# An Optimal Algorithm for a Strategy Game

Daxin Zhu[1, a] and Xiaodong Wang[2,b*]

[1] Quanzhou Normal University, Quanzhou 362000, China

[2] Fujian University of Technology, Fuzhou 350108, China

[a] dex@qztc.edu.cn, [b] wangxd@fzu.edu.cn, [*]corresponding author

**Keywords:** strategy game, algorithm, explicit solution, optimal move.

**Abstract.** A single player strategy game is studied in this paper. We are interested in algorithms which, given integer *n*, generate the corresponding move sequences to reach the final state of the game with smallest number of steps. In this paper we present an optimal algorithm to generate an optimal move sequence of the game consisting of $n$ black checkers and $n$ white checkers, and finally, we present an explicit solution for the general game of size *n*.

## Introduction

Combinatorial games often lead to interesting, clean problems in algorithms and complexity theory. Many classic games are known to be computationally intractable. Solving a puzzle is often a challenge task like solving a research problem. You must have a right cleverness to see the problem from a right angle, and then apply that idea carefully until a solution is found.

In this paper we study a single player game called moving checkers. The game is similar to the Moving Coins puzzle [2,3], which is played by re-arranging one configuration of unit disks in the plane into another configuration by a sequence of moves, each repositioning a coin in an empty position that touches at least two other coins. In our moving checkers game, there are $n$ black checkers and $n$ white checkers put on a table from left to right in a row. The $2n+2$ positions of the row are numbered $0,1,\cdots,2n-1$. Initially, the $n$ black checkers are put on the position $0,1,\cdots,n-1$, and the $n$ white checkers are put on the position $n,n-1,\cdots,2n-1$. The rightmost two positions $2n$ and $2n+1$ are vacant. In the final state of the game, the positions of even number $2,4,\cdots,2n$ are occupied by white checkers, and the positions of odd number $3,5,\cdots,2n+1$ are occupied by black checkers, leaving the two positions $0$ and $1$ vacant. For easy to say, we call the number $n$ the size of the game even though we have $2n$ checkers.

A move of the game consists of shifting two adjacent checkers, keeping their order, into the current two vacant positions. The goal of the game is to make a smallest number of moves to reach the final state of the game.

We are interested in algorithms which, given integer $n$, generate the corresponding move sequences to reach the final state of the game with smallest number of steps. In this paper we present an optimal algorithm to generate an optimal move sequence of the game consisting of $n$ black checkers and $n$ white checkers.

This paper is structured as follows.

In the following 4 sections we describe the algorithms and our computational experience with the algorithms for generating optimal move sequence of the general game consisting of $n$ black checkers and $n$ white checkers. In section 2, we describe a new variant tree search based algorithm for generating all optimal solutions for the moving checkers games of small size. A linear time recursive construction algorithm is proposed in section 3. Based on the recursive algorithm proposed in section 3, an explicit solution for the optimal move sequence of the general game is presented in section 4. Some concluding remarks are in section 5.

 

**A Backtracking Algorithm**

It is not difficult to verify the following facts on the minimum number of steps needed to play the game.

**Theorem 1** *For the general game consisting of $n$ black checkers and $n$ white checkers, it needs at least $n$ steps to reach the final state of the game from its initial state.*

**Proof.** In a row of checkers of the game, if two adjacent checkers have different colors, the two checkers are called an inversion pairs. For example, in the initial state of the game consisting of $n$ black checkers and $n$ white checkers, there is only one inversion pairs, while in the final state of the game, there are total $2n-1$ inversion pairs.

It is readily to see that the inversion pairs are increased by $2n-2$ from initial state to the final state of the game. In the first step of move, at most one inversion pairs can be added and in the subsequent moves at most two inversion pairs can be added in each step. If the final state is reached after $m$ steps, then at most $2m-1$ inversion pairs are added. Therefore, we have, $2m-1 \geq 2n-2$ and so $m \geq n-1/2$. Since $m$ is an integer, we have $m \geq n$.

In other words, it needs at least $n$ steps to reach the final state of the game from its initial state. ■

According to Theorem 1, if we can find a move sequence to reach the final state of the game with $n$ steps, then the sequence will be an optimal move sequence, since no move sequence can reach the final state of the game in less than $n$ steps. In order to study the structures of the optimal solutions for the general moving checkers game, we first present a backtracking algorithm [1,5,6] to generate all optimal solutions of the games with small size.

**Algorithm 2.1:** BACKTRACK$(i, e)$

**comment:** Generate all optimal solutions

**if** $i > n$ **and** $e = 0$ **and** final state reached
  **then** output current solution

**else**
- **for** $j \leftarrow 0$ **to** $2n$
  - **do if** $j < e-1$ **or** $j > e+1$
    - **then**
      - $x[i-1] \leftarrow j$
      - move checkers at position j and j+1 to vacant
      - BACKTRACK$(i+1, j)$
      - move checkers at position e and e+1 to vacant

In the algorithm described above, the parameter $i$ is current number of steps and the parameter $e$ is the left position of current vacant. The current solution is stored in array $x$. For $i = 1,2,\cdots,n$, the move of step $i$ is stored in $x[i-1]$. This means that we move the adjacent pair of checkers located at positions $x[i-1]$ and $x[i-1]+1$ to the current vacant positions and leaving the positions $x[i-1]$ and $x[i-1]+1$ the new vacant positions. A recursive function call *Backtrack*(1,2$n$) will generate all optimal solutions which move checkers from initial state to final state in $n$ steps.

It is not difficult to generate all optimal solutions of the game with small size by the backtracking algorithm described above.

For the cases of $n \leq 3$, there are no solutions found. The unique optimal solutions found for the cases of $n = 4,5,6$ are {1,4,7,0}, {1,7,4,9,0} and {1,7,3,8,11,0} respectively. There are 2 optimal solutions {1,10,4,9,6,13,0} and {1,10,4,13,6,9,0} for the case of $n = 7$. For the case of $n = 8,9$, there are total 16 and 32 optimal solutions found respectively.

**A Linear Time Construction Algorithm**

The backtracking algorithm described in the previous section can produce all optimal solutions for the game with fixed size $n$. It works only for small size $n$. If we concentrate to find one optimal solution for the game, we can do better. In this section, we will present a linear time construction algorithm which can produce an optimal solution in linear time for very large size $n$. The Decrease-and-Conquer strategy [4,7] for algorithm design is exploited to design our new algorithm.

For the cases of $n \le 7$, we can apply the solution found in the previous section directly. For the cases of $n \ge 8$, we can find one optimal solution for the game recursively as follows.

It has been known that the optimal solution for the game of size $n$ consists of $n$ steps.

In the new construction algorithm, first 2 steps are constructed explicitly to make checkers located at the positions $4,5,\cdots,2n-3$ exactly the same as the initial state of a game of size $n-4$.

Then, in the steps $3,\cdots,n-2$, the algorithm is applied recursively to the game of size $n-4$, in which the checkers are located at the positions $4,5,\cdots,2n-3$.

Finally, the last 2 steps are constructed explicitly to make a whole solution.

Now we can describe the new construction algorithm *Move* as follows.

In the algorithm described above, the parameters *first* and *k* describe the initial state of the sub-game of size *k* starting at position *first*. The current step of the game is stored in a global variable *step*. The array *x* is used to store the optimal solution of the game. For example, a function call *Move*(0,4) will return an optimal solution for the game of size 4 by the array $x = \{1,4,7,0\}$.

**Algorithm 3.1:** MOVE$(first, k)$

```
if k < 8
  then construct the solution directly
  else { comment: first 2 moves
         x[step] ← first + 1
         x[step + 1] ← first + 2 * k − 4
         step ← step + 2
         comment: recursive moves
         MOVE(first + 4, k − 4)
         comment: last 2 moves
         x[step] ← first + 2 * k − 1
         x[step + 1] ← first
         step ← step + 2
       }
```

The correctness of the algorithm can be proved readily by induction. Now we consider the time complexity for the whole algorithm. Suppose the time required by the algorithm for the game of size $n$ be $T(n)$. The first 2 moves and the last 2 moves of the algorithm *Move* cost $O(1)$ time. In the middle part of the algorithm, a recursive call is applied to a sub-game of size $n-4$ requiring $T(n-4)$ time. For the games of small size of $n < 8$, the time costs of the algorithm are obviously $O(1)$. Therefore, the following recurrence holds for $T(n)$.

$$T(n)=\begin{cases}O(1) & n<8\\ T(n-4)+O(1) & n\geq 8\end{cases} \qquad (1)$$

The solution of this recurrence is obviously $T(n)=O(n)$. In other words, the algorithm *Move* for general game of size $n$ requires $O(n)$ time. The space used by the algorithm is obviously $O(n)$.

**Theorem 2** *The algorithm Move for solving the general moving checkers game of size n requires $O(n)$ time and $O(n)$ space.*

**The Explicit Solution of the Problem**

The optimal solution found by the algorithm *Move* is presented by a vector $x$. For $i=1,2,\cdots,n$, the step $i$ of the optimal move sequence is given by $x[i-1]$. This means that the adjacent pair of checkers located at positions $x[i-1]$ and $x[i-1]+1$ will be moved in step $i$ to the current vacant positions and leaving the positions $x[i-1]$ and $x[i-1]+1$ the new vacant positions. This can also be viewed that $x$ is a function of $i$. In this section we will discuss the explicit expression of function $x$.

For the small size cases of $n=4,5,6,7$, the optimal solutions of the game can be listed as a small two dimensional array $d$ as follows.

Table 1: Small two dimensional array $d$

| $n$ \ $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 4 | 7 | 0 | 0 | 0 | 0 |
| 5 | 1 | 7 | 4 | 9 | 0 | 0 | 0 |
| 6 | 1 | 7 | 3 | 8 | 11 | 0 | 0 |
| 7 | 1 | 10 | 4 | 9 | 6 | 13 | 0 |

For the games of size $n=4,5,6,7$, the step $i,1\leq i\leq n$ of the optimal solution can be expressed as $d(n,i)$. For the cases of sub-games of size $n-t=4,5,6,7$, starting at position $t$, the corresponding step $j,1\leq j\leq n-t$ of the optimal solution can be expressed as $t+d(n-t,j)$. For the general cases of sub-games of size $n$, starting at position $t$, if the corresponding step $j,1\leq j\leq n$ of the optimal solution is denoted as $m(t,j)$, then for the game of size $n$, the optimal move $x[i-1],1\leq i\leq n$ must be $m(0,i),1\leq i\leq n$.

According to the algorithm *Move* presented in previous section, the function $m(t,j)$ can be computed as follows.

$$m(t,j)=\begin{cases}t+d(n-t,j) & 3<n-t,\quad n-t<8\\ t+1 & j=1\\ 2n-t-4 & j=2\\ 2n-t-1 & j=n-t-1\\ t & j=n-t\\ m(t+4,j-2) & \text{otherwise}\end{cases} \qquad (2)$$

If $i$ or $n-i$ is a constant, then $x[i-1]=m(0,i)$ can be computed in $O(1)$ time by the formula given above. In other cases, if both $i$ and $n-i$ are in $O(n)$, then the time costs to compute $m(0,i)$ by the formula given above must be $O(n)$. However, we can reduce the formula further to an explicit formula to compute each of $x[i-1]=m(0,i),1\le i\le n$ in $O(1)$ time as follows.

**Theorem 3** *The optimal solution $x[i-1]=m(0,i),1\le i\le n$ found by the algorithm Move for solving the general moving checkers game of size $n$ can be expressed explicitly as follows.*

$$m(0,i)=\begin{cases}4k+d(r+4,i-2k) & 2k<i<2k+r+3\\ 2i-1 & i\le 2k,\quad i \text{ odd}\\ 2(n-i) & i\le 2k,\quad i \text{ even}\\ 2i+1 & i\ge 2k+r+3,\quad \text{n - i odd}\\ 2(n-i) & i\ge 2k+r+3.\quad n-i \text{ even}\end{cases} \tag{3}$$

*where,* $k=\lfloor n/4\rfloor-1,\ r=n \bmod 4$.

**Proof.** From the construction steps of the algorithm *Move*, the optimal move steps can be divided into three parts, the first 2 moves, recursive moves and the last 2 moves. Thereby the $n$ steps of the optimal move sequence $x[i-1],1\le i\le n$ generated by the algorithm *Move* can also be divided into thee parts accordingly. These three parts are the first part $1\le i\le 2k$, the second part $2k<i<2k+r+3$ and the third part $2k+r+3\le i\le n$, respectively, where $k=\lfloor n/4\rfloor-1$, $r=n \bmod 4$.

Every move step $i,1\le i\le n$, corresponds to a sub-game starting at position $t$, and this starting position $t$ can also be determined by the value of $i$.

In the first part of the optimal move steps, the moves are generated by the first 2 moves of the algorithm *Move*. It is not difficult to see that if $i$ is odd then $t=2(i-1)$ otherwise $t=2(i-2)$. In this case, according to formula (2), if $i$ is odd then $m(0,i)=t+1=2(i-1)+1=2i-1$, otherwise, $m(0,i)=2n-t-4=2n-2(i-2)-4=2(n-i)$.

Similarly, in the third part of the optimal move steps, the moves are generated by the last 2 moves of the algorithm *Move*. In this case, the starting position $t$ of the sub-game corresponding to step $i$ can be determined by the value of $n-i$. It is not difficult to see that if $n-i$ is odd then $t=2(n-i-1)$ otherwise $t=2(n-i)$. It can also be derived by formula (2) that, if $n-i$ is odd then $m(0,i)=2n-t-1=2n-2(n-i-1)-1=2i+1$, otherwise, $m(0,i)=t=2(n-i)$.

For the second part of the optimal move steps, the starting position $t$ of the sub-game corresponding to step $i$ is obviously $4k$. There are $2k$ steps in the first part of the optimal move steps are generated before this part and thus the step $i$ corresponds to the step $j=i-2k$ of the sub-game. It follows from

$n=4\lfloor n/4\rfloor+r=4k+r+4$ that $n-t=n-4k=r+4$.

It can now be derived by formula (2) that $m(0,i)=t+d(n-t,j)=4k+d(r+4,i-2k)$.

The proof is completed. ■

It is obvious that the optimal move sequence $x[i-1],1\le i\le n$ of the general moving checkers game of size $n$ can be easily computed in optimal $O(n)$ time, since for each individual step $i$, its optimal move $x[i-1]=m(0,i)$ can be computed in $O(1)$ time by the explicit formula (3) described above.

## Concluding Remarks

We have studied the general moving checkers game of size $n$. It has been proved in the section 2 that the minimum number of steps needed to play the game of size $n$ is $n$. All of the optimal solutions for the moving checkers game of small size can be found by a backtracking algorithm presented in section 2. In the section 3, a linear time recursive construction algorithm which can produce an optimal solution in linear time for very large size $n$ is presented. The time cost of the new algorithm is $O(n)$ and $O(n)$ space is used. Finally, in the section 4, an extremely simple explicit solution for the optimal moving sequences of the general moving checkers game of size $n$ is given. The formula gives for each individual step $i$, its optimal move in $O(1)$ time.

For the interesting moving checkers problem, some research problems are open. An interesting result found by the backtracking algorithm is that the number of optimal solutions for moving checkers games of size 8 and 9 are 16 and 32 respectively. It seems to suggest that the number of optimal solutions for the general moving checkers games of size $n$ is $2^{n-4}$, but we have failed to prove it. Exponential time s required to find all optimal solutions for the general moving checkers game by the backtracking algorithm presented in section 2. Can we find an efficient algorithm to generate all optimal solutions for the general moving checkers game in the time proportional to the output size? This is also an open problem. We will investigate these problems further.

## Acknowledgement

This research was financially supported by the Natural Science Foundation of Fujian (Grant No.2013J01247), and Fujian Provincial Key Laboratory of Data-Intensive Computing and Fujian University Laboratory of Intelligent Computing and Information Processing.

## References

[1] R. Bird, Pearls of Functional Algorithm Design, 258-274, Cambridge University Press, 2010.

[2] Erik D. Demaine, Playing games with algorithms, Algorithmic combinatorial game theory. Proceedings of the 26th Symposium on Mathematical Foundations in Computer Science, LNCS 2136, 18-32, 2001.

[3] Erik D. Demaine and Martin L. Demaine, Puzzles, Art, and Magic with Algorithms, Theory of Computing Systems, vol. 39, number 3, 473-481, 2006.

[4] A. Levitin and M. Levitin, Algorithmic Puzzles, 3-31, Oxford University Press, New York, 2011.

[5] J. Kleinberg, E. Tardos. Algorithm Design, 223-238, Addison Wesley, 2005.

[6] D.L. Kreher and D. Stinson, Combinatorial Algorithms: Generation, Enumeration and Search, 125-133, CRC Press, 1998.

[7] S. Sukparungsee, Y. Areepong, Exact Average Run Length of Double Moving Control Chart, International Journal of Applied Mathematics & Statistics, Vol. 52, No. 2, 152-158, 2014.