# Understanding 31-derful

Emily Alfs
Mathematics and Computer Science
Doane College
Crete, Nebraska 68333
emily.alfs@doane.edu

## Abstract

Similar to Sudoku, 31-derful is a game where the player tries to place values into certain positions to win the game. 31-derful is also similar to magic squares in that to win the game, the rows and columns must all add up to the same value. To win 31-derful, the player gets to pick cards from a standard deck of playing cards (minus jokers) into a five by five grid with each row and column adding up to thirty-one.

The overarching goal of our research is to gain a better understanding of 31-derful. We want to find what range of sums are playable and then look into the levels of playing difficulty. We also want to see if an algorithm can be created to win all possible games. In order to fully understand the game, we began our research by minimizing our problem to its simplest form. Once this was completed, we started research on the full-scale game using a genetic algorithm.

# 1 Introduction

In the summer of 2014, along with a mathematics faculty advisor, I began research on the card game 31-derful. The game consists of a five by five grid and a full deck of playing cards. The goal of the game is to place the cards in open spaces to ultimately get the rows and columns to add up to thirty-one. All cards are worth their face value, except for jacks, kings, and queens. These cards are worth ten and aces are worth eleven. An example of a winning game is in figure one below.

| | | | | |
|---|---|---|---|---|
| K | Q | 5 | 2 | 4 |
| 10 | J | 5 | 4 | 2 |
| 6 | 6 | 9 | 5 | 5 |
| 2 | 3 | 6 | J | 10 |
| 3 | 2 | 6 | K | Q |

Figure 1: A winning game. The suits of the cards do not matter.

The research we are conducting on the game focuses on how many possible ways there are to complete the game, what makes a game unique, and attempting to construct a game winning strategy so players can always win. This paper will cover our initial approach to the problem by scaling it down and how we are utilizing a genetic algorithm to approach the full-scale game.

## 1.1 Uniqueness of a Game

Due to the number of possible solutions, we needed to determine what made a game unique. Doing so would lower the number of total solutions that would need to be analyzed. We applied matrix operations to games to determine if they are unique. We decided that a matrix operation, like a row or column swap, did not make a unique game from the original. For example, if a game is rotated 90 degrees, the player can see the same game by moving themself 90 degrees around the playing surface. We also eliminated transposes and rotations of originals, as these do not create what we have decided to call a unique game. An example of a few of these can be seen below in figure two.

Original:

| | | | | |
|---|---|---|---|---|
| K | Q | 2 | 5 | 4 |
| 10 | J | 4 | 5 | 2 |
| 6 | 6 | 5 | 9 | 5 |
| 2 | 3 | J | 6 | 10 |
| 3 | 2 | K | 6 | Q |

Row swap:

| | | | | |
|---|---|---|---|---|
| 10 | J | 5 | 4 | 2 |
| K | Q | 5 | 2 | 4 |
| 6 | 6 | 9 | 5 | 5 |
| 2 | 3 | 6 | J | 10 |
| 3 | 2 | 6 | K | Q |

Column swap:

|  |  |  |  |  |
|---|---|---|---|---|
| K | Q | 5 | 2 | 4 |
| 10 | J | 5 | 4 | 2 |
| 6 | 6 | 9 | 5 | 5 |
| 2 | 3 | 6 | J | 10 |
| 3 | 2 | 6 | K | Q |

Transpose:

|  |  |  |  |  |
|---|---|---|---|---|
| K | 10 | 6 | 2 | 3 |
| Q | J | 6 | 3 | 2 |
| 5 | 5 | 9 | 6 | 6 |
| 2 | 4 | 5 | J | K |
| 4 | 2 | 5 | 10 | Q |

Figure 2: Although these games look different upon first glance, they are considered the same in our research.

## 2 Scaling Down the Problem

We have completed research on the scaled down cases of the two by two, the three by three, and the four by four games. For our initial research, we chose to scale the game down to gain a better understanding of the problems involved when playing. This was also beneficial for future research in terms of accuracy of our program. In the smallest case, the two by two version, we were able enumerate the only existing solution by hand which allowed us to test the accuracy of our program.

To emulate the full-scale game in our small-scale research, we put the limitation of one suit less than the dimension. Therefore, in the case of the two by two game, only one suit was used and in the case of the three by three, two suits were used. We altered the limitation of adding up to thirty-one to have all of the rows and columns add up to the same number. When a game satisfies this rule, it will result in a winning game. An example of a winning three by three game can be seen below in figure three.

|  |  |  |
|---|---|---|
| 2 | 5 | 9 |
| 5 | 7 | 4 |
| 9 | 4 | 3 |

Figure 3: This is a winning three by three game with a goal sum of sixteen.

An example of a winning four by four game can be seen below in figure four.

|  |  |  |  |
|---|---|---|---|
| 2 | 4 | A | A |
| 5 | 4 | 9 | 10 |
| J | K | 3 | 5 |
| A | Q | 5 | 2 |

Figure 4: This winning four by four game has a goal sum of twenty-eight.

## 2.1 Two by Two Case

Research on the two by two version of the game was straightforward. We were able to find the only winning game by hand. The winning game can be seen below in figure three.

| | |
|---|---|
| K | Q |
| J | 10 |

Figure 3: A game with a goal sum of 20 is the only possible way to win in the two by two version.

We still created a computer program that would create the solution using a breadth first search. From a mathematical perspective, this was helpful in seeing partial games so we could do a deeper analysis on algorithm creation.

## 2.2 Three by Three case

Once the two by two research was completed, we scaled the program up to the three by three case. There were 358 unique games with sums ranging from twelve to thirty. Each game falls into one of five distinct categories. Using these categories, we were able to construct a game winning strategy so players can always win in the three by three case. Interestingly, we found that the list of cards to be played was enough to determine their placement. For example, if someone were to hand you nine cards that create a winning game, there is only one unique game that can be played. An example of this using our game winning strategy can be seen below in figure four.

i) List of cards to use: 2, 4, 4, 8, 8, 10, J, Q, K

| | | |
|---|---|---|
| | | |
| | 2 | |
| | | |

ii) List of cards to use: 4, 4, 8, 8, 10, J, Q, K

| | | |
|---|---|---|
| | J | |
| 10 | 2 | Q |
| | K | |

iii) List of cards to use: 4, 4, 8, 8

| | | |
|---|---|---|
| 4 | J | 8 |
| 10 | 2 | Q |
| 8 | K | 4 |

Figure 4: In first step, the player first places the card that has no pairing within the list, the two. Next the player places the cards with values of ten directly adjacent to the center. Finally, the player places the remaining pairs. They do so by placing the same values in

opposite corners. This is a winning game with a sum of twenty-two. There is no other unique way to place these cards.

### 2.3 Four by Four Case

As we scaled up to the four by four, the computational complexity increased exponentially. Because of this, we had to create a new program that would use a depth first search. We found that there are 251,212 unique solutions. In this case, we were unable to build a strategy to always win. This was partly due to the fact that unlike in the case of the three by three, the list of cards does not necessarily decide which game will be played. Depending on the list of cards, a player could make up to 178 unique games with the same list. An example of two unique games with the same list of cards can be seen below in figure four.

| | | | |
|---|---|---|---|
| 2 | 4 | 8 | A |
| 4 | 6 | 8 | 7 |
| 8 | Q | 3 | 4 |
| A | 5 | 6 | 3 |

| | | | |
|---|---|---|---|
| 2 | 4 | 8 | A |
| 4 | 6 | 7 | 8 |
| 8 | Q | 4 | 3 |
| A | 5 | 6 | 3 |

Figure 4: These games are played with the same cards but they cannot be made the same using matrix operations. A half row/column swap behavior can be seen in rows two and three column's three and four.

In the four by four case, "half row/column swaps" were the most common way that games would have the same list but still be unique. We gathered that this was due to the intricate balance of lowering and raising the row/column sum evenly. There were also some cases where the half row/column swap would happen in three columns.

## 3 Full Scale Game

Research is being done on the full-scale version of the game to reach the goals we initially set. Similar to the scale up from the three by three to the four by four, the scale up to the five by five becomes even more computationally complex. Thus, we are implementing a genetic algorithm to try to evolve winning five by five games.

### 3.1 Introduction to Genetic Algorithms

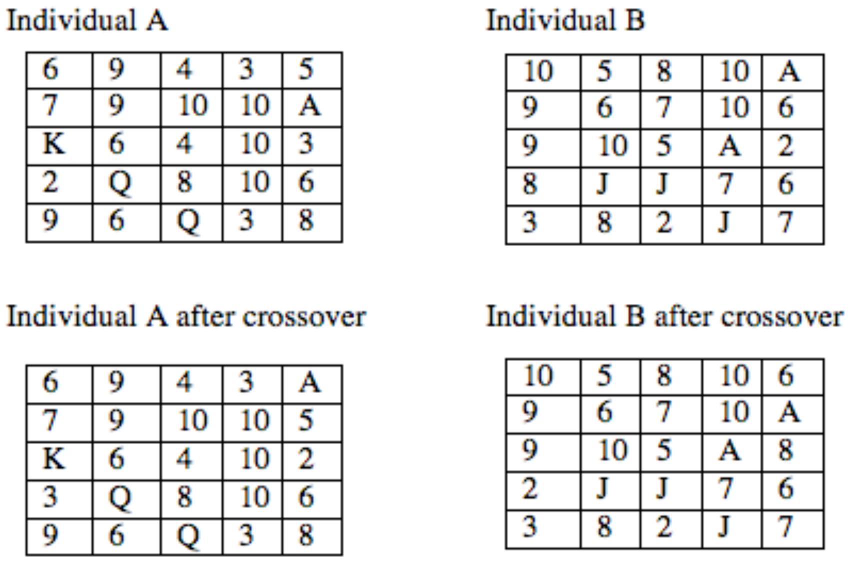
Genetic algorithms are used to model natural evolution using operators such as crossover, mutation, and selection. Crossover behaves like a mating process in that it takes two individuals, switches some of their DNA, and produces two new offspring. Mutation takes a single individual and randomly changes its DNA based on a probability of mutation. Selection takes two random individuals, evaluates their fitness's, and selects the individual with the better fitness [1].

The genetic algorithm we created was based off of DEA (Doane Evolutionary Algorithm) [2]. This program had abstract methods in place that allowed us to easily implement our specific needs by extending those existing methods. All we needed to do was modify the operators, create a method to evaluate fitness, and create a method to randomly generate individuals.

Our individuals are single games, which are represented as two-dimensional arrays. Our genetic algorithm creates these individuals by randomly selecting values without replacement from a linked list that contains the values zero to fifty-one and placing them into the array. The linked list ensures that the same value will never be selected twice. Those values represent our cards. Meaning, numbers zero through three represent all possible two's, four through seven represent all three's, and so on. This helps to maintain the number of repetitions allowed per card value. Thus, our algorithm will never create an invalid game by allowing too many repetitions. We wanted to have more meaningful output than a grid with values of zero through fifty-one, so we created a method that would interpret the values and put the corresponding cards in when output was needed.

## 3.2 Crossover

As of now, we have two versions of crossover. In one version, crossover takes a certain number of cards that two individuals have in common and changes the locations. An example of this can be seen in figure five below.



Individual A

| | | | | |
|---|---|---|---|---|
| 6 | 9 | 4 | 3 | 5 |
| 7 | 9 | 10 | 10 | A |
| K | 6 | 4 | 10 | 3 |
| 2 | Q | 8 | 10 | 6 |
| 9 | 6 | Q | 3 | 8 |

Individual B

| | | | | |
|---|---|---|---|---|
| 10 | 5 | 8 | 10 | A |
| 9 | 6 | 7 | 10 | 6 |
| 9 | 10 | 5 | A | 2 |
| 8 | J | J | 7 | 6 |
| 3 | 8 | 2 | J | 7 |

Individual A after crossover

| | | | | |
|---|---|---|---|---|
| 6 | 9 | 4 | 3 | A |
| 7 | 9 | 10 | 10 | 5 |
| K | 6 | 4 | 10 | 2 |
| 3 | Q | 8 | 10 | 6 |
| 9 | 6 | Q | 3 | 8 |

Individual B after crossover

| | | | | |
|---|---|---|---|---|
| 10 | 5 | 8 | 10 | 6 |
| 9 | 6 | 7 | 10 | A |
| 9 | 10 | 5 | A | 8 |
| 2 | J | J | 7 | 6 |
| 3 | 8 | 2 | J | 7 |

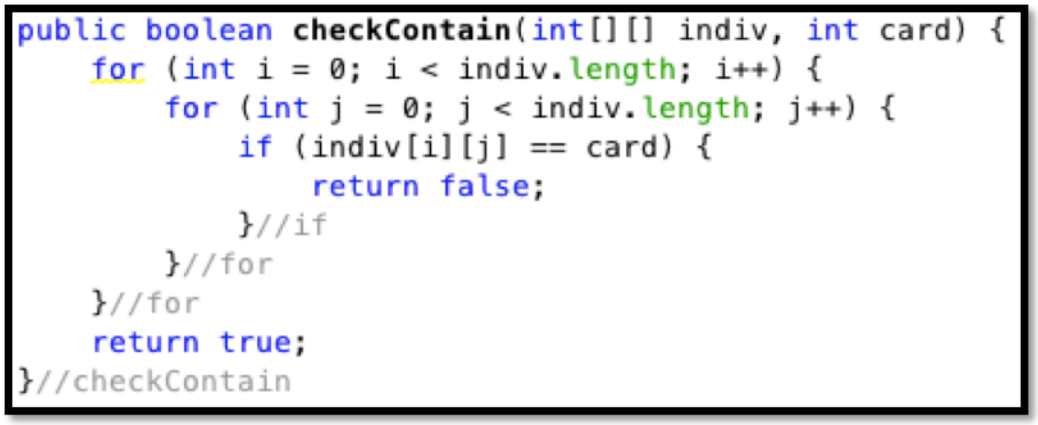
Figure 5: In this example, the ace and the two were the chosen cards in common.

In the figure five example, within individual a, the locations of individual a's two and ace were changed to the location of individual b's two and ace. The cards that were overwritten in individual a, the five and the three, were then placed in the two and ace's former positions according to which one overwrote it. A similar process happened in individual b but with the six and the eight.

In trials of this version we found that it has the tendency to change the population too dramatically. Thus, we had to create a new version of crossover.
The second version will take a whole row or column from the first individual and swap it with a whole row or column from the second individual. One of the main issues that have come from crossover is maintaining the proper number of cards in a game. When operating with crossover, we run a high risk of putting too many of the same card into a game. In order to compensate for this, we are implementing a method that will lower or raise the duplicate card by one. This will potentially allow crossover to make minor changes to make an invalid game into a valid game.

## 3.3 Mutation

Our mutation operator is rather straightforward. The mutation will happen if a random number between zero and one is less than our chosen μ, which we have set to .01. The method then picks a random card and location then places the card into the game. Similarly to crossover, we run the risk of making a game invalid by placing too many of the same card in the game. To compensate for that, we implemented a checkContain method, which can be seen below in figure six.



```
public boolean checkContain(int[][] indiv, int card) {
    for (int i = 0; i < indiv.length; i++) {
        for (int j = 0; j < indiv.length; j++) {
            if (indiv[i][j] == card) {
                return false;
            }//if
        }//for
    }//for
    return true;
}//checkContain
```

Figure 6: If the individual contains the card already then no mutation occurs.

Since checkContain can inadvertently lower our chance of mutation, we may need to raise our μ value after further testing. An example of an individual before and after mutation can be seen below in figure seven.

| | | | | |
|---|---|---|---|---|
| A | K | 5 | 6 | 2 |
| J | 3 | 9 | 7 | 10 |
| 2 | 4 | 8 | 7 | 8 |
| K | 10 | Q | A | 9 |
| 2 | Q | 9 | 4 | 5 |

After mutation:

| | | | | |
|---|---|---|---|---|
| A | K | 5 | 6 | 2 |
| J | 3 | 9 | 7 | 10 |
| 2 | 4 | 8 | 10 | 8 |
| K | 10 | Q | A | 9 |
| 2 | Q | 9 | 4 | 5 |

Figure 7: This would be an example of an individual that made it through mutation.

In figure seven, the seven that was in spot [3,4] was changed to a ten. Both the location and the new card were randomly chosen.

## 3.4 Selection

Within our program, we use elitist tournament selection. This ensures that our best individual from the population survives into the next generation. Without elitist tournament selection, this is not guaranteed. In regular selection two individuals from the current population are selected and have their fitness's compared. Whichever individual has the higher fitness goes to the next generation. However, both individuals remain in the current population and are subject to selection again. By using elitist tournament selection, our next generation will always contain the best member of the previous population.

## 3.5 GA Results

Using the first version of crossover, our genetic algorithm has produced winning games. The solutions our genetic algorithm has found thus far have had their rows and columns add up to the same value. However, it has not found a solution where the rows and columns add up to thirty-one. An example of a winning game that our genetic algorithm produced can be seen below in figure eight.

| | | | | |
|---|---|---|---|---|
| 10 | 8 | J | 10 | A |
| 9 | Q | 10 | 9 | A |
| K | J | Q | 10 | 9 |
| K | K | A | Q | 8 |
| J | A | 8 | Q | 10 |

Figure 8: This game has a goal sum of forty-nine

Another example of a winning game that has been found by our genetic algorithm can be seen below in figure nine.

| | | | | |
|---|---|---|---|---|
| 10 | 10 | 7 | A | 9 |
| 7 | 9 | 10 | 10 | A |
| J | Q | 9 | 8 | J |
| Q | K | J | K | 7 |
| J | 8 | A | 8 | K |

Figure 9: This game has a goal sum of forty-seven.

Both of these examples were created using a crossover probability of .6, a mutation probability of .01, population sizes of 100,000, and 2,000 generations.

# 4 Moving Forward

Though we have completed research on all of the small-scale versions of the game, we still have unanswered questions. Ultimately, we would like to see what the distribution of the full-scale sum range is to determine if thirty-one is the best version of the game. Our hope is that this research and continued research can shed light on this problem.

## References

[1] Hiu Man Wong. Genetic Algorithms. In *SURPRISE 96 Journal*, 1996
[2] Mark M. Meysenburg. The DEA: A Framework for Exploring Evolutionary Computation. In *MICS 2004: Proceedings of the Midwest Instruction and Computing Symposium*, April 2004.
[3]J.F.Crook. A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. In *Notices of AMS*, Volume 56 Number 4, April of 2009