

User Manual of Chinese Dark Chess Client

Jr-Chang Chen, Gang-Yu Fan and Ting-Yu Lin

This document includes five parts for the use of the client program to connect to the Chinese dark chess server. Section 1 describes the environment settings in **MS Windows** for the client package. Two modes are provided for your game-playing program to connect to a client. Section 2 describes the file I/O mode. Section 3 describes the socket I/O mode and the codes you need to write in your game-playing program to fit the client such that the pondering is possible. Section 4 shows a GUI interface. Section 5 is our contact information.

1. THE ENVIRONMENT

The demo game-playing program is a random-play program, called **MyAI** in the file “**search.exe**”. You can run two copies and connect to the server to test the connection.

1.1 Environment Setting

- The Chinese dark chess interface (**CDC_interface.zip**) includes an execution file “**DarkChess.exe**”, a library “**GameDLL.dll**” and a folder “**Search**”, as shown in Figure 1.

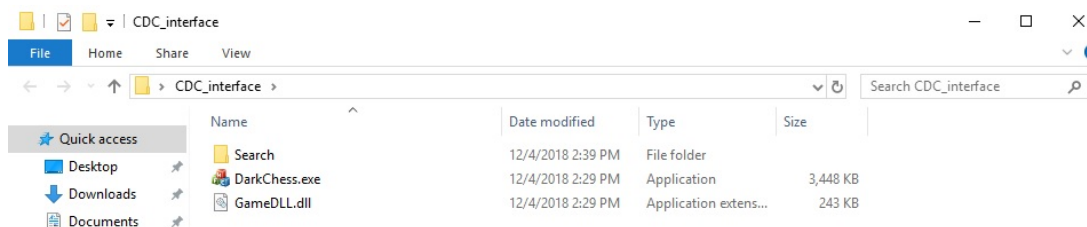


Figure 1: CDC_interface.zip

- Your game-playing program name has to be renamed as “**search.exe**”, and then replace the “**search.exe**” execution file in the “**Search**” folder, as shown in Figure 2.

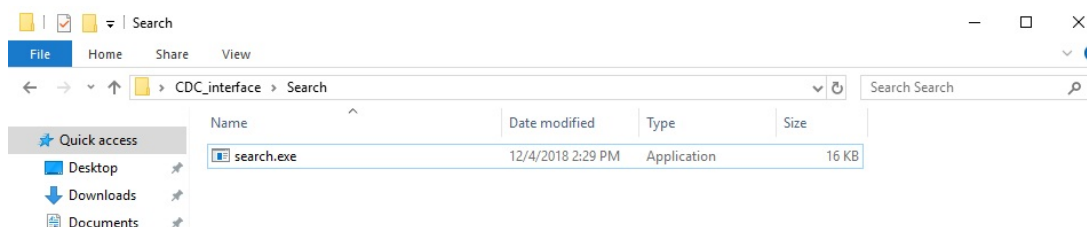


Figure 2: Replace search.exe by your game-playing program.

- Execute “**DarkChess.exe**” to run your game-playing program and connect to the server. Set the “**Mode**” as “**Computer vs. Computer**”, as shown in Figure 3.
- Click “**Start**” in “**Game**” to begin a new game, as shown in Figure 4.



Figure 3: Select a mode.



Figure 4: Start a game.

1.2 Settings for Creating and Joining a Room

The following settings are needed after clicking “**Start**” in “**Game**”, as shown in Figure 4.

Create a Room (Figure 5)

- **Accounts:** Default testing accounts are “a0”, “a1”, ..., and “a999”.
- **Password:** The password for all testing accounts is ”123”.
- **Game Mode:** Select a game mode that is “Single Game”, “Swiss” or “Round Robin”.
- **Reconnection** must be “Create (a room)”.
- **Activation:** Used when a disconnection between two players occurs. Choose “Yes” to continue the game with the position where the disconnection happens, and choose “No” to restart a new game.
- Choose a **Connection Mode** that is “file I/O” or “socket I/O”. Run search.exe by the GUI interface. “file I/O” indicates to read board.exe and output move.txt (see Section 2). “socket I/O” indicates to use the protocol that provides pondering (see Section 3). The search.exe can obtain parameters of this game via the protocol.
- Set **Debug Mode** to “Yes” to display a command line window and output a log; and “No” otherwise.
- Set **Automatic Next Game** to “Yes” to play multiple games and re-start automatically; and “No” otherwise.
- Set **#Games to Play** if **Automatic Next Game** is “Yes”. For example, “4” if four games will be played.
- **Move First:** Choose “First” to be the first player and “Second” to be the second player.
- **Move First Alternation:** Choose “Yes” to indicate playing firstly and playing secondly in turns if multiple games are played. Choose “No” to indicate that you always play firstly or secondly according to the choice in “**Move First**” for all games.
- Select a **Time Setting** that is “Total Time” or “Time per Move” (see below).
- **Time limit** is the total time you have in a game if you select “Total Time”. For example, “900” means you have to finish a game within 900 seconds. Or **Time limit** is the time your program is allowed to think per move. For example, “30” means you have to output a move within 30 seconds.

- **Times of Repetitive Positions:** Input “3” to mean the game is judged as a draw if the same position repeats three times.
- **Initial Position Assignment:** Choose “Yes” to assign an initial position, and “No” to randomly generate an initial position by the server.
- **Path:** When “Yes” is chosen in **Initial Position Assignment**, input the file path of the assigned initial position.
- **Server IP** is the IP of the Chinese dark chess server. Note that a new IP may be provided in a tournament.

Room Information

Account: a0

Password: 123

Game Mode: ☒ Single Game ☐ Swiss ☐ Round Robin

Reconnection: ☒ Create ☐ Enter (a Room)

Activation: ☒ No ☐ Yes

Connection Mode: ☐ file I/O ☒ socket I/O

Debug Mode: ☐ No ☒ Yes

Automatic Next Game: ☐ No ☒ Yes

#Games to Play: 4

Move First: ☒ First ☐ Second

Move First Alternation: ☐ No ☒ Yes

Time Setting: ☒ Total Time ☐ Time per Move

Time Limit (sec): 900

Times of Repetition: 3

Initial Position Assignment: ☒ No ☐ Yes

Path:

Server IP: 120.126.145.147

Buttons: Read Previous Setting, Choose File, Confirm, Cancel

Figure 5: Create a room.

Enter a Room

- **Reconnection** must be “Enter (a room)”.
- You can only set the **Activation**, **Connection Mode** and **Debug Mode** as mentioned above.
- Click “Confirm” and then display the room information set by the room owner, as shown in Figure 6.
- Choose “**Room ID**” in “**Room information**” window to start a game.

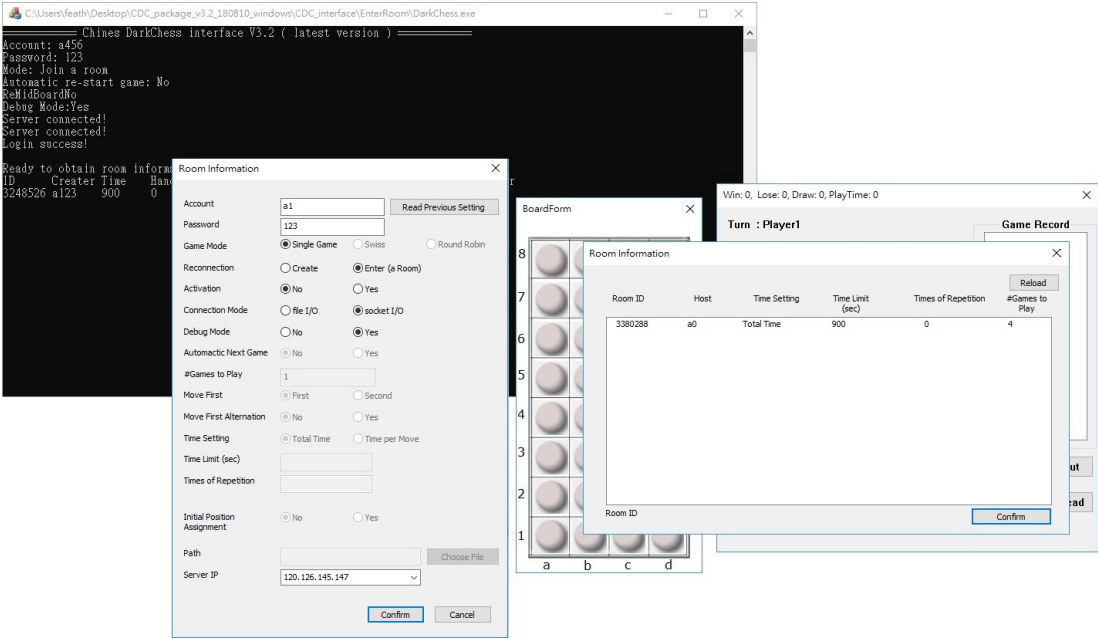


Figure 6: Enter a room.

2. FILE I/O MODE (WITHOUT PONDERING)

2.1 Notations

Piece types

- The letters '**K**', '**G**', '**M**', '**R**', '**N**', '**C**', and '**P**' represent the king, guard, minister, rook, knight, cannon, and pawn of red pieces.
- The letters '**k**', '**g**', '**m**', '**r**', '**n**', '**c**', and '**p**' represent those of the black pieces.
- The letter '**X**' represents an unrevealed/hidden piece.
- The letter '-' represents an empty square.

Board configuration

- The letters '**a**' to '**d**' from left to right for columns
- The numbers '**1**' to '**8**' bottom up for rows

For example, in Figure 7(a), a black pawn (labelled by **p**) is on square **c3**, and an unrevealed piece (labelled by **X**) is on square **b4**. In Figure 7(b), square **a4** is empty and is labelled by -, and the red king (labelled by **K**) is on square **d3**.

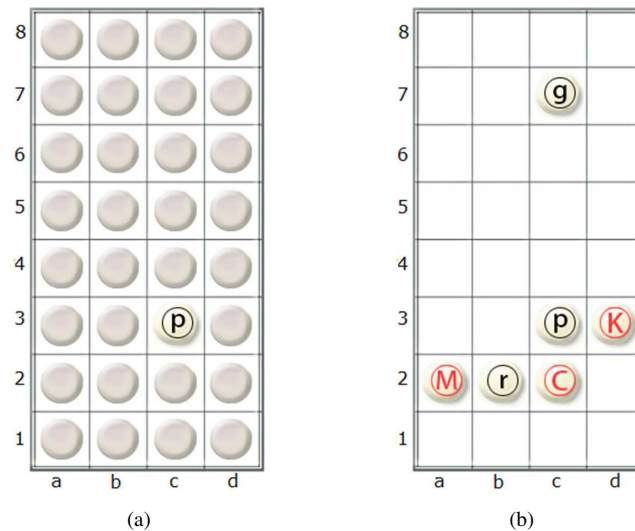
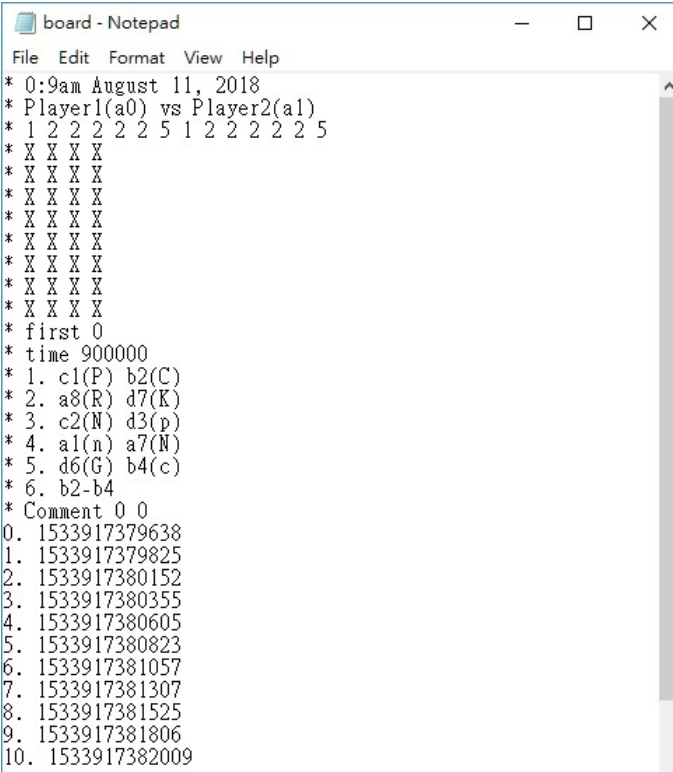


Figure 7: Notations of piece types and the board.

2.2 board.txt

Each move is recorded in the file "board.txt" from the first ply, as shown in Figure 8. For each move, "search.exe" reads the current board from board.txt and then searches the best move.



```

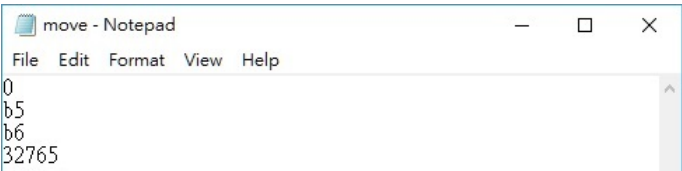
board - Notepad
File Edit Format View Help
* 0:9am August 11, 2018
* Player1(a0) vs Player2(a1)
* 1 2 2 2 2 2 5 1 2 2 2 2 2 5
* X X X X
* X X X X
* X X X X
* X X X X
* X X X X
* X X X X
* X X X X
* X X X X
* X X X X
* first 0
* time 900000
* 1. c1(P) b2(C)
* 2. a8(R) d7(K)
* 3. c2(N) d3(p)
* 4. a1(n) a7(N)
* 5. d6(G) b4(c)
* 6. b2-b4
* Comment 0 0
0. 1533917379638
1. 1533917379825
2. 1533917380152
3. 1533917380355
4. 1533917380605
5. 1533917380823
6. 1533917381057
7. 1533917381307
8. 1533917381525
9. 1533917381806
10. 1533917382009

```

Figure 8: board.txt

2.3 move.txt

After the search finishes, the best move is output to “move.txt”. The move may be either moving or flipping a piece, as shown in Figure 9 and Figure 10 respectively .

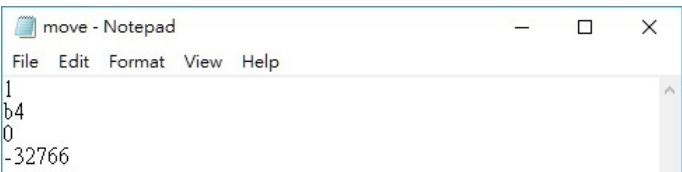


```

move - Notepad
File Edit Format View Help
0
b5
b6
32765

```

Figure 9: move.txt for moving a piece.



```

move - Notepad
File Edit Format View Help
1
b4
0
-32766

```

Figure 10: move.txt for flipping a piece

3. SOCKET I/O MODE (WITH PONDERING)

The package **CDC_client.zip** includes:

- ClientSocket.h, ClientSocket.cpp, Protocol.h and Protocol.cpp – the client protocol
- myai.h, myai.cpp – the random-playing program in Section 1
- main.cpp, main_clear.cpp – to connect myai and the client

To connect a game-playing program to the client, you only have to modify a few lines in **main.cpp**. These lines are marked by “// **todo:**” in **main_clear.cpp**. Then, compile with the following command in case you use g++.

```
g++ -o search.exe main.cpp Protocol.cpp ClientSocket.cpp myai.cpp
```

You can replace myai.cpp with your game-playing program, **YourAI.cpp**, and the compile command is as follows.

```
g++ -o search.exe main.cpp Protocol.cpp ClientSocket.cpp YourAI.cpp
```

The newly compiled file, **search.exe**, overwrites the **search.exe** file in the **Search** folder. Thus, you can connect to the server with your program by execute **DarkChess.exe**, as described in Subsection 1.1.

The class and functions used are described in the following subsections.

```
enum PROTO_CLR {PCLR_RED, PCLR_BLACK, PCLR_UNKNOWN};

class Protocol
{
public:
    Protocol();
    ~Protocol();
    void init_protocol(const char *ip, const int port);
    void init_board(int piece_count[14], char current_position[32], struct History &history);
    void get_turn(bool &turn, PROTO_CLR &color);
    void send(const char src[3], const char dst[3]);
    void send(const char move[6]);
    void recv(char move[6]);
    PROTO_CLR get_color(const char move[6]);
};
```

3.1 init_protocol

```
void init_protocol(const char *ip, const int port);
```

Connect to the server by inputting the **ip** and **port** of the server via command line or GUI interface. **init_protocol** must be called in the beginning.

```
#include "protocol.h"
int main(int argc, char **argv)
{
    Protocol protocol;
    switch (argc) {
    case 3:
        if (!protocol.init_protocol(argv[1], atoi(argv[2]))) return 0;
        break;
    }
    ...
    return 0;
}
```

3.2 struct History

```
struct History{
    char** move;
    int number_of_moves;
};
```

The meanings of **move** and **number_of_moves** are as follows.

- **move:**
If the ply is a move or capture (e.g., the 2nd ply is moving a piece from a3 to a4), then **move[2]** = "a3-a4".
If the ply is a flip (e.g., the 2nd ply is flipping a red king on c2), then **move[2]** = "c2(K)".
- **number_of_moves:**
The total number of ply.
For example, if **number_of_moves** = 3, we use **move[0]**, **move[1]** and **move[2]**.

If the program resumes to play, you have to restore the history as follows (in the TODO part).

```
struct History history;
protocol->init_board(piece_count, current_position, &history);
for (int i = 0; i < history.number_of_moves; i++) {
    // TODO: restore the history to your program.
}
```

3.3 init_board

```
void init_board(int piece_count[14], char current_position[32], struct History history);
```

After calling **init_board**, you get the initial board settings as follows.

- **piece_count[14]:** The number of pieces that are alive of 14 piece types.
- **current_position[32]:** The value of each element represents the piece type on the board. Notations are described in Subsection 2.1.
- **history:** The history of the game.

For example, in Figure 11, two red ministers (M), a black knight (n), and a black king (k) are revealed. And a red pawn and a black cannon are captured. The values of **piece_count[14]** and **current_position[32]** are:

```
piece_count[14] = {1, 2, 2, 2, 2, 2, 4, 1, 2, 2, 2, 2, 1, 5}
current_position[32] = "XnXXX-XXXXXXXXXXXXM-XXXkXMXXXXXXX"
```

3.4 get_turn

```
enum PROTO_CLR {PCLR_RED, PCLR_BLACK, PCLR_UNKNOWN};
void get_turn(bool &turn, PROTO_CLR &color);
```

turn = true means the first player, and **turn = false** means the second player.

The value of **color** are **PCLR_RED** as red, **PCLR_BLACK** as black, and **PCLR_UNKNOWN** as unknown.

If the game is played from the beginning (that is, all pieces are unrevealed/hidden), you get your turn and color that is set to **PCLR_UNKNOWN**.

If the game is played from midgame, you get your turn and color that may be set to **PCLR_RED** or **PCLR_BLACK**.

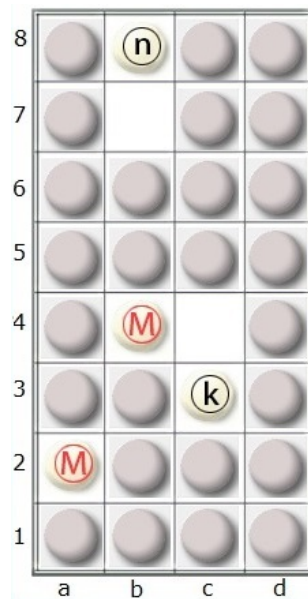


Figure 11: `current_position[32]`.

3.5 send

You may choose one of the following two functions to send your ply.

```
void send(const char src[3], const char dst[3]);
```

If the ply is a move or capture (e.g., move a piece from d5 to c5), then **src** = "d5" and **dst** = "c5".

If the ply is a flip (e.g., flip a piece on d5), then **src** = "d5" and **dst** = "d5".

```
void send(const char move[6]);
```

If the ply is a move or capture (e.g., move a piece from d5 to c5), then **move** = "d5-c5".

If the ply is a flip (e.g., flip a piece on d5), then **move** = "d5-d5".

3.6 recv

```
void recv(char move[6]);
```

move is the ply that the opponent played and sent to you by the server.

If the ply is a move or capture (e.g., move a piece from a3 to a4), then **move** = "a3-a4".

If the ply is a flip (e.g., flip a red king on c2), then **move** = "c2(K)".

3.7 get_color

```
enum PROTO_CLR {PCLR_RED, PCLR_BLACK, PCLR_UNKNOWN};
PROTO_CLR get_color(const char move[6]);
```

This function returns the color of the flipped piece. For example,

```
PROTO_CLR color;
char move[6] = "a8(G) ";
color = get_color(move);    /* color == PCLR_RED */
move[6] = "d6(p) ";
color = get_color(move);    /* color == PCLR_BLACK */
```

4. THE GUI INTERFACE

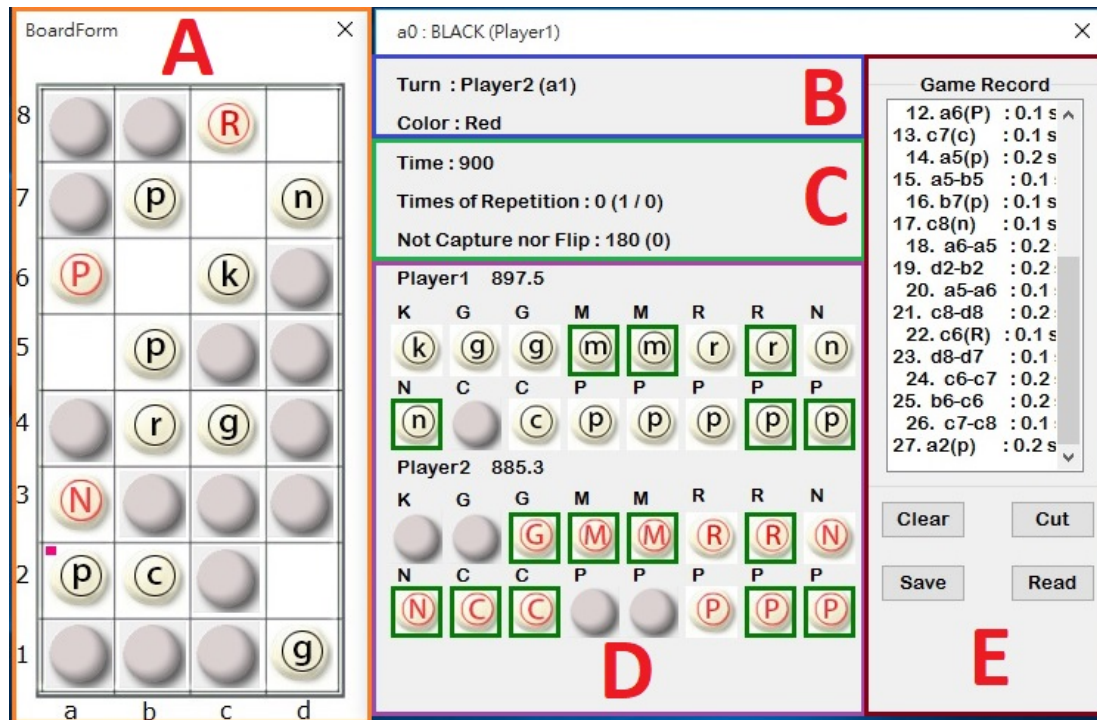


Figure 12: The GUI interface.

The interface includes five zones described below.

A. Board

- Display the current board.

B. Turn and Color

- Turn – Indicate which player to search now.
- Color – The color owned by the current player. Both players know his/her color after flipping the first piece.

C. Rules

- Time setting – the time setting, e.g., 900 seconds to finish a game
- Times of Repetition – Count the number of repeated positions.
- Not Capture nor Flip – Count the number of consecutive moves without capturing and flipping.

D. Remaining Time and Captured Pieces

- Remaining time – the remaining time in a game or in a move, shown at the right side of Player1 and Player2.
- Captured pieces – The revealed pieces are alive, while the unrevealed pieces are captured.

E. Move Information

- Game Record – the moves from the beginning to the current position
- win/loss/draw – The win count, loss count and draw count of games
- Clear button – Clear the current game, and start a new game.
- Cut button – Back to n moves before.
- Save button – Save the current game. The default file is “Search\Saveboard.txt”.
- Read button – Read the game record from the file “Search\Saveboard.txt”.

5. CONTACT INFORMATION

If there are any unclear description about the protocol, please contact:

- Jr-Chang Chen (<http://web.ntpu.edu.tw/~jcchen/>, email: jcchen@mail.ntpu.edu.tw)
- Gang-Yu Fan (imloed10000@gmail.com)
- Ting-Yu Lin (feather115@gmail.com)

The rules and notations of Chinese dark chess are mentioned in the following articles.

- Bo-Nian Chen, Bing-Jie Shen, Tsan-sheng Hsu (2010). Chinese Dark Chess, *ICGA Journal*, 33(2): 93–106. (pdf)
- Jr-Chang Chen, Ting-Yu Lin, Tsan-sheng Hsu (2015). Equivalence Classes in Dark Chess Endgames. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2): 109–122. (pdf)
- Shi-Jim Yen, Cheng-Wei Chou, Jr-Chang Chen, I-Chen Wu, Kuo-Yuan Kao (2015). Design and Implementation of Chinese Dark Chess Programs, *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1): 66–74. (pdf)
- Chu-Hsuan Hsueh, I-Chen Wu, Wen-Jie Tseng, Shi-Jim Yen, Jr-Chang Chen (2016). An Analysis for Strength Improvement of an MCTS-Based Program Playing Chinese Dark Chess. *Theoretical Computer Science*, 644(C): 63–75. (pdf)
- Jr-Chang Chen, Gang-Yu Fan, Hung-Jui Chang, Tsan-sheng Hsu (2018). Compressing Chinese Dark Chess Endgame Databases by Deep Learning. *IEEE Transactions on Games*, DOI: 10.1109/TG.2018.2802484. (pdf)