729, Jun. 2020.

[32] X. Xia, L. Gui, F. Yu, H. Wu, B. Wei, Y. Zhang, and Z. H. Zhan, “Triple archives particle swarm optimization,” *IEEE Trans. Cybern.*, vol. 50, no. 12, pp. 4862-4875, Dec. 2020.

[33] J. Y. Li, Z. H. Zhan, K. C. Tan, and J. Zhang, “Dual differential grouping: A more general decomposition method for large-scale optimization,” *IEEE Trans. Cybern.*, early access, doi: 10.1109/TCYB.2022.3158391.

[34] J. Y. Li, K. J. Du, Z. H. Zhan, H. Wang, and J. Zhang, “Distributed differential evolution with adaptive resource allocation,” *IEEE Trans. Cybern.*, early access, doi: 10.1109/TCYB.2022.3153964.

[35] Z. H. Zhan,Z. J. Wang, H. Jin, and J. Zhang, “Adaptive distributed differential evolution,” *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4633-4647, Nov. 2020.

[36] J. Y. Li, Z. H. Zhan, K. C. Tan, and J. Zhang, “A meta-knowledge

transfer-based differential evolution for multitask optimization," *IEEE Trans. Evolut. Comput.*, vol. 26, no. 4, pp. 719-734, Aug. 2022.
[37] J. Y. Li, Z. H. Zhan, J. Xu, S. Kwong, and J. Zhang, "Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, early access, doi: 10.1109/TNNLS.2021.3106399.
[38] Z. H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artif. Intell. Rev.*, vol. 55, no. 1, pp. 59-110, Jan. 2022.
[39] Z. H. Zhan, J. Y. Li, and J. Zhang, "Evolutionary deep learning: A survey," *Neurocomputing*, vol. 483, pp. 42-58, Apr. 2022.
[40] Z. G. Chen, Z. H. Zhan, S. Kwong, and J. Zhang, "Evolutionary computation for intelligent transportation in smart cities: A survey," *IEEE IEEE Comput. Intell. M.*, vol. 17, no. 2, pp. 83-102, May, 2022.
[41] J. Y. Li, Z. H. Zhan, and J. Zhang, "Evolutionary computation for expensive optimization: A survey," *Mach. Intell. Res.*, vol. 19, no. 1, pp. 3-23, Jan. 2022.
[42] N. Pathak and R. Kumar, "Improved wisdom of crowds heuristic for solving sudoku puzzles," *Soft Comput. Signal Proc.*, Singapore, Jan. 2019, pp. 369-377.
[43] X. Q. Deng and Y. D. Li, "A novel hybrid genetic algorithm for solving Sudoku puzzles," *Optim. Lett.*, vol. 7, no.2, pp. 241-257, Oct. 2013.
[44] K. Rodríguez-Vázquez, "GA and entropy objective function for solving sudoku puzzle," in *IEEE CEC*, New York, USA, Jul. 2018, pp. 67-68.
[45] N. Musliu and F. Winter, "A Hybrid Approach for the Sudoku Problem: Using Constraint Programming in Iterated Local Search," in *IEEE Intell. Syst.*, vol. 32, no. 2, pp. 52-62, Mar.-Apr. 2017.
[46] Y. Sato and H. Inoue, "Solving Sudoku with genetic operations that preserve building blocks," in *Conf. Comput. Intell. Games*, Copenhagen, Denmark, Aug. 2010, pp. 23-29.
[47] Y. Sato, N. Hasegawa, and M. Sato, "GPU acceleration for Sudoku solution with genetic operations," in *IEEE CEC*, June 2011, pp. 296-303.
[48] Pillay N. "Finding solutions to Sudoku puzzles using human intuitive heuristics". *S. Afr. Comput. Journal*, vol.49, no.1, pp. 25-34, Sept. 2012.
[49] Groza, Adrian. "Japanese Puzzles." *Modelling Puzzles in First Order Logic*, Springer, Berlin, 2021, pp. 221-253.
[50] X. Peng, Y. Huang, and F. Li, "A steganography scheme in a low-bit rate speech codec based on 3D-sudoku matrix," in *IEEE ICCSN*, June 2016, pp. 13-18.

Chuan Wang received the B.S. degree in computer science and M.S. degree in education from Henan Normal University, Xinxiang, China, in 1999 and 2009 respectively. He is currently an Associate Professor with College of Software, Henan Normal University.

His current research interests computational intelligence and its applications on intelligent information processing and big data.

Bing Sun (Student Member, IEEE) received the B.S. degree in computer science and technology from Henan University of Science and Technology, Henan, China, in 2020, where he is currently pursuing the M. S. degree in electronic and information engineering with Henan Normal University.

His current research interests mainly include evolutionary computation, swarm intelligence, and their applications in real-world problems.

Ke-Jing Du received the B.S. degree from Sun Yat-Sen University, Guangzhou, China, in 2012, and the M.S. degree from City University of Hong Kong, Hong Kong, in 2014. She is currently working toward the Ph.D. degree in Victoria University, Melbourne, VIC, Australia.

Her current research interests include evolutionary computation (EC) and supply chain management, especially the distributed EC and application of EC in supply chain, feature selection, and games.

Jian-Yu Li (Member, IEEE) received the Bachelor's degree and the Ph. D. degree in Computer Science and Technology from the South China University of Technology, China, in 2018 and 2022, respectively.

His research interests mainly include computational intelligence, data-driven optimization, machine learning including deep learning, and their applications in real-world problems, and in environments of distributed computing and big data. He has been invited as the reviewer of the *IEEE Transactions on Evolutionary Computation* and the *Neurocomputing* journal, and the program community member and reviewer of some international conferences.

Zhi-Hui Zhan (Senior Member, IEEE) received the Bachelor's degree and the Ph. D. degree in Computer Science from the Sun Yat-Sen University, Guangzhou China, in 2007 and 2013, respectively.

He is currently the Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include evolutionary computation, swarm intelligence, and their applications in real-world problems and environments of cloud computing and big data.

Dr. Zhan was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award in 2021, the Outstanding Youth Science Foundation from National Natural Science Foundations of China (NSFC) in 2018, and the Wu Wen-Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. His doctoral dissertation was awarded the IEEE CIS Outstanding Ph. D. Dissertation and the China Computer Federation Outstanding Ph. D. Dissertation. He is one of the World's Top 2% Scientists for both Career-Long Impact and Year Impact in Artificial Intelligence and one of the Highly Cited Chinese Researchers in Computer Science. He is currently the Chair of Membership Development Committee in IEEE Guangzhou Section and the Vice-Chair of IEEE CIS Guangzhou Chapter. He is currently an Associate Editor of the *IEEE Transactions on Evolutionary Computation*, the *Neurocomputing*, the *Memetic Computing*, and the *Machine Intelligence Research*.

Sang-Woon Jeon (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Yonsei University, Seoul, South Korea, in 2003 and 2006, respectively, and the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2011.

He has been an Associate Professor with the Department of Military Information Engineering (undergraduate school) and the Department of Electronics and Communication Engineering (graduate school), Hanyang University, Ansan, South Korea, since 2017. From 2011 to 2013, he was a Postdoctoral Associate at the School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland. From 2013 to 2017, he was an Assistant Professor with the Department of Information and Communication Engineering, Andong National University, Andong, Korea. His research interests include network information theory, wireless communications, sensor networks, and their applications to the Internet of Things and big data.

Dr. Jeon was a recipient of the Haedong Young Scholar Award in 2017, which was sponsored by the Haedong Foundation and given by the Korea Institute of Communications and Information Science (KICS), the Best Paper Award of the KICS journals in 2016, the Best Paper Award of the IEEE International Conference on Communications in 2015, the Best Thesis Award from the Department of Electrical Engineering, KAIST, in 2012, the Best Paper Award of the KICS Summer Conference in 2010, and the Bronze Prize of the Samsung Humantech Paper Awards in 2009.

Hua Wang (Senior Member, IEEE) received his Ph.D. degree from the University of Southern Queensland, Toowoomba, Qld., Australia in 2004.

He is now a full-time Professor at Victoria University. He has expertise in electronic commerce, business process modeling, and enterprise architecture. As a Chief Investigator, three Australian Research Council (ARC) Discovery grants have been awarded since 2006, and 200 peer-reviewed scholar papers have been published.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong in 2002.

He is currently a Korea Brain Pool Fellow Professor with Hanyang University, South Korea. His current research interests include computational intelligence, cloud computing, operations research, and power electronic circuits. He has published more than 150 IEEE Transactions papers in his research areas.

Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, The National Science Fund for Distinguished Young Scholars of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the *IEEE Transactions on Evolutionary Computation* and the *IEEE Transactions on Cybernetics*.