mathematics

Article

A Novel Discrete Differential Evolution with Varying Variables for the Deficiency Number of Mahjong Hand

Xueqing Yan [1] and Yongming Li [2,*]

[1] School of Computer Science, Shaanxi Normal University, Xi'an 710062, China; xueqingyan@snnu.edu.cn
[2] School of Mathematics and Statistics, Shaanxi Normal University, Xi'an 710062, China
* Correspondence: liyongm@snnu.edu.cn

Abstract: The deficiency number of one hand, i.e., the number of tiles needed to change in order to win, is an important factor in the game Mahjong, and plays a significant role in the development of artificial intelligence (AI) for Mahjong. However, it is often difficult to compute due to the large amount of possible combinations of tiles. In this paper, a novel discrete differential evolution (DE) algorithm is presented to calculate the deficiency number of the tiles. In detail, to decrease the difficulty of computing the deficiency number, some pretreatment mechanisms are first put forward to convert it into a simple combinatorial optimization problem with varying variables by changing its search space. Subsequently, by means of the superior framework of DE, a novel discrete DE algorithm is specially developed for the simplified problem through devising proper initialization, a mapping solution method, a repairing solution technique, a fitness evaluation approach, and mutation and crossover operations. Finally, several experiments are designed and conducted to evaluate the performance of the proposed algorithm by comparing it with the tree search algorithm and three other kinds of metaheuristic methods on a large number of various test cases. Experimental results indicate that the proposed algorithm is efficient and promising.

Keywords: Mahjong; differential evolution; deficiency number; combinatorial optimization

MSC: 68T20; 90C27

Citation: Yan, X.; Li, Y. A Novel Discrete Differential Evolution with Varying Variables for the Deficiency Number of Mahjong Hand. *Mathematics* **2023**, *11*, 2135. https://doi.org/10.3390/math11092135

Academic Editor: Jian Dong

Received: 20 March 2023
Revised: 27 April 2023
Accepted: 30 April 2023
Published: 2 May 2023

1. Introduction

Mahjong is a traditional, Chinese, tile-based game with a long history and is often played by four people [1,2]. In this game, each player is devoted to devising a proper strategy to win as soon as possible, with luck also playing an important role in this process. Due to its imperfect information, Mahjong has become a popular testbed for artificial intelligence (AI) research [3–9]. Nowadays, various variants of Mahjong with different rules for computing paybacks and/or the legality of actions have emerged due to the individual cultures of different regions and countries. However, they all have similar processes and can be easily extended by the basic version of Mahjong. Therefore, the basic version of Mahjong is just considered in the following without loss of generality.

In the basic version of Mahjong, four players are involved, and 108 tiles are used, consisting of 36 tiles of bamboo type, 36 tiles of character type, and 36 tiles of dot type. At the start of the game, each player in turn draws thirteen tiles from the tile wall, and then they each take one tile from the tile wall and discard one tile from their hand. When one player gets a winning hand or there are no tiles left in the tile wall, the game ends. So, all players need to continuously change their hands' tiles in order to ensure that their hands win quickly, and these processes involve a number of evaluations on the quality of various cases of tiles, i.e., calculating the minimum number of tiles needed to be changed to win, which is called the deficiency number [10]. Thereby, computing the deficiency number has an important role in Mahjong, and can promote AI development for the game. Moreover, the winning hand closely depends on the combinations of the tiles, and these combinations include a sequence of three or four identical tiles (called a pong or kong), a sequence of three consecutive tiles with the same type (called a chow), and a pair of identical tiles (called a pair or an eye). A pong or a chow is also called a meld, and a pseudomeld (abbr. pmeld) refers to a pair of tiles that can constitute a meld. Therefore, computing the deficiency number is in fact a combinatorial optimization problem, which is often not easily calculated due to its large search space.

As far as we know, there are very few studies on the calculation of the deficiency number at present [10–12]. Specifically, in the paper [10], Li and Yan presented a recursive method and a tree-based method for calculating the deficiency number. In the recursive method, for one hand with 14 tiles, all cases of 14 tiles with k deficiency must be known in advance before the deficiency number of one hand is determined to be $k+1$. In the tree-based method, all pseudo-decompositions of the tiles must be found and evaluated, and then the deficiency number is obtained by the minimum cost of these pseudo-decompositions. Herein, a pseudo-decomposition is a sequence π of five subsequences, $\pi[0], \pi[1], \ldots, \pi[4]$, where $\pi[i]$ for $0 \leq i \leq 3$ can be a meld, a pmeld, a single tile, or empty, and $\pi[4]$ can be a pair, a single tile, or empty. The cost of each pseudo-decomposition is the number of missing tiles compared to the current hand. Wang et al. [11] constructed a theoretical model of weighted restarting automaton to compute the deficiency number of 14 tiles, where the tiles are combined into melds and eyes during the process of simplification, and the number of changed tiles is counted using its weight function. Recently, Wang et al. [12] further proposed an efficient algorithmic approach, where the tiles in the player's hand are first divided into several groups, such as melds, pseudomelds, and isolated tiles, and then the deficiency number is computed based on the number of pseudomelds and that of the isolated tiles. Although these approaches can calculate the deficiency number of a Mahjong hand, the full searches are all implicit in them, which often take too much computational time and thus cannot meet the actual demand of response time in the game. Therefore, it is necessary to develop a more promising search approach for the deficiency number.

As is well known, because of the lower requirement on the problem to be solved, heuristic intelligent search/optimization methods have been regarded as the most important tools for solving real problems, especially complicated ones. In addition, various metaheuristic algorithms have been presented by simulating nature phenomena, animal behaviors, human activities, or physical criteria, such as the genetic algorithm (GA) [13], differential evolution (DE) [14], particle swarm optimization (PSO) [15], artificial bee colony algorithm [16], water cycle algorithm (WCA) [17], squirrel search algorithm [18], gravitational search algorithm (GSA) [19], teaching–learning-based optimization (TLBO) [20], gaining–sharing knowledge-based algorithm [21], and so on. In detail, more metaheuristic algorithms can be found in [22], and they have been widely researched and successfully applied in many scientific fields and practical problems up to now, including data clustering [23,24], stock portfolios [25], knapsack optimization problems [26], multitask optimization [27], and multimodal optimization [28].

As pointed out in [22], among the existing intelligent approaches, differential evolution (DE) [14] is one of the most popular population-based stochastic optimizers, and has been proven to be more efficient and robust on various problems. Due to its simplicity and simple implementation, DE always attracts more attention from researchers, and numerous DE variants have been put forward to strengthen its performance [29–34] and/or solve special practical problems [35–39]. For example, by dividing a population into multiple swarms and randomly selecting the solutions with better fitness values in each swarm to conduct mutations, Wang et al. [29] developed a novel adaptive DE algorithm. Through making full use of the information of the best neighbor, one randomly selected neighbor, and the current individual for each individual to predict the promising region, Yan and Tian [30] presented an enhanced adaptive DE variant, while, by adaptively combining the benefits of multiple operators, Yi et al. [34] developed a novel approach for continuous optimization problems. Moreover, by designing a Taper-shaped transfer function based on power functions to transform a real vector representing the individual encoding into a binary vector, He et al. [35] proposed a novel binary differential evolution algorithm for binary optimization problems. Meanwhile, for traveling salesman problems, Ali et al. [38] proposed an improved discrete DE version, where an enhanced mapping mechanism is devised to map continuous variables to discrete ones, a k-means clustering-based repairing method is devised to enhance the solutions in the initial population, and an ensemble of mutation strategies is used to maintain its exploration capability. In particular, a large number of numerical experiments were conducted in their papers, and the numerical results validated their effectiveness and superiority. Thereby, DE has a greater potential in a promising search performance. Following this, we adopt the framework of DE to calculate the deficiency number of the tiles in this paper, so as to obtain a promising performance. Specifically, more detailed research on the improvements and applications of DE can be further referred to in [40].

In this paper, a more efficient method is researched to calculate the deficiency number of a Mahjong hand, and the framework of DE is adopted to achieve this, based on its intrinsic advantages. In detail, in order to reduce the difficulty of computing the deficiency number, some pretreatment mechanisms are first devised to convert the original problem into a simple combinatorial optimization problem, where the dimensions of the search space are degraded to five, and each feasible solution may have different lengths. Noticeably, this makes the dealing problem significantly different to the existing studied discrete and/or combinatorial optimization problems, in which the size of every solution is fixed and consistent. Moreover, for this converted new problem, based on the basic framework of DE, a novel discrete DE (NDDE) algorithm is then specially presented by devising proper initialization, a mapping solution method, a repairing solution technique, a fitness evaluation approach, and mutation and crossover operations, which aim to meet the available search requirement of the discrete space and the characteristic of the varying sizes of different individuals. Then, the proposed NDDE algorithm is capable of computing the deficiency number of one hand efficiently and effectively. Finally, a large number of experiments are designed and conducted to verify the performance of the NDDE algorithm. Specifically, three representative data sets are employed and tested, including 118,800 hands with one type, 100,000 hands with two types, and 100,000 hands with three types, and the NDDE algorithm is compared with the tree search method in [10] and three other kinds of metaheuristic methods. The sensitivity of the parameters involved in the NDDE algorithm is also investigated. The experimental results show that the proposed algorithm has a promising performance for the deficiency number of the hand.

For clarity, compared with the existing works, the main contributions and novelties of this paper are described as follows.

(1) To our knowledge, the research presented in the paper is the first to utilize a heuristic intelligent algorithm for computing the deficiency number of a Mahjong hand. In fact, the works on computing the deficiency number of one hand are still rare up to now, and all of them adopt a deterministic approach. Thus, this study may provide a new alternative for devising more effective and efficient methods for calculating the deficiency number.

(2) The original problem of computing the deficiency number is converted into a more simple combinatorial optimization problem. This may effectively reduce the difficulty and computational costs of solving it.

(3) To solve the simplified problem above, a novel discrete DE (NDDE) algorithm is further specially proposed by devising proper initialization, a mapping solution method, a repairing solution technique, a fitness evaluation approach, and mutation and crossover operations. Significantly, the simplified problem has the characteristic that the feasible solutions may have various lengths, which is not involved in the previous discrete and/or combinatorial optimization problems. Therefore, the proposed algorithm has different and more rigorous design requirements.

(4) A large number of experiments are designed and conducted to verify the performance of the NDDE algorithm, where three representative data sets are employed and tested, including 118,800 hands with one type, 100,000 hands with two types, and 100,000 hands with three types. Moreover, the NDDE algorithm is compared with the tree search method in [10] and three other kinds of metaheuristic methods, and the sensitivity of the parameters involved in the NDDE algorithm is also investigated. Experimental results indicate that the NDDE algorithm is a more promising technique for the deficiency number of the hand.

Finally, it should also be mentioned that the reason the framework of DE is chosen to design the solver in this paper is solely because much of the research reported in the literature has proven its superiority on various problems, but other metaheuristic algorithms can also be adopted here, which we will further study in our future work.

The reminder of this paper is organized as follows. Some related works and basic notions on Mahjong are presented in Section 2. The proposed algorithm for computing the deficiency number of a Mahjong hand is introduced in Section 3, and the experimental tests are conducted and analyzed in Section 4. Finally, conclusions are drawn in Section 5.

2. Preliminaries

In this section, the related concepts and research on Mahjong and the classical DE algorithm shall be described.

2.1. Related Concepts on Mahjong

In this part, some related concepts in Mahjong are introduced to provide a foundation for the below descriptions. For simplicity, the basic version of Mahjong, denoted as M_0, is considered, which consists of tiles with bamboo type, tiles with character type, and tiles with dot type. Specifically, these tiles with bamboo, character, and dot type are represented as follows:

- Bamboos: $B_1, B_2, \ldots, B_9$.
- Characters: $C_1, C_2, \ldots, C_9$.
- Dots: $D_1, D_2, \ldots, D_9$.

Moreover, in M_0, each of the above tiles has four identical tiles; thus, there are 108 tiles in M_0 in total. Subsequently, some basic notions on Mahjong shall be provided, which mainly refers to the the literature [10,12].

Definition 1. *A* pong *is a sequence of three identical tiles, that is, a pong has the form of $X_iX_iX_i$ for $X \in \{B, C, D\}$ and $1 \le i \le 9$; A* chow *is a sequence of three consecutive tiles with the same type, that is, a chow has the form of $X_iX_{i+1}X_{i+2}$ for $X \in \{B, C, D\}$ and $1 \le i \le 7$; An* eye *(or* pair*) is a pair of identical tiles, that is, an eye has the form of X_iX_i for $X \in \{B, C, D\}$ and $1 \le i \le 9$. A* meld *is either a pong or a chow.*

Definition 2. *A* pseudochow (pchow) *is a pair of tiles X_iX_j with the same type, having $1 \le |j - i| \le 2$ and $X \in \{B, C, D\}$, which can become a chow if we add an appropriate tile with the same type in it. A* pseudomeld (pmeld) *is a pchow or a pair. We say a tile c* completes *a pmeld (ab) if (abc) (after reordering) is a meld. Similarly, a pair is completed from a single tile t if it is obtained by adding an identical tile to t.*

For example, $(B_1B_1B_1)$ is a pong, $(C_1C_2C_3)$ is a chow, (D_1D_1) is a pair, and (B_2B_3) and (C_3C_5) are two pchows.

Definition 3. *A* hand *is a set of 14 tiles, denoted by H, i.e., a sequence of 14 tiles from M_0, where every tile can not appear more than four times.*

Definition 4. *A hand H from M_0 is* winning *or* complete *if it can be decomposed into four melds and one pair. (For ease of presentation, we do not regard a hand with seven pairs as complete.) Given* *a complete hand H, a* decomposition *π of H is a sequence of five subsequences of H, where $\pi[i]$ is a meld for $0 \leq i \leq 3$ and $\pi[4]$ is a pair. If this is the case, we call $\pi[4]$ the* eye *of this decomposition.*

For example, the hand $H_1 = (B_1B_2B_3B_3B_3B_8B_8B_8)(C_4C_5C_6)(D_6D_7D_8)$ is a winning or complete hand, and its decomposition is $\pi = (B_1B_2B_3)(B_8B_8B_8)(C_4C_5C_6)(D_6D_7D_8)(B_3B_3)$.

As is well known, the hands involved in a Mahjong game are mostly incomplete. That is, there is no decomposition defined above for these hands. So, corresponding to the decomposition of a winning hand, another concept of predecomposition is further given here for the hands.

Definition 5. *Given a hand H, the predecomposition (abbr.* pDCMP*) of H is a sequence π of five subsequences, $\pi(1), \ldots, \pi(5)$, of H such that each $\pi(i)$ $(1 \leq i \leq 5)$ is a meld, pair, pchow, or empty.*

Noticeably, unlike the concept of pseudo-decomposition in [10], a single tile is not contained in the predecomposition and the position of the eye is no longer fixed. For example, with respect to the above hand H_1, the sequence $\pi_1 = (B_1B_2)(B_3B_3B_3)(B_8B_8)(C_4C_5C_6)(D_6D_7D_8)$ is one of its pDCMPs.

Definition 6. *Suppose π and π' are two* pDCMPs *for one hand H. We say π' is* finer *than π if $\pi(i)$ is identical to or a subsequence of $\pi'(i)$ for $1 \leq i \leq 5$. A* pDCMP *π is* completable *if there exists a decomposition π^* for H that is finer than π. If this is the case, we call π^* a* completion *of π. Moreover, the cost of a completable* pDCMP *π, written $cost(\pi)$, is the number of missing tiles required to complete π into a decomposition, which consists of four melds and one eye.*

In particular, for one pDCMP π of one hand H, we say it has infinite cost if it is incompletable. Moreover, for the above pDCMP π_1 of H_1, its cost is 1 since it can be completed by one tile, B_3.

Definition 7. *For one hand H, the minimal number of necessary tile changes for making T a winning is called the* deficiency number (or simply *deficiency*) of H. If the deficiency of H is ℓ, we write $dfncy(H) = \ell$. Obviously, for a winning hand H, it holds that $dfncy(H) = 0$.

Based on the above notions, we can see that for one hand, H, its deficiency number can be obtained by finding all its possible predecompositions and then comparing their differences with the ideal decompositions.

2.2. Research on Mahjong Game

With the development of artificial intelligence (AI) techniques, games are continuously regarded as one of the most important test platforms. In particular, for games with perfect information, such as checkers [41], chess [42] and Go [43,44], AI has now even been able to beat the best human players. In contrast, for imperfect information games [45–49], there are still few works since players have to deal with some invisible information during the game, especially for Mahjong.

As stated in the Introduction, there have been various variants of Mahjong due to the unique cultures of each region and country. Among them, Riichi Mahjong is a popular version played in Japan, and most of the current research on Mahjong is based on it [5,8,9,49–52]. Specifically, Li et al. [51] proposed a Mahjong AI system, suphx, based on reinforcement learning, and the test results on the "Tenhou" platform showed its effectiveness. Kurita [5] abstracted the Mahjong process by defining multiple Markov decision processes, and then constructed an effective search tree for optimal decision-making. Mizukami and Tsuruoka [49] built a strong Mahjong AI by modeling opponent players and performing Monte Carlo simulations. Yoshimura et al. [50] proposed a tabu search method of optimal movements without using game records, while Sato et al. [52] presented a new method to classify the opponent players' strategy by analyzing Mahjong playing records. Additionally, for the version of bloody Mahjong, Gao and Li [8] developed a fusion model by using the deep learning and XGBoost algorithm to extract the Mahjong situation features and derive the card strategy, respectively. In particular, a more detailed review about the existing works on Mahjong AI can be further found in ref. [9], where the advantages and disadvantages of each method are analyzed.

Unlike the above works that aim to develop Mahjong AI players, there are still few studies to evaluate the quality of a Mahjong hand, which is more helpful for a player to devise an appropriate strategy during the game [10–12]. In [10], Li and Yan first introduced the notation of the deficiency number for measuring the quality of a hand, and developed two different calculation methods for it, namely, the recursive method and the tree-based method. After this, Wang et al. [11] presented a theoretical model of weighted restarting automaton for computing the deficiency number of 14 tiles. In the developed model, the tiles are combined into melds and eyes in the process of simplification, and the number of changed tiles is counted by its weight function. Moreover, by dividing the tiles into some groups, such as melds, pseudomelds, and isolated tiles, in advance and fully considering the relation between their numbers, Wang et al. [12] recently further proposed an efficient algorithm to calculate the deficiency number of 14 tiles. Even though these above methods have made some progress in measuring the quality of a hand, they all contains the idea of full searches, which might make their computational time too long to timely satisfy the actual demand of response time during a game. Therefore, it is necessary to devise a more promising technique for calculating the deficiency number.

2.3. Traditional DE Algorithm

For the convenience of later descriptions, the detailed operations of a classical DE is drawn as follows, including initialization, mutation, crossover, and selection [14]. Specifically, the minimization problem $\min\{f(\vec{x})|x_{min,j} \le x_j \le x_{max,j}, j = 1, 2, \ldots, D\}$ is considered here, where $\vec{x} = (x_1, x_2, \ldots, x_D)$ is the solution vector, D is the dimension of the search space, and $x_{min,j}$ and $x_{max,j}$ are the lower and upper bounds of the j-th component of the search space, respectively.

First, one population, P^0, consisting of NP solutions is randomly created in the search space. Each solution is denoted by $\vec{x}_i^0 = (x_{i,1}^0, x_{i,2}^0, \ldots, x_{i,D}^0)$ with $i = 1, 2, \ldots, NP$, and NP is the size of population. Concretely, the j-th component of $\vec{x}_i^0$ is generated by

$$x_{i,j}^0 = x_{min,j} + rand \cdot (x_{max,j} - x_{min,j}), \tag{1}$$

where $rand$ is a random number uniformly distributed in [0, 1].

After initialization of the population, for each solution $\vec{x}_i^g$ at g generation, the mutation operation is executed to create its mutation individual $\vec{v}_i^g$. The detailed process of operator 'DE/rand/1' can be provided as

$$\vec{v}_i^g = \vec{x}_{r1}^g + F \cdot (\vec{x}_{r2}^g - \vec{x}_{r3}^g), \tag{2}$$

where r_1, r_2, and r_3 are three random integers in $[1, NP]$ and have $r1 \ne r2 \ne r3 \ne i$, and F is a scaling factor.

Then, by combining the components of $\vec{v}_i^g$ and its corresponding target individual $\vec{x}_i^g$, one trial individual $\vec{u}_i^g = (u_{i,1}^g, u_{i,2}^g, \ldots, u_{i,D}^g)$ is created with the crossover operation. The rule of the binomial crossover method is just described here as:

$$u_{i,j}^g = \begin{cases} v_{i,j}^g, & \text{if } rand \le Cr \text{ } or \text{ } j = randn(i), \\ x_{i,j}^g, & \text{otherwise} \end{cases} \tag{3}$$

where $u_{i,j}^g$, $v_{i,j}^g$, and $x_{i,j}^g$ are, respectively, the j-th components of $\vec{u}_i^g$, $\vec{v}_i^g$, and $\vec{x}_i^g$, $Cr \in [0, 1]$ is the crossover rate, and $randn(i)$ returns a random integer within $[1, D]$, which ensures that $\vec{u}_i^g$ obtains at least one component from $\vec{v}_i^g$.

Finally, the selection operation is conducted to update the current population by comparing each target solution $\vec{x}_i^g$ with its trial vector $\vec{u}_i^g$ based on their fitness values. In particular, the greedy selection strategy can be expressed as follows.

$$\vec{x}_i^{g+1} = \begin{cases} \vec{u}_i^g, & \text{if } f(\vec{u}_i^g) \le f(\vec{x}_i^g) \\ \vec{x}_i^g, & \text{otherwise.} \end{cases} \tag{4}$$

where $f(\cdot)$ is the objective function to be optimized.

Note that DE with (4) will either get better or remain at the same fitness status, but never deteriorate. Meanwhile, when DE is activated for one optimization problem, the mutation, crossover, and selection operations will in turn be executed until one satisfying solution is found or the prescribed termination criterion is met.

3. Proposed Algorithm

As described and discussed in the Introduction, the deficiency number evaluates the quality of a Mahjong hand and is helpful for a player to devise an appropriate strategy, which can facilitate the development of Mahjong AI. However, due to the fact that the tiles in one hand always have a large number of possible combinations, it usually cannot be easily calculated. To address this issue, several approaches have been presented, but they are very limited and always require too much computational time. Therefore, inspired by the advantages of DE, such as simplicity, easy implementation, strong robustness, and superior performance, we propose a novel discrete DE variant (NDDE) to compute the deficiency number of a Mahjong hand in this section. Specifically, the problem of computing the deficiency number is first converted to a simpler combinatorial optimization problem, and a new DE variant is presented for it by devising proper initialization, a mapping solution method, a repairing solution technique, a fitness evaluation approach, and mutation and crossover operations.

3.1. Simplified Problem of Deficiency Number

As stated in [10], the problem of computing the deficiency number is in fact an optimization problem and can be regarded as a combinatorial optimization problem to solve, i.e., finding one proper combination of the tiles that has minimal differences between it and its corresponding ideal winning hand. Recently, several approaches have been proposed to calculate the deficiency number of the tiles based on this [10–12]. However, there are often too many combinations in the search space, which might degrade the efficiency of these methods. To alleviate this demerit, we propose converting the problem of computing the deficiency number into a simpler one, where the dimension of the new search space is reduced to five. The details of the concrete processes are described in the following.

For one Mahjong hand H, we first search all its possible melds and pmelds and denote them by S. The reason for this is that the decomposition of a winning hand is constituted by four melds and one eye, and it can be completed by the predecomposition, which consists of melds and/or pmelds. Then, the problem of computing the deficiency number can be alternatively described as:

$$\min_{\pi \in \Pi} g(\pi) \tag{5}$$

where $\pi = (\pi(1), \pi(2), \pi(3), \pi(4), \pi(5))$ is the predecomposition of H with $\pi(i)$ $(1 \le i \le 5)$ being a meld, pair, or pchow from S or empty, Π is the set of all predecompositions of H, and $g(\pi)$ denotes the differences between π and its ideal decomposition, i.e., having $g(\pi) = cost(\pi)$, which will be further discussed in the next subsection.

According to the definition of Equation (5), all possible melds and pmelds of one Mahjong hand H must be found and contained in S, so as to ensure the completeness of all its predecompositions. Moreover, since the meld has the modes of chow and pong, the pmeld has the modes of pair and pchow, and the pchow always has two different modes, such as $(D_6 D_7)$ and $(D_6 D_8)$; then, S can be formed by considering each case above for each tile. Note that, to form a winning hand, the pchow must be able to constitute a chow, and thus, the number of tiles used to complete the pchow should be less than four in H. Thereby, for a Mahjong hand H, its related set S can be generated as follows. First, we find the distinct tiles of H and denote them by S_t, while initializing the set S to be empty. Then, for each tile $X_i \in S_t$, the following five cases are successively checked to create all possible melds and pmelds, which will be added into S.

Case 1. If $X_{i+1} \in H$ and the number of X_{i-1} or X_{i+2} in H is less than four, then add pchow $(X_i X_{i+1})$ into S;

Case 2. If $X_{i+2} \in H$ and the number of X_{i+1} in H is less than four, then add pchow $(X_i X_{i+2})$ into S;

Case 3. If $X_{i+1} \in H$ and $X_{i+2} \in H$, then add chow $(X_i X_{i+1} X_{i+2})$ into S;

Case 4. If the number of X_i in H is more than two, then add pair $(X_i X_i)$ into S;

Case 5. If the number of X_i in H is more than three, then add pong $(X_i X_i X_i)$ into S.

Particularly, Cases 1, 2, and 4 can provide all the possible pmelds, and Cases 3 and 5 can give all the possible melds involved in H.

In summary, from the above descriptions, all the possible melds and pmelds can be obtained for each Mahjong hand H, and then every predecomposition involved in it can be created by using S. Thus, the predecompositions obtained by the tree-based search algorithm will be contained in the search space Π of the new problem. Meanwhile, since the tree-based search algorithm is a full search approach, the legal solution for the new problem can also be obtained by it. So, the proposed operations cannot affect the deficiency number of one Mahjong hand, and the converted problem and its original one are equivalent. Moreover, it is easy to find that the dimension of this new problem is just five, and its search space is decided by the length of S. Additionally, the generation process of S is relatively cheap, and its size of S is often smaller; therefore, the new problem has a smaller search space and is easier to solve.

3.2. *Proposed NDDE Algorithm*

In this subsection, the special components of the NDDE algorithm shall be introduced, including the initialization, mapping solution method, repairing solution technique, fitness evaluation approach, and mutation and crossover operations.

3.2.1. Initialization of Population and Mapping Solution

According to the framework of DE, when DE is activated, an initial population with NP solutions will first be created. In this paper, since the objective function is a discrete one, the following special method is used to initialize the population P^0.

For convenience, we let N_S denote the size of the set S, each solution $\vec{x}_i^0$ denote a predecomposition π, and then have $\vec{x}_i^0 = (x_{i,1}^0, x_{i,2}^0, \ldots, x_{i,5}^0)$ with $i = 1, 2, \ldots, NP$. Specifically, for each element $x_{i,j}^0$ of $\vec{x}_i^0$, $j = 1, 2, \ldots, 5$, it will be created by

$$x_{i,j}^0 = randint(N_S), \tag{6}$$

where $randint(A)$ returns a random integer uniformly distributed in $[1, A]$.

Moreover, from Equation (6), one can easily find that each solution in P^0 is only represented by some integer values, and they are not compatible with the new problem described in Equation (5). Thus, a mapping method is further provided here for matching the solutions of the population with those of the new problem. In particular, for each solution $\vec{x}_i^0$ in P^0, its corresponding solution π_i^0 for the new problem can be obtained by

$$\pi_i^0 = (\pi_i^0(1), \pi_i^0(2), \pi_i^0(3), \pi_i^0(4), \pi_i^0(5)), \tag{7}$$

where $\pi_i^0(j)$ is the $x_{i,j}^0$-th element of S for $j = 1, 2, \ldots, 5$, which may be a meld or pmeld.

From the above descriptions, the solutions matching the DE algorithm and the new problem can all be obtained. Noticeably, the operations of initializing the population and mapping it are both simple and easy to implement. Thus, these operations will not add severe computational burdens.

3.2.2. Repairing Solution Technique

As described above, through the proposed mapping method, each individual in a population will generate a corresponding solution for the new problem. However, there is still a shortcoming in it, that is, the mapped solutions cannot be guaranteed to be feasible. In fact, for one Mahjong hand H and a mapped solution π_i, the number of some tiles in π_i may exceed that in H. Thereby, a repairing technique is also necessary to correct these infeasible solutions.

For the sake of saving the computational cost of the algorithm, the following simple repairing approach is used in this paper. For one Mahjong hand H and a mapped solution π_i, all distinct tiles of π_i are first found and recorded in one set, denoted by S_p. Then, for each tile X_i in S_p, we separately count its numbers in π_i and H, and if the number in π_i is larger than that in H, we gradually remove one meld or pmeld containing X_i from π_i until its number in π_i is less than or equal to that in H. At the same time, when one meld or pmeld is removed in π_i, its corresponding element in $\vec{x}_i^g$ will also be set to empty. Clearly, by this repairing method, each mapped solution will be changed to be feasible, and this correcting process is very easy to achieve.

3.2.3. Fitness Evaluation Approach

In order to determine which solutions will be selected in the next iteration, their fitness values should be evaluated as well. At each generation g, the fitness value of each individual $\vec{x}_i^g$ in population P^g is evaluated by the cost of its corresponding mapped solution π_i^g for the new problem. That is, we let $f(\vec{x}_i^g) = g(\pi_i^g)$ in this paper.

Concretely, for a mapped solution π^g, according to $g(\pi^g) = cost(\pi^g)$ and the definition of $cost(\pi^g)$, i.e., the number of missing tiles required to complete π^g into a decomposition, the fitness value of $\vec{x}_i^g$ can be calculated by

$$f(\vec{x}_i^g) = \begin{cases} 4 - m, & \text{if } m + n = 5 \text{ } and \text{ } p = 1, \\ 5 - m, & \text{if } m + n = 5 \text{ } and \text{ } p = 0, \\ 9 - 2 * m - n, & \text{if } m + n < 5. \end{cases} \tag{8}$$

Herein, m and n are the numbers of meld and pmeld in π_i^g, respectively, and p is the index of whether there is a pair in π_i^g. If there is a pair in π_i^g, then we have $p = 1$; otherwise, $p = 0$.

Moreover, it should be mentioned that there are still two other special cases in the evaluation of the solution. For one predecomposition π^g, the first case is that it has $m + n = 5$ and contains two identical pairs. In this case, one of the two pairs in π^g should have formed a meld, but the number of tiles in π^g will have increased to four. Thus, the actual cost should be added by one due to the invalid pmeld. Another case is that it has $m = 4$, $n = 0$, and the remaining two different tiles will have formed a pong in π^g. In this case, we further need to form a pair to complete π^g, but the remaining tiles cannot form a pair. Therefore, we need to change both of these two tiles to form a pair and the actual cost of π^g should be two.

For example, considering one Mahjong hand $H = (B_1B_2B_3B_3B_3B_3B_4B_8)(C_4C_5C_9)$ $(D_6D_7D_8)$ and its three predecompositions $\pi_1 = (B_1B_2B_3)(B_3B_4)(B_3B_3)(C_4C_5)(D_6D_7D_8)$, $\pi_2 = (B_1B_2B_3)(B_3B_3B_3)(C_4C_5)(D_6D_7D_8)$, and $\pi_3 = (B_2B_3B_4)(B_1B_3)(B_3B_3)(C_4C_5)(D_6D_7)$, by using the above evaluation approach, we can find that for π_1, it has $m = 2$, $n = 3$, and $p = 1$; thus, $g(\pi_1) = 2$. While for π_2, it has $m = 3$, $n = 1$, and $p = 0$; thus, $g(\pi_2) = 2$. For π_3, it has $m = 1$, $n = 4$, and $p = 1$; thus, $g(\pi_2) = 3$.

In conclusion, from the above descriptions, the proposed fitness evaluation method can accurately assess the quality of each solution in the population.

3.2.4. Mutation and Crossover Operators

To fully search the decision space and ensure the computational efficiency of the algorithm, the mutation operator "DE/rand/1" and the binomial crossover operation are enhanced and employed to generate the mutant and trial individual, respectively, in the proposed NDDE algorithm.

Especially considering the fact that the search space involved in the NDDE algorithm is discrete and the lengths of different solutions may be various, a discrete version of "DE/rand/1" is devised and employed in this paper. In detail, for each individual $\vec{x}_i^g$ at g generation, its mutant individual $\vec{v}_i^g = (v_{i,1}^g, v_{i,2}^g, \ldots, v_{i,5}^g)$ can be generated by

$$v_{i,j}^g = \begin{cases} round(x_{r1,j}^g + F*(x_{r2,j}^g - x_{r3,j}^g)), & \text{if } x_{r1,j}^g \neq \varnothing \text{ and } x_{r2,j}^g \neq \varnothing \text{ and} x_{r3,j}^g \neq \varnothing, \\ randint(N_S), & \text{otherwise} \end{cases} \tag{9}$$

where $j = 1, 2, \ldots, 5$, r_1, r_2, and r_3 are three random integers in $[1, NP]$ with $r_1 \neq r_2 \neq r_3 \neq i$, F is a scaling factor, $round(A)$ denotes the nearest integer around A. In particular, we set $F = 0.5$ in this paper.

Similarly, based on the property of varying variables of individuals and to make full use of the information of the target individual, the following modified binomial crossover operation is developed and used to generate the trial individuals. Specifically, for each individual $\vec{x}_i^g$ and its mutant individual $\vec{v}_i^g$, the corresponding trial individual $\vec{u}_i^g = (u_{i,1}^g, u_{i,2}^g, \ldots, u_{i,5}^g)$ is obtained by

$$u_{i,j}^g = \begin{cases} x_{i,j}^g, & \text{if } x_{i,j}^g \neq \varnothing \text{ and } rand \leq Cr, \\ v_{i,j}^g, & \text{otherwise} \end{cases} \tag{10}$$

where $Cr \in (0, 1)$ is the crossover rate, and especially, we set $Cr = 0.5$ here.

As described above, the proposed mutation and crossover operator can broadly search the search space, fully utilize the acquired information, and have a simple implementing process. Consequently, they are capable of boosting the search ability of the algorithm for finding the deficiency number of one Mahjong hand.

Finally, after generating the trial individual for each target one, the population will be updated by comparing them based on their fitness values, and the best one among them will enter the new population. To achieve this process, the greedy selection strategy (see Equation (4)) is utilized in this paper. Overall, by integrating the proposed initialization, mapping solution method, repairing solution technique, fitness evaluation approach, and mutation and crossover operations, the framework of the proposed NDDE algorithm is shown in Algorithm 1. To improve the understanding of this paper, a flowchart of the proposed approach for computing the deficiency number is provided in Figure 1, where G and G_{max} denote the current number and maximum number of iterations, respectively.

It should be pointed out that, unlike the existing approaches for obtaining the deficiency number in [10–12], the proposed method simplifies the original problem of calculating the deficiency number and makes full use of the benefits of DE to improve the computational efficiency. Specifically, by previously finding all possible melds, pmelds, and pairs contained in one hand and constructing the form of the solution based on the structure of the decomposition, the problem of calculating the deficiency number is converted to a more simple combinatorial optimization problem. Meanwhile, according to the properties of discrete and varying variables of the new problem, a proper initialization, mapping solution method, repairing solution technique, fitness evaluation approach, and mutation and crossover operations are separately developed, and then a novel discrete DE algorithm is proposed. Thereby, the proposed method can effectively and efficiently compute the deficiency number of one Mahjong hand. It should also be mentioned that, for the hands with a larger deficiency number, the proposed approach might be less efficient than the tree-based deterministic algorithms. This is because the ones with a larger deficiency number always just contain a few melds, pmelds, and/or pairs, which might lead to few child nodes for each search step in the deterministic methods and then reduce the search cost, while the stochastic algorithms need to conduct the predefined searches for each hand at all times. Moreover, compared to the general discrete and/or combinatorial optimization problems, the simplified problem involved in this paper has a special characteristic that its feasible solutions may have various lengths. So, the previous discrete DE versions are not able to directly solve this problem. Meanwhile, for solving the simplified problem, other metaheuristic algorithms can be alternatively adopted instead of DE by designing some proper operations, which we will further study in our future work.

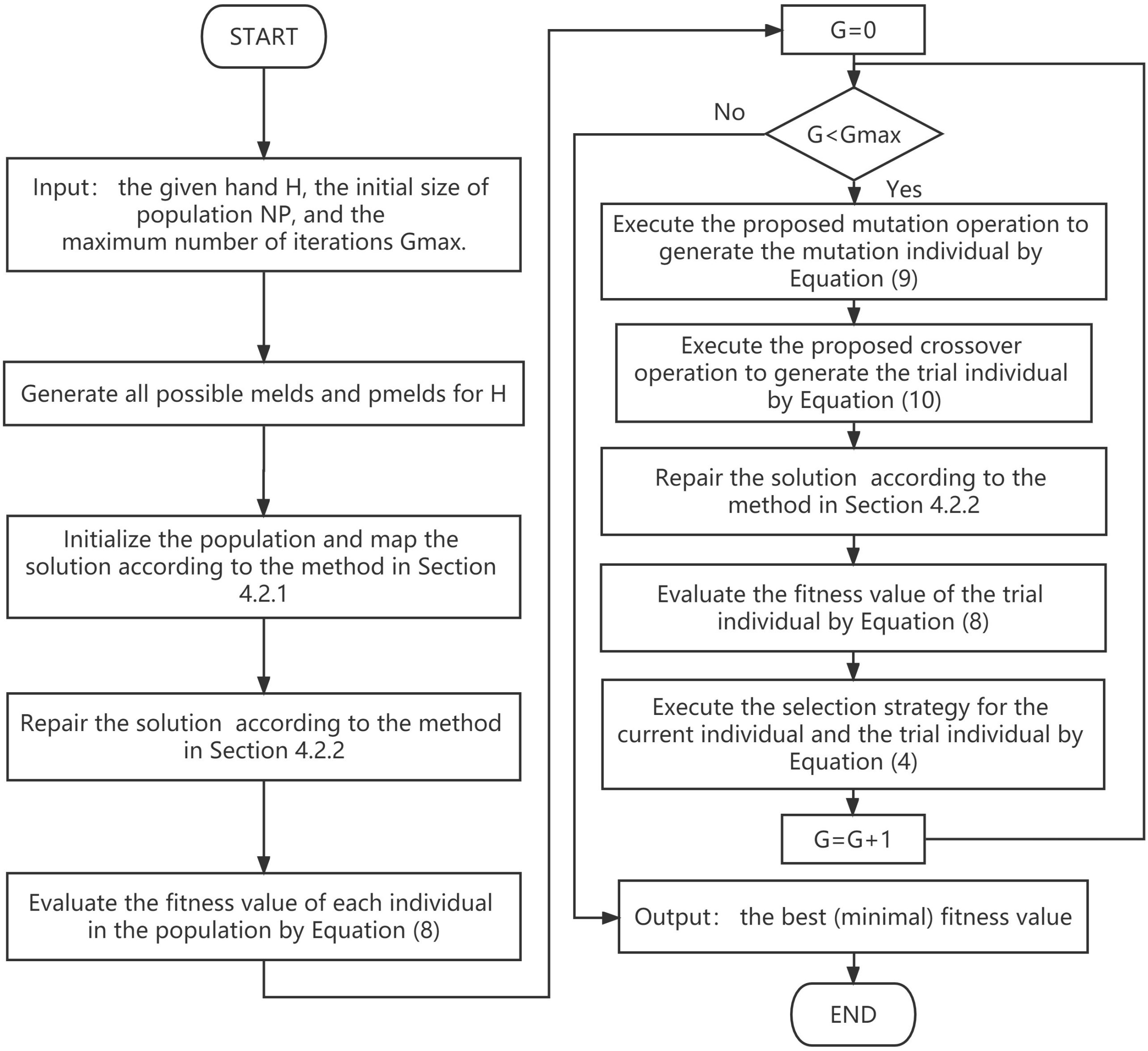


Figure 1. The flowchart of the proposed method in this paper for computing deficiency number.

Algorithm 1 (The framework of the proposed NDDE algorithm).

Input: the given hand H, the initial size of population NP, the maximum number of iterations G_{max}.
Generate the set S consisting of all possible melds and pmleds for H, and calculate N_S.
Set the current generation $g = 0$.
Initialize the population P^g by Equation (6), map and repair each individual in P^g by Equation (7) and the proposed repairing technique described in Section 4.2.2, respectively.
Evaluate the fitness value of each individual in P^g by Equation (8).
while $g \leq G_{max}$ **do**
  for $i = 1 : NP$ **do**
    Execute the proposed mutation operation to generate $\vec{v}_i^g$ by Equation (9);
    Execute the proposed crossover operation to generate $\vec{u}_i^g$ by Equation (10);
    Repair $\vec{u}_i^g$ by Equation (7);
    Evaluate the fitness value of $\vec{u}_i^g$ by Equation (8);
    Execute the selection strategy for the current individual $\vec{v}_i^g$ and its trial individual $\vec{u}_i^g$ by Equation (4);
  end for
  Set $g = g + 1$;
end while
Output: the best (minimal) fitness value.

4. Experimental Analyses

In this part, the performance of the proposed NDDE algorithm is evaluated by conducting a series of experiments on a large number of various, randomly generated test hands, including 118,800 hands with one type, 100,000 hands with two types, and 100,000 hands with three types. Meanwhile, the influence analyses of the parameters involved in the NDDE algorithm are investigated in terms of both search accuracy and running time, and the tree search algorithm (TSA) in [10] and three other metaheuristic algorithms, including PSO [15], GA [13], and TLBO [20], are also compared with the NDDE algorithm. Finally, the effectiveness of the NDDE algorithm is further demonstrated in a large number of Mahjong game battles.

In these experiments, the performance of each algorithm is measured by the accuracy rate and average running time. Moreover, in order to make fair comparisons and obtain statistical conclusions, each algorithm is run 30 times independently for each hand, and three widely used statistical tests, including the t test [53], Wilcoxon rank sum test [54], and Friedman test [55], are further adopted to distinguish the differences between the NDDE algorithm and each compared method. All algorithms are all implemented in Python 3.0 on a personal laptop with Intel i7-6700 CPU and 16 GB RAM.

4.1. Influence Analyses of NP and G_{max}

Herein, the influences of NP and G_{max} on the performance of the NDDE algorithm are analyzed. As stated in Algorithm 1, NP determines the number of solutions created at each iteration, while G_{max} decides the total iterations of the NDDE algorithm. Thus, various values for them might cause different performances of the NDDE algorithm. Specifically, in order to show the influence of each parameter, the full factorial design (FFD) [56] is used here, and NP and G_{max} are set to four and six different values, respectively, i.e., $NP \in \{10, 20, 30, 50\}$ and $G_{max} \in \{10, 30, 50, 100, 200, 500\}$. Tables 1 and 2 list the accuracy rate and average running time, respectively, of the NDDE algorithm on three kinds of test hands with one run. Note that when the actual deficiency number for a hand is found, the proposed NDDE algorithm will be terminated, and the corresponding running time is used just to measure its performance.

Table 1. Accuracy rates of NDDE algorithm with various NP and G_{max} on three kinds of test hands.

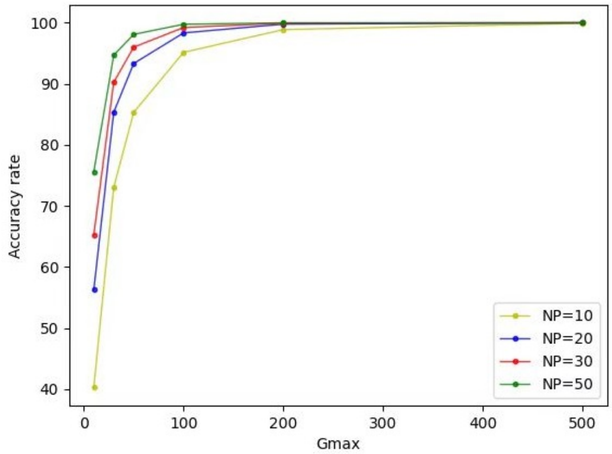
NP \ G_{max}		10	30	50	100	200	500
One type	10	40.246%	73.034%	85.332%	95.154%	98.879%	99.902%
	20	56.237%	85.348%	93.325%	98.337%	99.774%	99.988%
	30	65.250%	90.253%	96.000%	99.219%	99.927%	99.997%
	50	75.591%	94.689%	98.093%	99.751%	99.988%	100%
Two types	10	73.177%	92.509%	96.929%	99.407%	99.915%	99.998%
	20	84.632%	97.065%	99.044%	99.878%	99.992%	100%
	30	89.621%	98.464%	99.607%	99.955%	99.998%	100%
	50	93.979%	99.368%	99.873%	99.994%	100%	100%
Three types	10	87.124%	97.477%	99.146%	99.878%	99.982%	100%
	20	93.707%	99.154%	99.797%	99.977%	100%	100%
	30	96.130%	99.613%	99.930%	99.994%	100%	100%
	50	98.066%	99.863%	99.990%	99.999%	100%	100%

From Table 1, it can be seen that the accuracy rate of the NDDE algorithm is closely related to the values of NP and G_{max}, and gradually improves with their increase in all cases. Specifically, for the hands with one type, the accuracy rate of the NDDE algorithm exceeds 90% when G_{max} = 30 and NP = 30 and 50, G_{max} = 50 and NP = 20, 30, and 50, and every value for NP with G_{max} = 100, 200, and 500. The accuracy rate of the NDDE algorithm is 100% when $G_{max} = 500$ and $NP = 50$. Moreover, for the hands with two types, the accuracy rate of the NDDE algorithm exceeds 90% except when G_{max} = 10 and NP = 10, 20, and 30, while it reaches 100% when G_{max} = 200, NP = 50 and G_{max} = 500, NP = 20, 30, and 50. Moreover, for the hands with three types, the accuracy rate of the NDDE algorithm exceeds 90% except when G_{max} = 10 and NP = 10, while it reaches 100% when $G_{max} = 200$ and NP = 20, 30, and 50, and every value for NP with $G_{max} = 500$. In addition, from Table 2, one can further find that the average running time of the NDDE algorithm is also dependent on the values of NP and G_{max}, and it always gradually increases with their increase in all cases. Specifically, when the types of the hands increase, the average running time of the NDDE algorithm on them gradually decreases. The reason for this might be that as the types of the hands increase, the corresponding search space will be reduced. Hence, the proposed NDDE algorithm can always obtain the actual deficiency number for all hands with certain search costs.

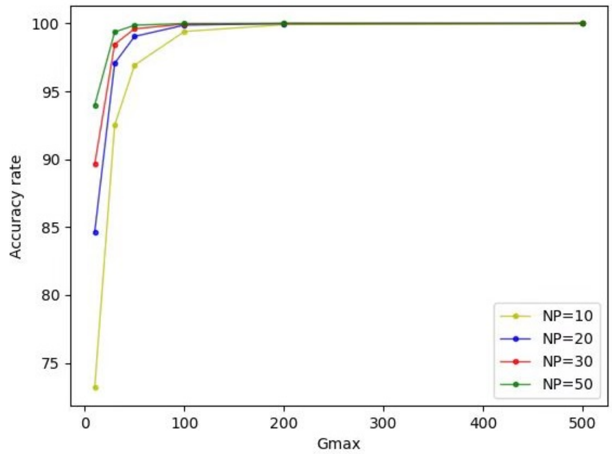
Table 2. Average running time of NDDE algorithm with various NP and G_{max} on three kinds of test hands.

NP \ G_{max}		10	30	50	100	200	500
One type	10	4.97×10^{-3} s	1.01×10^{-2} s	1.20×10^{-2} s	1.48×10^{-2} s	1.56×10^{-2} s	1.72×10^{-2} s
	20	8.13×10^{-3} s	1.40×10^{-2} s	1.64×10^{-2} s	1.85×10^{-2} s	1.88×10^{-2} s	1.99×10^{-2} s
	30	1.11×10^{-2} s	1.81×10^{-2} s	1.98×10^{-2} s	2.07×10^{-2} s	2.22×10^{-2} s	2.11×10^{-2} s
	50	1.54×10^{-2} s	2.21×10^{-2} s	2.40×10^{-2} s	2.44×10^{-2} s	2.55×10^{-2} s	2.52×10^{-2} s
Two types	10	3.28×10^{-3} s	4.96×10^{-3} s	5.49×10^{-3} s	5.90×10^{-3} s	6.04×10^{-3} s	6.22×10^{-3} s
	20	4.94×10^{-3} s	6.39×10^{-3} s	7.22×10^{-3} s	7.21×10^{-3} s	7.22×10^{-3} s	7.36×10^{-3} s
	30	6.08×10^{-3} s	7.39×10^{-3} s	7.68×10^{-3} s	7.82×10^{-3} s	8.06×10^{-3} s	7.92×10^{-3} s
	50	7.95×10^{-3} s	9.00×10^{-3} s	9.28×10^{-3} s	9.61×10^{-3} s	9.74×10^{-3} s	9.57×10^{-3} s
Three types	10	2.31×10^{-3} s	3.08×10^{-3} s	3.12×10^{-3} s	3.22×10^{-3} s	3.25×10^{-3} s	3.27×10^{-3} s
	20	3.07×10^{-3} s	3.67×10^{-3} s	3.79×10^{-3} s	3.84×10^{-3} s	3.84×10^{-3} s	3.86×10^{-3} s
	30	3.73×10^{-3} s	4.24×10^{-3} s	4.34×10^{-3} s	4.35×10^{-3} s	4.38×10^{-3} s	4.34×10^{-3} s
	50	4.83×10^{-3} s	5.23×10^{-3} s	5.21×10^{-3} s	5.28×10^{-3} s	5.21×10^{-3} s	5.23×10^{-3} s

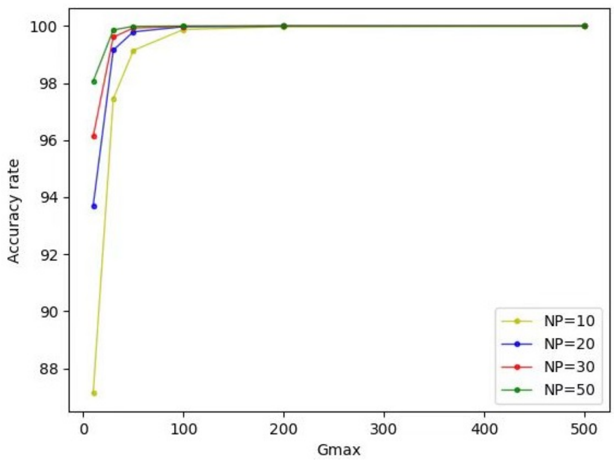
For the sake of clarity, Figures 2 and 3 further depict the accuracy rate and average running time of the NDDE algorithm on all kinds of hands. From Figures 2 and 3, it can easily be seen that whenever either NP or G_{max} increase, the accuracy rate of the NDDE algorithm improves for each type of hand. Meanwhile, with the increase in NP, the average running time of the NDDE algorithm increases in each case, while the NDDE algorithm has a minimal average running time on the hands with three types. Thus, it is essential to set suitable NP and G_{max} in the NDDE algorithm, and we let $NP = 20$ and $G_{max} = 50$ in the following experiments due to its promising performance in terms of both accuracy rate and average running time.



(**a**)



(**b**)



(**c**)

Figure 2. Accuracy rates of NDDE algorithm with various NP and G_{max} on three kinds of test hands. (**a**) Hands with one type, (**b**) hands with two types, and (**c**) hands with three types.

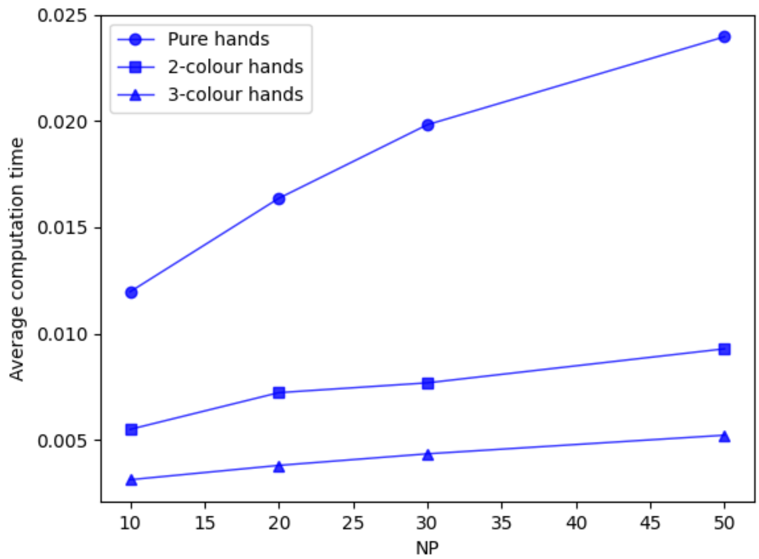


Figure 3. Average running time of NDDE algorithm with various NP and $G_{max} = 50$ on three kinds of test hands.

4.2. Comparisons and Discussions

In this subsection, in order to verify the performance of the NDDE algorithm, one typical deterministic method, namely TSA [10], and three other famous metaheuristic algorithms, including PSO [15], GA [13], and TLBO [20], are compared with it on all the above cases of hands. Specifically, to persuasively estimate the performance of these methods, each approach is run 30 times independently for each hand, and the average accuracy rate and running time of 30 runs on each kind of hand are employed to measure its performance. Moreover, t tests [53], Wilcoxon rank sum tests [54], and Friedman tests [55] are also utilized to show the differences between their performances.

It should be mentioned that PSO [15] is a famous swarm intelligent optimization algorithm, GA [13] is a typical approach belonging to evolutionary computation, and TLBO [20] is a promising metaheuristic method inspired by human activities. Meanwhile, the proposed NDDE algorithm is developed based on just the basic framework of DE. So, these methods are very representative and suitable and thus chosen as the compared ones here.

4.2.1. Comparisons of NDDE Algorithm with TSA

First, one typical deterministic method, namely TSA [10], is compared with the NDDE algorithm on all three kinds of hands. To clearly demonstrate the performance of the NDDE algorithm, its two versions, named NDDE$_1$ and NDDE$_2$, where NP and G_{max} are set to 20 and 50 and 50 and 500, respectively, are simultaneously employed here to compare with TSA. Tables 3 and 4 provide their average and statistical results of 30 runs on each kind of hand in terms of the accuracy rate and running time, respectively. Herein, p_t-value and p_w-value denote the p-values of the t test [53] and Wilcoxon rank sum test [54], respectively (the same below).

From Tables 3 and 4, it can be seen that NDDE$_1$ has the worst results among them in all cases, and NDDE$_2$ and TSA each obtain the actual deficiency number for all hands. Meanwhile, with respect to the average running time, TSA takes the longest time in each case, and NDDE$_1$ takes less time than NDDE$_2$. Moreover, according to the results of the t test and Wilcoxon rank sum test reported in both Tables 3 and 4, NDDE$_1$ has significant differences compared with TSA, and NDDE$_1$ and NDDE$_2$ are both significantly faster than TSA in all cases. Thus, the proposed NDDE is more effective and efficient than TSA for the deficiency number of one Mahjong hand.

Table 3. The average and statistical results of TSA, $NDDE_1$, and $NDDE_2$ on three kinds of hands in terms of accuracy rate.

Hands	Methods	Best Result	Worst Result	Median Result	Mean Result	Standard Deviation	p_t-Value	p_w-Value
One type	TSA	100%	100%	100%	100%	0.00	- -	- -
	$NDDE_1$	93.191%	93.503%	93.301%	93.308%	7.07×10^{-4}	<0.0001	<0.0001
	$NDDE_2$	99.999%	100%	100%	100%	3.19×10^{-6}	0.0192	0.0214
Two types	TSA	100%	100%	100%	100%	0.00	- -	- -
	$NDDE_1$	99.013%	99.129%	99.070%	99.070%	3.21×10^{-4}	<0.0001	<0.0001
	$NDDE_2$	100%	100%	100%	100%	0.00	1.0000	1.0000
Three types	TSA	100%	100%	100%	100%	0.00	- -	- -
	$NDDE_1$	99.760%	99.808%	99.788%	99.785%	1.21×10^{-4}	<0.0001	<0.0001
	$NDDE_2$	100%	100%	100%	100%	0.00	1.0000	1.0000

Table 4. The average and statistical results of TSA, $NDDE_1$, and $NDDE_2$ on three kinds of hands in terms of running time.

Hands	Methods	Best Result (s)	Worst Result (s)	Median Result (s)	Mean Result (s)	Standard Deviation	p_t-Value	p_w-Value
One type	TSA	1.88×10^{-1}	1.89×10^{-1}	1.88×10^{-1}	1.88×10^{-1}	1.66×10^{-4}	- -	- -
	$NDDE_1$	1.57×10^{-2}	1.59×10^{-2}	1.58×10^{-2}	1.58×10^{-2}	3.87×10^{-5}	<0.0001	0.0004
	$NDDE_2$	2.40×10^{-2}	2.43×10^{-2}	2.41×10^{-2}	2.41×10^{-2}	8.74×10^{-5}	<0.0001	0.0004
Two types	TSA	4.32×10^{-2}	4.41×10^{-2}	4.36×10^{-2}	4.36×10^{-2}	3.48×10^{-4}	- -	- -
	$NDDE_1$	7.05×10^{-3}	7.48×10^{-3}	7.09×10^{-3}	7.12×10^{-3}	9.04×10^{-5}	<0.0001	0.0004
	$NDDE_2$	7.95×10^{-3}	8.94×10^{-3}	8.03×10^{-3}	8.10×10^{-3}	2.10×10^{-4}	<0.0001	0.0004
Three types	TSA	1.72×10^{-2}	1.72×10^{-2}	1.72×10^{-2}	1.72×10^{-2}	4.46×10^{-6}	- -	- -
	$NDDE_1$	3.85×10^{-3}	4.91×10^{-3}	3.94×10^{-3}	3.97×10^{-3}	1.86×10^{-4}	<0.0001	0.0004
	$NDDE_2$	5.18×10^{-3}	5.30×10^{-3}	5.21×10^{-3}	5.21×10^{-3}	2.59×10^{-5}	<0.0001	0.0004

4.2.2. Comparisons of NDDE Algorithm with Three Other Famous Metaheuristic Algorithms

To further demonstrate the benefit of the NDDE algorithm, three other famous stochastic intelligent algorithms, including PSO [15], GA [13], and TLBO [20], are also compared with it in all cases of hands above. Specifically, in these chosen compared methods, the same mapping, repairing, and evaluation methods as in the NDDE algorithm are adopted, and the size of the population and the maximum number of iterations are also set to 20 and 50, which is consistent with the setting of the NDDE algorithm. Moreover, to show the differences between these compared methods and the NDDE algorithm, the three statistical tests above are further adopted to give statistical conclusions. Tables 5 and 6 list the average and statistical results of 30 runs on each kind of hand in terms of the accuracy rate and running time, respectively, and Table 7 reports their final comparison results based on the Friedman test [55].

As seen from Tables 5 and 6, the NDDE algorithm has a better accuracy rate than PSO, GA, and TLBO in all cases, and there are significant differences between them and the NDDE algorithm according to both the t test and Wilcoxon rank sum test. Meanwhile, in terms of running time, the NDDE algorithm also has the least time on all kinds of hands, and significantly performs best based on the statistical results. The reason for this might be because GA needs to calculate the selection probability for each individual at each generation, PSO always needs to record and update the personal best individual for each solution, and TLBO has to additionally compute the mean point of the whole population and compare the two target individuals based on their performances to determine the search direction. So, the NDDE algorithm has a more efficient search procedure than the others. Moreover, from Table 7, according to the Friedman test, the NDDE algorithm has the top performance among them on all three kinds of hands in terms of both the accuracy rate and running time. Thereby, the NDDE algorithm is the most promising solver for computing the deficiency number.

Table 5. The average and statistical results of NDDE algorithm, PSO, GA, and TLBO on three kinds of hands in terms of accuracy rate.

Hands	Methods	Best Result	Worst Result	Median Result	Mean Result	Standard Deviation	p_t-Value	p_w-Value
One type	PSO	80.900%	81.200%	81.100%	81.100%	6.05×10^{-4}	<0.0001	<0.0001
	GA	42.600%	43.000%	42.800%	42.800%	1.08×10^{-3}	<0.0001	<0.0001
	TLBO	67.300%	67.700%	67.500%	67.500%	8.29×10^{-4}	<0.0001	<0.0001
	NDDE	93.200%	93.500%	93.300%	93.300%	7.07×10^{-4}	- -	- -
Two types	PSO	94.300%	94.500%	94.400%	94.400%	5.71×10^{-4}	<0.0001	<0.0001
	GA	72.500%	72.900%	72.700%	72.700%	9.47×10^{-4}	<0.0001	<0.0001
	TLBO	91.100%	91.400%	91.300%	91.300%	6.50×10^{-4}	<0.0001	<0.0001
	NDDE	99.000%	99.100%	99.100%	99.100%	3.21×10^{-4}	- -	- -
Three types	PSO	97.800%	97.900%	97.800%	97.800%	3.09×10^{-4}	<0.0001	<0.0001
	GA	86.300%	86.700%	86.500%	86.500%	9.58×10^{-4}	<0.0001	<0.0001
	TLBO	97.000%	97.100%	97.100%	97.100%	3.28×10^{-4}	<0.0001	<0.0001
	NDDE	99.800%	99.800%	99.800%	99.800%	1.21×10^{-4}	- -	- -

Table 6. The average and statistical results of NDDE algorithm, PSO, GA, and TLBO on three kinds of hands in terms of running time.

Hands	Methods	Best Result (s)	Worst Result (s)	Median Result (s)	Mean Result (s)	Standard Deviation	p_t-Value	p_w-Value
One type	PSO	2.09×10^{-2}	2.20×10^{-2}	2.10×10^{-2}	2.10×10^{-2}	1.85×10^{-4}	<0.0001	<0.0001
	GA	2.85×10^{-2}	2.93×10^{-2}	2.86×10^{-2}	2.88×10^{-2}	2.41×10^{-4}	<0.0001	<0.0001
	TLBO	2.89×10^{-2}	3.52×10^{-2}	3.00×10^{-2}	3.03×10^{-2}	1.36×10^{-3}	<0.0001	<0.0001
	NDDE	1.57×10^{-2}	1.59×10^{-2}	1.58×10^{-2}	1.58×10^{-2}	3.87×10^{-5}	- -	- -
Two types	PSO	1.04×10^{-2}	1.05×10^{-2}	1.05×10^{-2}	1.05×10^{-2}	3.46×10^{-5}	<0.0001	<0.0001
	GA	1.68×10^{-2}	1.78×10^{-2}	1.70×10^{-2}	1.71×10^{-2}	3.00×10^{-4}	<0.0001	<0.0001
	TLBO	1.38×10^{-2}	1.55×10^{-2}	1.44×10^{-2}	1.44×10^{-2}	3.94×10^{-4}	<0.0001	<0.0001
	NDDE	7.05×10^{-3}	7.48×10^{-3}	7.09×10^{-3}	7.12×10^{-3}	9.04×10^{-5}	- -	- -
Three types	PSO	5.87×10^{-3}	6.17×10^{-3}	6.09×10^{-3}	6.08×10^{-3}	6.18×10^{-5}	<0.0001	<0.0001
	GA	9.68×10^{-3}	1.07×10^{-2}	9.81×10^{-3}	9.91×10^{-3}	2.15×10^{-4}	<0.0001	<0.0001
	TLBO	7.77×10^{-3}	1.09×10^{-2}	7.98×10^{-3}	8.10×10^{-3}	5.64×10^{-4}	<0.0001	<0.0001
	NDDE	3.85×10^{-3}	4.91×10^{-3}	3.94×10^{-3}	3.97×10^{-3}	1.86×10^{-4}	- -	- -

Table 7. The final comparison results of NDDE algorithm, PSO, GA, and TLBO on all kinds of hands according to Friedman test.

Algorithm	Accuracy Rate				Running Time			
	NDDE	PSO	GA	TLBO	NDDE	PSO	GA	TLBO
Rank	1.00	2.00	3.67	3.33	1.00	2.00	4.00	3.00

Furthermore, in order to clearly illustrate the performance of the NDDE algorithm, the convergence curves of the NDDE algorithm, PSO, GA, and TLBO are also depicted here on six different hands, including $H_1 = (B_4B_4B_6B_6B_6B_7B_7B_7B_7B_8B_9B_9B_9B_9)$, $H_2 = (B_3B_5B_5B_5B_5B_6B_6B_6B_7B_7B_8B_8B_9B_9)$, $H_3 = (B_1B_2B_4B_5B_5)(C_2C_3C_3C_3C_4C_4C_5C_7C_7)$, $H_4 = (B_4)(C_1C_1C_3C_4C_4C_4C_4C_5C_7C_8C_8C_9C_9)$, $H_5 = (B_4B_6)(C_5C_7C_8C_9)(D_1D_1D_2D_2D_3D_7D_7D_8)$, and $H_6 = (B_3)(C_9)(D_1D_4D_5D_6D_6D_6D_7D_7D_8D_8D_9D_9)$. Herein, H_1 and H_2 have just one color, H_3 and H_4 have two colors, and H_5 and H_6 have three colors. From Figure 4, one can easily find that the NDDE algorithm always has a better convergence performance than PSO, GA, and TLBO on each hand. Therefore, the NDDE algorithm has a more promising performance.

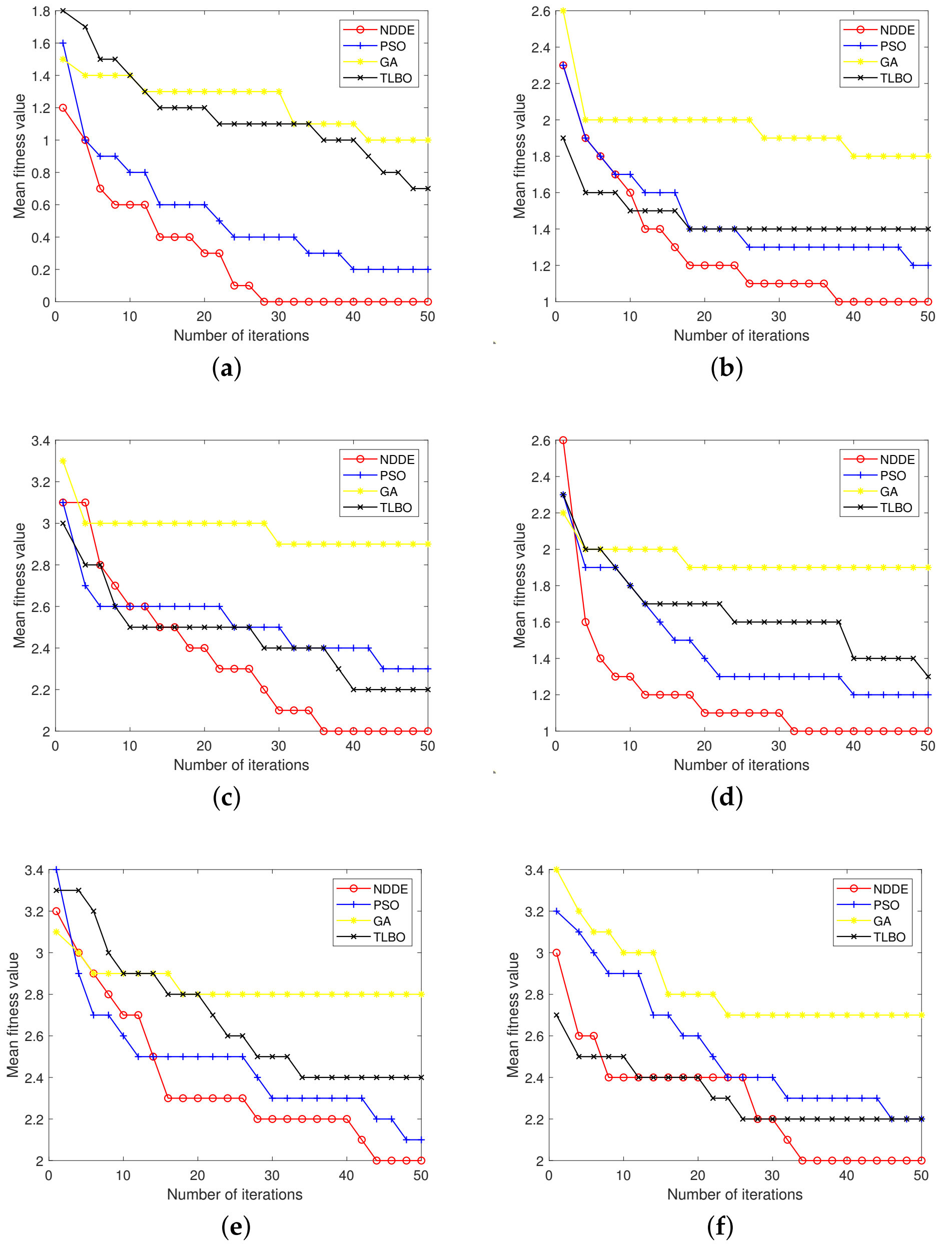


Figure 4. Convergence curves of NDDE algorithm and PSO, GA, and TLBO on six test hands. (**a**) H_1, (**b**) H_2, (**c**) H_3, (**d**) H_4, (**e**) H_5, and (**f**) H_6.

4.3. Effectiveness of NNDE on Mahjong Game Battles

In this part, the practicality of the NDDE algorithm is further evaluated by comparing it with the tree-based search method (TSA) [10] in a Mahjong battle with four players. In this test, 1000 randomly generated states of Mahjong are employed, where all players have drawn their hands and the order of tiles on the wall is fixed, and for each Mahjong game, two rounds are played. Moreover, all players adopt the same strategy to make the decisions for each action, such as pong, chow, and kong [10], except for the method employed to calculate the deficiency number. Specifically, for each state of the game, player 1 and player 3 use the NDDE algorithm to compute the deficiency number, while player 2 and player 4 adopt TSA in the first round. In contrast, player 1 and player 3 use TSA to compute the deficiency number, while player 2 and player 4 adopt the NDDE algorithm in the second round. The sum of the scores obtained by player 1 and player 3 in the first round and by player 2 and player 4 in the second round is recorded as the final score to evaluate the effectiveness of the NDDE algorithm. Herein, we set the basic score in game as 1, and let $NP = 20$ and $G_{max} = 50$ in the NDDE algorithm. After conducting these 1000 different games, the final score of the NDDE algorithm on them is -1.173. Importantly, it should be mentioned that the NDDE algorithm with $NP = 20$ and $G_{max} = 50$ has accuracy rates of 93.325%, 99.044%, and 99.797% on the hands with one, two, and three types, respectively, which can be found in Table 1. This result means that the NDDE algorithm has almost the same performance as TSA in the real battles. Thus, the NDDE algorithm is a promising approach for calculating the deficiency number of a Mahjong hand.

5. Conclusions

In this paper, a novel DE-based approach was presented to calculate the deficiency number of one Mahjong hand, which plays an important role in Mahjong and is helpful for boosting its AI development. Concretely, in order to decrease the difficulty of computing the deficiency number, some pretreatment mechanisms were first presented to convert the original problem into a simpler combinatorial optimization one, where the dimension of the search space of the new problem was reduced to five, and the feasible solutions might have various lengths. Meanwhile, inspired by the benefits of DE, such as simplicity, ease of implementation, a strong robustness, and a superior performance, a novel discrete DE (NDDE) variant was specially developed for solving this new problem by devising proper initialization, a mapping solution method, a repairing solution technique, a fitness evaluation approach, and mutation and crossover operations. Compared to the existing methods for calculating the deficiency number, where the full searches are all implicit in them, thus being very costly, the proposed algorithm employed the framework of the stochastic intelligent approach, and the problem of calculating the deficiency number was converted into a simpler one to solve in this paper. Thereby, the proposed approach is capable of more effectively and efficiently computing the deficiency number of one Mahjong hand. Finally, the performance of the proposed algorithm was evaluated by comparing with the tree search algorithm and three other kinds of metaheuristic methods on a large number of various test cases, and the sensitivity of the parameters involved in the NDDE algorithm was also investigated. The experimental results indicated that the proposed algorithm is more efficient and promising.

It should also be mentioned that this paper only adopted the framework of DE in the design of the algorithm due to its previous superior practical experiences, and the most simple version of DE only was used. Therefore, in our future work, we will focus on devising other solvers for calculating the deficiency number based on other metaheuristic algorithms, the existing enhanced DE variants, and discrete DE versions. Meanwhile, we will also focus on designing a hybrid method by properly integrating the merits of both the metaheuristic methods and the deterministic ones for calculating the deficiency number of a Mahjong hand.

Author Contributions: Conceptualization, X.Y. and Y.L.; methodology, Y.L.; software, X.Y.; validation, X.Y. and Y.L.; formal analysis, X.Y.; investigation, X.Y.; resources, Y.L.; data curation, X.Y.; writing—original draft preparation, X.Y.; writing—review and editing, X.Y.; visualization, X.Y.; supervision, Y.L.; project administration, Y.L.; funding acquisition, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China No. 11671244 and 12071271.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Origins of Mahjong. Available online: http://www.mahjongsets.co.uk/origins-mahjong.html (accessed on 15 December 2022).
2. Wikipedia. Mahjong. Available online: https://en.wikipedia.org/wiki/Mahjong (accessed on 15 December 2022).
3. Tang, J. Designing an Anti-swindle Mahjong Leisure Prototype System using RFID and ontology theory. *J. Netw. Comput. Appl.* **2014**, *39*, 292–301. [CrossRef]
4. Silver, D. Technical Perspective: Solving Imperfect Information Games. *Commun. ACM* **2017**, *60*, 80–80. [CrossRef]
5. Kurita, M.; Hoki, K. Method for Constructing Artificial Intelligence Player with Abstractions to Markov Decision Processes in Multiplayer Game of Mahjong. *IEEE Trans. Games* **2021**, *13*, 99–110. [CrossRef]
6. Wang, M.; Yan, T.; Luo, M.; Huang, W. A novel deep residual network-based incomplete information competition strategy for four-players Mahjong games. *Multimed. Tools Appl.* **2019**, *78*, 23443–23467. [CrossRef]
7. Gao, S.; Okuya, F.; Kawahara, Y.; Tsuruoka, Y. Building a Computer Mahjong Player via Deep Convolutional Neural Networks. *arXiv* **2019**, arXiv:1906.02146.
8. Gao, S.; Li, S. Bloody Mahjong playing strategy based on the integration of deep learning and XGBoost. *CAAI Trans. Intell. Technol.* **2022**, *7*, 95–106. [CrossRef]
9. Zheng, Y.; Li, S. A Review of Mahjong AI Research. In Proceedings of the RICAI 2020: 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence, Shanghai, China, 17–19 October 2020; pp. 345–349.
10. Li, S.; Yan, X. Let's Play Mahjong! *arXiv* **2019**, arXiv:1903.03294.
11. Wang, Q.; Li, Y.; Chen, X. A Mahjong-Strategy based on Weighted Restarting Automata. In Proceedings of the MLMI '20: 2020 the 3rd International Conference on Machine Learning and Machine Intelligence, Hangzhou, China, 18–20 September 2020.
12. Wang, Q.; Zhou, Y.; Zhu, D.; Li, Y. A new approach to compute deficiency number of Mahjong configurations. *Entertain. Comput.* **2022**, *43*, 100509. [CrossRef]
13. Holland, J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1975.
14. Storn, R.; Price, K. Differential evolution-a simple and efficient heuristic for global optimization over continuous space. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
15. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 1942–1948.
16. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
17. Eskandar, H.; Sadollah, A.; Bahreininejad, A.; Hamdi, M. Water cycle algorithm-a novel metaheuristic optimization method for solving constrained engineering optimization problems. *Comput. Struct.* **2012**, *110–111*, 151–166. [CrossRef]
18. Jain, M.; Singh, V.; Rani, A. A novel nature-inspired algorithm for optimization: Squirrel search algorithm. *Swarm Evol. Comput.* **2019**, *44*, 148–175. [CrossRef]
19. Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. GSA: A gravitational search algorithm. *Inf. Sci.* **2009**, *179*, 2232–2248. [CrossRef]
20. Rao, R.V.; Savsani, V.J.; Vakharia, D.P. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Comput.-Aided Des.* **2011**, *43*, 303–315. [CrossRef]
21. Mohamed, A.W.; Hadi, A.A.; Mohamed, A.K. Gaining-sharing knowledge based algorithm for solving optimization problems: A novel nature-inspired algorithm. *Int. J. Mach. Learn. Cybern.* **2020**, *11*, 1501–1529. [CrossRef]
22. Ma, Z.Q.; Wu, G.H.; Suganthan, P.N.; Song, A.J.; Luo, Q.Z. Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. *Swarm Evol. Comput.* **2023**, *77*, 101248. [CrossRef]
23. Taib, H.; Bahreininejad, A. Data clustering using hybrid water cycle algorithm and a local pattern search method. *Adv. Eng. Softw.* **2021**, *153*, 102961. [CrossRef]
24. Chen, J.X.; Gong, Y.J.; Chen, W.N.; Li, M.; Zhang, J. Elastic Differential Evolution for Automatic Data Clustering. *IEEE Trans. Cybern.* **2021**, *51*, 4134–4147. [CrossRef]
25. Wang, J.Z.; Zhang, H.P.; Luo, H. Research on the construction of stock portfolios based on multiobjective water cycle algorithm and KMV algorithm. *Appl. Soft Comput.* **2022**, *115*, 108186. [CrossRef]
26. Sallam, K.M.; Abohany, A.A.; Allahi, R.M. An enhanced multi-operator differential evolution algorithm for tackling knapsack optimization problem. *Neural Comput. Appl.* **2023**.
27. Li, J.Y.; Zhan, Z.H.; Tan, K.C.; Zhang, J. A Meta-knowledge transfer-based differential evolution for multitask optimization. *IEEE Trans. Evol. Comput.* **2022**, *26*, 719–734. [CrossRef]
28. Liao, Z.W.; Mi, X.Y.; Pang, Q.S.; Sun, Y. History archive assisted niching differential evolution with variable neighborhood for multimodal optimization. *Swarm Evol. Comput.* **2023**, *76*, 101206. [CrossRef]
29. Wang, Z.; Chen, Z.; Wang, Z.; Wei, J.; Chen, X.; Li, Q.; Zheng, Y.; Sheng, W. Adaptive memetic differential evolution with multi-niche sampling and neighborhood crossover strategies for global optimization. *Inf. Sci.* **2022**, *583*, 121–136. [CrossRef]
30. Yan, X.; Tian, M. Differential evolution with two-level adaptive mechanism for numerical optimization. *Knowl.-Based Syst.* **2022**, *241*, 108209. [CrossRef]
31. Liu, D.; He, H.; Yang, Q.; Wang, Y.; Jeon, S.W.; Zhang, J. Function value ranking aware differential evolution for global numerical optimization. *Swarm Evol. Comput.* **2023**, *78*, 101282. [CrossRef]

32. Li, X.S.; Wang, K.Y.; Yang, H.C. PAIDDE: A permutation-archive information directed differential evolution algorithm. *IEEE Access* **2022**, *10*, 50384–50402. [CrossRef]
33. Li, Y.Z.; Wang, S.H.; Yang, H.Y. Enhancing differential evolution algorithm using leader-adjoint populations. *Inf. Sci.* **2023**, *622*, 235–268. [CrossRef]
34. Yi, W.C.; Chen, Y.; Pei, Z.; Lu, J.S. Adaptive differential evolution with ensembling operators for continuous optimization problems. *Swarm Evol. Comput.* **2022**, *69*, 100994. [CrossRef]
35. He, Y.; Zhang, F.; Mirjalili, S.; Zhang, T. Novel binary differential evolution algorithm based on Taper-shaped transfer functions for binary optimization problems. *Swarm Evol. Comput.* **2022**, *69*, 101022. [CrossRef]
36. Han, Y.; Yan, X.; Gu, X. Novel hybrid discrete differential evolution algorithm for the multi-stage multi-purpose batch plant scheduling problem. *Appl. Soft Comput.* **2022**, *115*, 108262. [CrossRef]
37. Gao, Z.; Zhang, M.; Zhang, L. Ship-unloading scheduling optimization with differential evolution. *Inf. Sci.* **2022**, *591*, 88–102. [CrossRef]
38. Ali, M.; Essam, D.; Kasmarik, K. A novel design of differential evolution for solving discrete traveling salesman problems. *Swarm Evol. Comput.* **2020**, *52*, 100607. [CrossRef]
39. Ali, M.; Essam, D.; Kasmarik, K. Novel binary differential evolution algorithm for knapsack problems. *Inf. Sci.* **2021**, *542*, 177–194. [CrossRef]
40. Opara, K.R.; Arabas, J. Differential evolution: A survey of theoretical analyses. *Swarm Evol. Comput.* **2019**, *44*, 546–558. [CrossRef]
41. Samuel, A.L. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* **1959**, *3*, 211–229. [CrossRef]
42. Shannon, C.E.; Hsu, T.S. Programming a Computer for Playing Chess. *Philos. Mag.* **1950**, *314*, 256–275. [CrossRef]
43. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef]
44. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Hassabis, D. Mastering the game of Go without human knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef]
45. Sandholm, T. Depth-Limited Solving for Imperfect-Information Games. *Science* **2018**, *347*, 122–123. [CrossRef]
46. Bowling, M.; Burch, N.; Johanson, M.; Tammelin, O. Heads-up limit hold'em poker is solved. *Science* **2015**, *347*, 145–149. [CrossRef]
47. Zhao, E.; Yan, R.; Li, J.; Li, K.; Xing, J. AlphaHoldem: High-Performance Artificial Intelligence for Heads-Up No-Limit Poker via End-to-End Reinforcement Learning. In Proceedings of the 36th AAAI Conference on Artificial Intelligence, Virtual, 22 February–1 March 2022; Volume 36, pp. 4689–4697.
48. Jiang, Q.; Li, K.; Du, B.; Chen, H.; Fang, H. DeltaDou: Expert-level Doudizhu AI through Self-play. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), Macao, China, 10–16 August 2019.
49. Mizukami, N.; Tsuruoka, Y. Building a computer Mahjong player based on Monte Carlo simulation and opponent models. In Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG), Tainan, Taiwan, 31 August–2 September 2015; pp. 275–283.
50. Yoshimura, K.; Hochin, T.; Nomiya, H. Searching optimal movements in multi-player games with imperfect information. In Proceedings of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), Okayama, Japan, 26–29 June 2016; pp. 1–6.
51. Li, J.; Koyamada, S.; Ye, Q.; Liu, G.; Hon, H.W. Suphx: Mastering Mahjong with Deep Reinforcement Learning. *arXiv* **2020**, arXiv:2003.13590.
52. Sato, H.; Shirakawa, T.; Hagihara, A.; Maeda, K. An analysis of play style of advanced mahjong players toward the implementation of strong AI player. *Int. J. Parallel Emergent Distrib. Syst.* **2017**, *32*, 195–205. [CrossRef]
53. Box, J. Guinness, Gosset, Fisher, and Small Samples. *Stat. Sci.* **1987**, *2*, 45–52. [CrossRef]
54. Wilcoxon, F. Individual comparisons by ranking methods. *Biom. Bull* **1945**, *6*, 80–83. [CrossRef]
55. Derrac, J.; Garca, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
56. Lee, Y.; Filliben, J.J.; Micheals, R.J.; Phillips, P.J. Sensitivity analysis for biometric systems: A methodology based on orthogonal experiment designs. *Comput. Vis. Image Underst.* **2013**, *117*, 532–550. [CrossRef]