Data generation for Machine Learning on Game and Puzzle Positions

Shingo Tamai † Sho Imamura † Kazuki Ueno † Shuji Jimbo †
† Okayama University

1. Introduction

In recent years, it has been demonstrated that a powerful game AI can be implemented by combining a well-learned neural network with a conventional search part. The authors focused on Freecell, a single-player card game, and Pentago, a two-person zero-sum perfect information game. It is necessary to create datasets of sufficient quality and quantity to create neural nets for two games. We will also clarify how to create a data set for each and future plans.

2. Freecell

2.1 What is Freecell?

Freecell is a single-player playing card game, in which 52 cards excluding the jokers face up on eight mountains called tableaux are put into a place called home cell by making good use of four spaces called free cell. The purpose is to move from ace to king for each suit (mark on the card). As shown in Fig. 1, the initial board surface randomly divides the card into eight tableaus, and prepares seven tableaus four and six tableaus four. The cards that can be moved are the cards in the heavens (top, bottom in the figure) of each tableau or the cards placed in the free cell. Free cells are four empty cells that can be placed in each free cell. The only cards that can be placed on top of the tableau are the ones with a different color from the heavenly cards of each tableau (A = 1, J = 11, Q = 12, K = 13). According to the above rules, if you move all cards from the initial board to the home cell in order from A to K for each suit, you will win.

2.2 Free cell solver

Randomly create initial board to obtain solutions for various initial boards. The randomly generated initial board surface is given to the free cell solver to find the solution. The solver used limits the search space with a heuristic function and performs depth-first search [1] [2]. The following four parameters were used for the function: “the number of moves to the position (T)”, “the number of cards not moved to home cell (N)”,

“the number of blocked cards (B)”, and “the number of cards that are not (M) ”. Here, as shown in Fig. 2, when there are multiple cards of the same suit in the same tableau and the card with the higher number is on the top, the card with the higher number is called the blockade card [4].
This time, heuristic function is defined as follows.

1500*T+alph1*N+alph2*B+alph3*M

When moving the solver, five conditions of alph1, alph2, alph3, delt (tolerance), and the upper limit of the number of steps are given, and a move search is performed so that the heuristic function does not exceed the predetermined upper limit of the cost. Specify the initial value of the upper limit of the cost so that it does not exceed the heuristic function of the initial board, and if a solution with a cost smaller than the upper limit of the cost cannot be found, raise the upper limit of the cost by a fixed number (tolerance) and solve again within the range. This searches for possible solutions of. Even if the upper limit is raised to some extent (this time is 20 times), if a solution with a shorter number of actions than the upper limit is not found, give up solving. This time, we use a 48×2 two-dimensional array table as a speed-up method to find the next move, and it is possible to find the move from tableau to tableau just by drawing the table.
The contents are 48 pieces in total from A to Q of each suit (K does not need to be registered because it cannot be placed on other cards), and the number of the tableau that can be placed on the current board is registered. After setting the table for the initial board, you can find the moving hand at high speed just by repeating the minor correction for each hand and drawing the table for each non-empty tableau heaven card. Zobrist hash is used to eliminate the same situation. In addition to the 63-bit hash value, the registered contents include the 16-bit minimum number of steps up to that stage (total 10 bytes). The solution output is expressed in hexadecimal notation for the board. One move is expressed in hexadecimal notation with *0* to *3* for each free cell, *4* to *b* for each tableau, and *c* to *f* for each home cell, and is represented by two characters that combine the source and destination (e.g. *6b*. If the card at position *6* is moved to position *b*).

2.3 Training data

The purpose of this time is to create a data set for deep learning with sufficient quality and quantity, and let the neural network perform supervised learning [3]. The neural network learns what kind of action is effective for the current board. Therefore, the data given is “current board + start”. By giving the solution obtained by the solver and the

initial board surface, the board surface for each move from the initial board surface to the state where everything has returned to the home cell is reproduced, and combined with the move to make one data.

2.4 Experiment

Described here are the experimental results regarding the time required to create the dataset. This time, we created a data set of 2500 solutions for the initial board. As the value of each variable, the value obtained early by several experiments was adopted. Also, n is specified as an argument and the size of the hash table is specified. The size is set to be $10 \times 2^{n+12}$ bytes when n is given (e.g. $10 \times 2^{26} \approx 6.4 \times 10^8$ bytes = 640MB when n = 14). The specifications of the machine used are shown below.

table 1 : Specifications of the machine used for the experiment

CPU:	Intel(R) Xenon(R) CPU E5-2630 v2 @ 2.60GHz × 2
Memory:	256GB
OS:	Cent OS Linux release 7.5.1804
compiler	gcc 4.8.5

With the above machine, the operation for obtaining the solution for 125 initial boards and creating a data set was performed in 20 parallels, and it took about 12 minutes to obtain a total of 2500 solutions for the initial boards. It was. The average time taken to find a solution for one initial board was about 5.8 seconds. In the solver of Paul.[4], the average time to find the shortest procedure for one game is 39.9 seconds. Although Paul'S solver requires the shortest procedure, it cannot be a direct comparison, but considering that the purpose is to create a large amount of data necessary for learning, this method was sufficient.

2.5 To get a better solution

To improve learning accuracy, it is desirable that the solver find a solution that is as close to the shortest procedure as possible. Since the solver currently used is based on the depth-first search, if the longest procedure is specified, it will be output as soon as a solution is found within that range, so the solution is long, close to the upper limit of the number of operations limited by the algorithm. It tends to be a thing. The concept of a method for causing the solver to find the shortest procedure or a solution close to the shortest procedure is described below.

2.5.1 Useless hand removal

The following operations to save unnecessary steps from the obtained solution procedure

should be incorporated because they can be easily implemented.

A hand that moves a card at location *a* to another location b is represented by (*a, b*). If the hand (*a, b*) appears, then it is to achieve one of the following three purposes.

- Because *a* is the top of the tableau and a card other than the card in *a* is placed at *a*. Therefore, a hand of the form (*x, a*) then appears.
- *a* is the top of tableau, and the card that appeared due to the movement of the card in a was moved to another location. Therefore, a hand of the form (a, x) then appears.
- The destination *b* to which the card in *a* has moved is the tableau heaven, and another card is to be placed after the card in *a* is placed in *b.* Therefore, the (*x, b*) hand then appears.

If a hand of the form (*b, y*) appears before these objectives are achieved, then the first (*a, b)* hand was wasted, and the first (*a, b*) hand from the solution procedure. By omitting and changing (*b, y*) to (*a, y*), one short solution procedure can be obtained. Continue this operation until the number of solution steps is no longer shorter.

In order to find the shortest procedure in the depth-first search, a solution is obtained, and after performing the above-described waste-saving operation on the solution, the limitation value of the procedure is set to a value one less than the procedure of the obtained solution. You can change it and restart the search. If the search is completed in this way, the procedure finally obtained is the shortest procedure. However, it may be necessary to terminate the search before it is completed due to restrictions on computational resources such as hash table size and execution time. Even in that case, it is expected that a procedure close to the shortest procedure can be obtained by taking a large enough computing resource.

2.5.2 Acceptable heuristics

We think that the execution time can be significantly shortened by devising the ordering of choices in the depth-first search and creating an excellent permissible heuristic function. Here, a permissible heuristic function is a function that assigns a non-negative integer as a function value to a situation given as a variable value and whose function value is guaranteed to be less than or equal to the shortest move from the situation of variable value to victory Call. If the permissible heuristic function value of the position ***p*** in the search is larger than the difference obtained by subtracting the work up to ***p*** from the limited work, the search after ***p*** can be aborted and backtracking can occur. Therefore, the admissible heuristic function is superior as the difference between the shortest move from the variable value stage to the victory and the function value is smaller. We are currently working on the improvement of the free cell solver according

to the concept described above, and are working on the creation of higher quality and larger data sets.

2.5.3 Strongly connected component decomposition

In general, if the directed graph G includes strongly connected components with edges, cycles exist in the strongly connected components; therefore, if G is a strongly connected component and has edges, then G includes cycles. The deadlock graph of position n of the free cell is *D(n)*, and the card on the heaven side of one soot cycle at *D (n)* is a. Let *D ′(n)* be the directed graph obtained by removing all the closed edges (*a ′ , a*) that are removed as a result of moving, where *a ′* is directly adjacent to the ground side of a on the tableau sequence. In this case, if the number of strongly connected components of *D′ (n)* that have edges is k_0(n) , then

$$m_e(n) \geq m_0(n) + k_0(\mathrm{n})$$

However, if *D′ (n)* consists of a few large strongly connected components, then there may be a large number of strongly connected components with edges in the subgraph of *D′ (n)*. Therefore, a small number (1, 2, 3) of edges are randomly removed from *D′ (n)* to create a directed graph$D_1, D_2, \ldots D_j$, and the number of edges with strongly connected components is calculated. Represented by k_0(n),

$$m'_e(n) = m_0(n) + \max_{i=0} k_i(n)$$

define $m'_e(n)$.

3. Pentago

3.1 What is Pentago?

This is a two-player battle board game that is classified as a two-person zero-sum finite definite complete information game. In Pentago, the board surface of 6 × 6 cells is divided into 4 small disks of 3 × 3 cells. The player puts his or her stone (white or black) and then rotates one of the four slabs 90 degrees. The victory condition is the same as for Gomoku, and the player who wins 5 stones of his or her color in a row vertically, horizontally, or diagonally first will win. The victory judgment is made when the stone is placed and when the small board is rotated. If the players can't arrange their own 5 stones and there is no space for stones to be placed, or when either player spins the board

and both players have 5 stones at the same time, it is a draw.

3.2 Data set

For Pentago, the complete analysis data up to the 18th hand has been published by Irving so far. The information about which one wins when each player takes the best action from that point on that board is called win / loss information for that board. By randomly arranging the stones and comparing the board surface with the complete analysis data, the win / loss information of the board surface is obtained. A set of boards on which one stone is placed is called slice1, and up to slice18 is included in the complete analysis data, but there is a problem. Nowadays, when creating slice13 or more training data samples on a typical workstation, the amount of win / loss information for that slice positions in the full analysis data is too large to fit in main memory. Therefore, consider compressing the data to a size that can be handled by a general workstation.

3.2.1 Data format

This section describes the format in which the board of Pentago is represented by a 64-bit integer. This format is based on Irving and will be referred to as Irving format below. Order each small board and each square in the small board, and assign "0", "1", "2" to "no stone", "black stone", "white stone" in each small board, respectively. The board surface in each board is represented by a ternary number. At this time, the i-th cell corresponds to the i-th digit from the least significant of the ternary number. The non-negative integer represented by this ternary number can be represented by a binary number of 15 bits or less. Furthermore, when the non-negative integers representing the surface of the 1st, 2nd, 3rd, and 4th small boards are represented by a, b, c, and d, respectively, an Irving-form nonnegative integer representing the entire board is given.

$$\mathrm{w(s)} = \mathrm{a} + \mathrm{b} * 2^{16} + c * 2^{32} + d * 2^{48}$$

by expressing with, it can be expressed by a 63-bit binary number.

The data file of is divided by the distribution of stones in the four discs called a section. A set consisting of all the positions with the same section s is also called a section s, and the position whose section is s is called the section s. However, not all slice win / loss information that satisfies the above equation is included in the slice n data file.

Only the section s that satisfies the condition that there is no section s 'that satisfies w(s') < w(s) among the positions obtained by subjecting the section s to 8 inversions and rotations is slice. Included in n data files. Furthermore, the standard section is the position of the minimum section obtained by subjecting it to eight inversions and

rotations for any position, and rotating the four plates so that the ternary number represented by each plate is minimized. We call the arrangement of the stones in each small plate included in the standard type position (the ternary number that represents the stone) the standard type of small plate. Since each small plate has 4 types by rotation, the total win / loss information of each slice is Every 256 pieces of winning / losing information having a 4D structure about rotation can be placed in a 4D space having a dimensional axis corresponding to each boards. Irving also defined blocks by dividing each dimension by 8 as follows. For a given set of four numbers (k_1, k_2, k_3, k_4),

$8k_1 \leq i_1 < 8(k_1 + 1)$, $8k_2 \leq i_2 < 8(k_2 + 1)$, $8k_3 \leq i_3 < 8(k_3 + 1)$, $8k_4 \leq i_4 < 8(k_4 + 1)$

A set consisting of all the standard positions corresponding to a set of numbers (i_1, i_2, i_3, i_4) that satisfy is called a block (k_1, k_2, k_3, k_4). All winning / losing information contained in the block is stored in the main memory. When expanded to, the number of positions of the standard type contained in the block is at most 8^4= 4096, and each has 512 bits, that is, 256 pieces of winning / losing information represented by 64 bytes, so the maximum storage capacity is occupied. 4096 64 = 262144 bytes, or 256KB.

table 2 : Storage capacity of winning / losing information of slice n

n	Number of bytes during compression	Number of bytes after expansion
10	376,550,807	3,998,676,992
11	3,162,876,487	15,073,419,776
12	6,590,531,550	57,817,397,504
13	38,162,720,858	177,120,575,744
14	66,400,234,992	545,180,925,184
15	294,225,040,548	1,430,185,318,400
16	461,162,080,723	3,732,582,001,792
17	1,475,380,615,039	7,847,311,093,824
18	1,954,957,518,434	16,251,783,918,912

3.2.2 Data creation method

The training sample used in this study is an image of the winning and losing information in the main memory after expansion for each slice.

Was created on the hard disk and divided into 64GB each, and partially loaded into main memory and used. As can be seen from Table 1, slice 18 has 237 pieces of winning / losing information. It is divided into files. These files, or the winning / losing information contained in the files, is called a divided image. The image in the main memory here does

not have the concept of a block and is 512 bits (64 bytes).), The win / loss information for each standard type is arranged on a four-dimensional structure with the standard type of each board as a dimension. To create this image file, the algorithm of the win / loss information retrieval program should be diverted. Can be created easily. For the Pentago position as a training sample, for each slice from 10 to 18, 230 sets were created according to a uniform distribution with 230 each, and 10 sets were created, for a total of 90 sets. Using /dev/urandom, we used Fisher-Yates shuffle to generate a black and white stone arrangement based on random numbers, and a hash table to eliminate duplicates. In the Irving format, the 15th, 31, 47, and 63th bits are unused, and up to 4 bits of information can be inserted. The information of winning (1) or not (0) is inserted, and the information of white winning (1) or not (0) is inserted in the 31st bit. Be done. As a training sample, after creating a binary file F in which Irving formats are arranged in 8-byte units, it is necessary to insert win / loss information as a label into each sample in the file. After reading into the main memory as an array A, the divided images of the slices to which the samples in F belong are read one by one. It is possible to insert the win / loss information into all Irving formats in A by repeating the operation of inserting the win / loss information into the Irving format for all the divided images. The processing of 10 sets of sample positions was completed in about 18 hours and 40 minutes in slice 18, and in about 11 hours in slice 17. Each dataset created as described above is called an original dataset. The percentages of the three types of win / loss information included vary greatly for each slice. Table 2 shows the percentages of the three types of win / loss information included in the first set of the original datasets for slices 15 to 18.

table3 : Content ratio of win / loss information for each slice

slice	draw	Black win	White win
15	0.128	0.207	0.666
16	0.059	0.865	0.076
17	0.096	0.203	0.701
18	0.050	0.861	0.089

As you can see from this table, it is clear that the percentage of wins in the turn is very high in all slices; in detail, the tendency is stronger in the black turn than in the white turn. In this study, we mainly used the experimental data set created by extracting samples from the original data set so that the numbers of three types of winning / losing information were equal.

3.3 Retrograde analysis

Since the Irving's complete analysis data includes only slice 18, the winning and losing commercial methods after the 19th move cannot be obtained. This problem can be solved by performing a retrograde analysis. The retrograde analysis is described as follows. Enumerate the boards that can be reached by the end position that ends. Then, the winning / losing result is confirmed in order from the listed end nodes, and the result is used to judge the winning / losing of the parent node. The regression analysis is performed as follows. Stations that have won games from all stages Extract a set of faces. The phasing side wins, the phasing side loses, and the phasing side loses. The position that is a draw is the draw position. Perform the following operations from the set of extracted positions.

1. Extract one aspect from the set.
2. All the transition source positions of the extracted positions are the target positions.
   If there is at least one winning position at the transition destination, the target position is the winning position
   If all the transition destinations are losing positions, the target position is the losing position
   If there is at least one draw position at the transition destination, and if all the remaining transition sites are losing or draw positions, the target position is the draw position.
3. Add the transition source of the newly won position to the set.

When the above operation is performed and the set is empty, the analysis ends. After the analysis, if the initial stage is a winning stage, it is a win on the turn side, if it is a losing stage, it is on the opponent side, and if it is a draw stage, it is a draw.

4. Summary

This time, I described how to create learning data for deep learning of FreeCell and Pentago. It is considered that GPU or MPI should be used to expect further speedup in the future. Future prospects include finding a better heuristic function for FreeCell, and creating a Pentago player by learning using these data for Pentago.

References

[1] Elyasaf, Achiya, Ami Hauptman, and Moshe Sipper. "GA-FreeCell: Evolving solvers

for the game of FreeCell." Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[2] Elyasaf, Achiya, Ami Hauptman, and Moshe Sipper. "Evolutionary design of freecell solvers." IEEE Transactions on Computational Intelligence and AI in Games 4.4 (2012): 270-281.

[3] Chan, Chris. "Helping Human Play Freecell."

[4] Paul, Gerald, and Malte Helmert. "Optimal solitaire game solutions using A* search and deadlock analysis." Ninth Annual Symposium on Combinatorial Search. 2016.