Playing PuyoPuyo: Two Search Algorithms for Constructing Chain and Tactical Heuristics

Kokolo Ikeda, Daisuke Tomizawa, Simon Viennot and Yuu Tanaka

***Abstract*—Tetris is one of the most famous tile-matching video games, and has been used as a test bed for artificial intelligence techniques such as machine learning. Many games have been derived from such early tile-matching games, in this paper we discuss how to develop AI players of "PuyoPuyo". PuyoPuyo is a popular two-player game, and where the main point is to construct a "chain" longer than the opponent. We introduce two tree search algorithms and some tactical heuristics for improving the performance. We were able to reach an average chain length of 11, notably higher than that of the commercial AIs.**

I. Introduction

Tetris is one of the origins of tile-matching video games [8]. It was published in 1985 by Alexey Pajitnov and soon became one of the most popular video games. There are many variations of it [7] and many new commercial games have been derived from it. Tetris is also a good test bed for artificial intelligence (AI) techniques, such as machine learning [1] and direct policy search [4]. Although most variations for human players have a real-time feature, in many AI papers Tetris is simplified to a turn-based game or a Markov decision process (MDP) [5]. In a turn-based context, the goal is usually to create an AI player which can "survive" long turns, because Tetris is a one-player game where the original goal is also to survive as long as possible, or to delete a given number of (for example 25) lines.

In the numerous existing tile-matching video games, PuyoPuyo is remarkable for having been developed as a two-player game. The goal of this game is not to survive nor to delete a given amount of blocks, but to attack the opponent by sending garbage blocks on his board. In order to send more garbage blocks, it is necessary to prepare and execute a long "chain" of eliminations of blocks. Then, construction of chains is the most important and most interesting issue of this game.

PuyoPuyo was published in 1991 by Compile, a Japanese developer. Since then, many series and variations have been released. For example PuyoPuyo-7 was released in 2009, and the series sold more than 10 million copies in total [6]. However, PuyoPuyo-2 is still the most popular version, because of its well established rules. There are many skillful human players, and the best ones are greatly respected for their fast decision, accurate control, shrewd tactics, effective and creative construction of chains.

On the other hand, the AIs of PuyoPuyo-2 are very weak, at least for advanced players. The average length of chains of AIs is only around 3 to 4, despite that of experts is around 11 to 12. Then, though playing with the AI of PuyoPuyo-2 is still exciting because of its real-time aspect, almost all advanced players prefer playing with other human experts.

Though the performance of commercial AIs has not improved much, some clones with comparatively strong AIs have been developed [10] [9]. The average length of chains of these AIs is around 9 to 10, but their algorithms to construct chains have not been published yet in academic papers. In this paper, we introduce two different algorithms to construct long chains, and some tactical heuristics to play. The effectiveness of the proposed methods is evaluated by several experiments.

K. Ikeda, D. Tomizawa, S. Viennot and Y. Tanaka are with Japan Advanced Institute of Science and Technology, email: kokolo@jaist.ac.jp

II. The Rules of Synchronized PuyoPuyo-2

In this section, we introduce the elementary rules of PuyoPuyo series. As there are many variations in this series, and also many minor and optional rules, we focus on the core rules of PuyoPuyo-2.

Further, though the original PuyoPuyo-2 is a real-time game, we use a turn-based version in this paper. For human players, of course it is very important issue whether the game is timed (real-time game such as FPS) or untimed (turn-based game such as chess). But for AI players, the issue is not so important except the computational time is limited. So we employ a turn-based version for reasons of simplicity and reproducibility, like many papers dealing with untimed Tetris [1] [4]. PuyoPuyo-2 is a two-player game, so we assume that the actions of the players are done in a synchronized manner, in other words, the two players play their actions at once in one turn.

Considering the opponent's strategy is fixed, the environment can be described as a MDP.

A. State : boards, blocks and controlled blocks

Each player has his own 6×13 board (sometimes called "field"), so there are two boards in a game (Figure 1(a)). Each cell of a board is empty or filled by a colored block (called **Puyo**, (b)). Normally four colors are used. A pair of colored blocks (PuyoPuyo) is given to be controlled over the board (c), and the next two pairs (four colored-blocks to be given) are pre-announced (d). All such information is available to both players.

B. Action : moving left/right and rotations

If a pair is given to a player, he can move the pair left and right, and rotate it 90 degrees, any number of times in a turn (Figure 1(c)). Since the number of possible positions for the pair is 5 (horizontal case) or 6 (vertical case), the number of actions is 22 at max. Each player decides the position

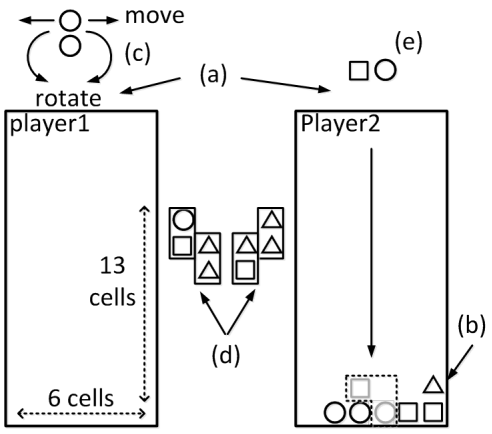

Fig. 1. The states and actions of PuyoPuyo. There are two boards, and a player owns his board.

and rotation, and drops the given pair (Figure 1(e)). Different from Tetris or Dr. Mario, the two blocks are not bonded, each block falls to the surface. When both cells to be dropped are not empty, such action is not accepted. The final decision is observed by both players, but the moving and rotating is hidden from the opponent.

C. Transition (1) : elimination of blocks

If four or more blocks of a same color are connected horizontally or vertically, they are eliminated and disappear from the board (Figure 2(a)). In addition, garbage blocks (to be explained) connected to the disappeared blocks are also eliminated. If some blocks disappear, no pair is given to the player at the next turn, else, a pair of blocks with pre-announced colors is given to the player, and more two colors are randomly fixed and pre-announced.

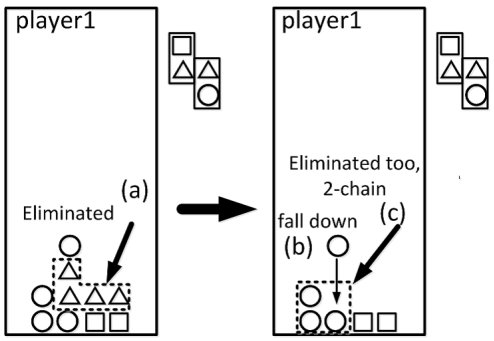

Fig. 2. State transition of PuyoPuyo. Elimination is caused (left) and 2-chain is executed by this elimination (right).

D. Transition (2) : free fall and chain

When some blocks disappear, there may be some blocks over some empty cells. Such floating blocks fall to the surface (Figure 2(b)). At the next turn no pair is given to control, but by this fall some new groups may be eliminated by the rule Transition (1) (Figure 2(c)). Such cascade of elimination is called a "**chain**". Figure 3 shows a simple and a complex example of a 5-chain.

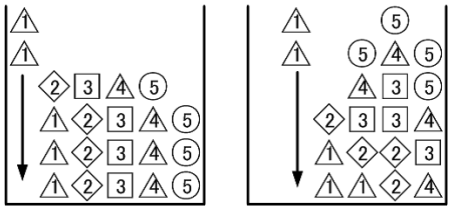

Fig. 3. Examples of a 5-chain, simple one (left) and complex one (right). The blocks with number 1 are eliminated at first, these with number 2 next.

E. Transition (3) : garbage blocks and cancellation

A chain produces an amount of "garbage blocks" (sometimes called Ojama) to attack the opponent. When the numbers of eliminated colored-blocks (except garbage blocks) are $\{a_1, a_2, ..., a_n\}$ from the first to the last, the amount is defined by $3n(n-1) + \sum_{i=1}^{n}(a_i - 4)i$. For example in the two cases of Figure 3, the amount is 61. After a chain is finished, all garbage blocks are *sent* to the opponent's board, and fall at once on the surface at the end of the next turn.

As it costs n turns to finish a n-chain, the number of possible opponent's actions till the fall is $n+1$, after he notices a chain is started. Garbage blocks cannot control it in the player side if it start a chain.

An essential rule "cancellation" of garbage blocks was introduced at PuyoPuyo-2. When both players executed chains and produce garbage blocks, only the difference of them are sent. For example, assume that playerB started a 5-chain and produced 60 garbages, which are normally fatal amount. So playerA noticed the matter, constructed and executed a 6-chain and produced 90 garbages. In this case, 30 $(= 90 - 60)$ garbages fall only on the playerB's board.

In the first PuyoPuyo, it was important to construct a 5-chain as rapidly as possible, because of the lack of this rule. But since the introduction of this rule in PuyoPuyo-2, it became more important to construct a longer chain than the opponent.

F. Reward : The end of the game

If the opponent's board is filled with blocks, the player wins the reward 1.0, and loses in the opposite case. Although it occurs rarely, the game is draw and the reward is 0.5 if both boards are filled at once.

There are many minor differences between this synchronized version and the original PuyoPuyo-2, which are summarized at appendix. We believe almost the same methods as proposed in this paper can be used on the original version. The research platform in C# of this synchronized PuyoPuyo-2 is now ready, and will be published in a website.

III. Potential Maximization Search for Chain Construction

As already noted, many abilities are required to win the game both for human players and AIs; making the decision

quickly, controlling the given pair accurately, watching and analyzing the opponent's board, making a faint, responding to it correctly, preparing a long chain and executing it in an adequate timing, and so on. In this paper, we mainly discuss how to construct a long chain with search algorithms.

The main difficulty of constructing a chain is the randomness of colors of the given pairs. If we could choose the colors arbitrarily, it would not be so difficult to construct a 19-chain (a 20-chain requires 80 blocks at least, so it is impossible on a 6×13 board), but actually 13 or 14-chains are almost the limit with random colors and enough to fight.

In this section, we introduce and evaluate a tree search method for chain construction. The colors of the given pair are known, and the colors of the two next pairs are pre-announced, so the player can predict and evaluate the possible future states to depth three, with no randomness except the opponent's attack. The procedure used in this paper is as follows (see Figure 4):

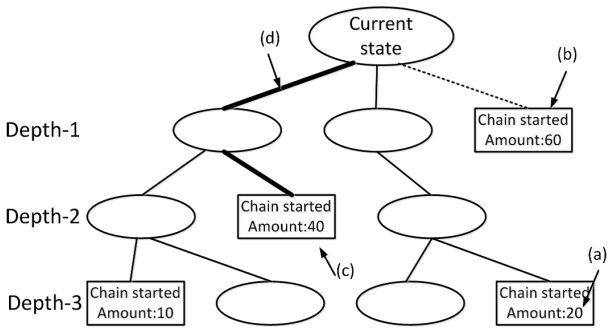

Fig. 4. Search tree of potential maximization search.

1) All the possible future states are represented as nodes, until a depth of three or until some blocks disappear (in other words, a chain is started). As the maximum number of actions is 22, the maximum number of the depth three nodes is $22^3 = 10648$. If only $m = \{1, 2\}$ actions are possible until the garbage blocks fall by the opponent's attack, the depth is limited to m.
2) The number of garbage blocks to be produced is counted (a), for each node where a chain is started.
3) The node which produces the most garbage blocks is selected (b). If at least one of the following conditions is satisfied, the action to the node is employed and the chain is started.
   - If the opponent already executed a chain.
   - If the remaining space of the board, or the number of empty cells, is smaller than a threshold T_{space}.
   - If other optional conditions described in Section VI are satisfied.
4) If such conditions are not satisfied, the node which produced the most garbage blocks in depth-2 or depth-3 is selected (c), and only the first action/edge of sequence to the node is selected (d).
5) If there is no node in depth-2 and depth-3 where a chain is started, some evaluation function of the blocks patterns is used to select the best node. In the experiments of this paper, the number of connections between blocks of a same color is maximized.

The key point of this procedure is to keep and maximize the potential strength by avoiding premature attacks (step 3, 4). We call such algorithm **potential maximization search** (PMS) in this paper. By personal communications, we know that some clones with comparatively strong AIs [10] [9] use a variation of PMS.

A. Pseudo Depth-4 Search

If the colors of more than three pairs were known and deeper search could be done, it is imagined that the performance of PMS would be improved. Especially in the above simple PMS, the potential of a node can be evaluated only when the chain is started, then often *underestimated*. In other words, even if a chain is ready to be started like in Figure 3, one more block of a color to start the first elimination must be found in given colors, to estimate the node correctly. By these reasons, simple PMS sometimes fails to keep some critical cells empty for starting a chain later.

In this paper we introduce a **pseudo depth-4 search** heuristic for this problem. In this method, the depth-3 leaf node (where no chain is started) is expanded to depth-4 nodes, assuming that a pair of blocks of a (same) color is given after the pre-announced ones. 22×4 nodes can be expanded from a depth-3 node, which implies that at most $22^4 \times 4$ depth-4 nodes are evaluated. The other steps for decision making are the same to the original PMS described in the last subsection.

This method selects the "luckiest" case, then sometimes the nodes are *overestimated*. However, at least in our experiments, the performance is greatly improved. One more difficulty of this method is its high computational cost. The computational cost of PMS is almost proportional to the number of nodes to be evaluated, then that of pseudo depth-4 search is 88times higher than the original depth-3 search. We confirmed that the maximum search time is under 1.0 second, in our environment using C#.Net and Intel Core i5-2310. So we consider pseudo depth-4 search can be employed, but pseudo depth-5 search cannot be employed, even after some code optimizations.

B. Performance Evaluation of PMS

In this subsection, we show how a long chain can be constructed by the PMS. In order to know the pure performance of PMS, no tactical heuristic is introduced in this experiment, and the opponent is prohibited from executing any attack. Depth-2 search, depth-3 search and pseudo depth-4 search are compared, and parameter $T_{space} \in \{8, 16, 24, 32\}$ is tested. 1000 independent games have been done for each settings. The performance is evaluated by the average length of chains and its standard deviation, and summarized in Table I.

The performance is clearly improved when the depth increases. The degree of improvement is about one (or more) longer chain, per one more deeper search. It is interesting to note that the best T_{space} depends on the depth of the search. In PuyoPuyo, more space is required for a longer chain, but

TABLE I
PERFORMANCES OF PMS, THE AVERAGE (AND STD.) LENGTH OF CHAINS IN 1000 GAMES, AVERAGED SEARCH TIME

search method	$T_{space} = 8$	16	24	32	CPU time [msec]
depth-2	5.29 (1.58)	6.07 (1.70)	6.29 (1.73)	**6.49** (1.75)	25.1
depth-3	6.58 (2.15)	7.39 (1.99)	**7.57** (1.75)	7.31 (1.58)	41.2
pseudo depth-4	7.69 (3.19)	**9.43** (1.91)	9.13 (1.68)	8.57 (1.39)	100.2

the management of narrower empty space is more difficult for weak AIs/players. Then, a more robust strategy (big T_{space}) is better for depth-2 search, and a more aggressive but risky strategy (small T_{space}) is better for pseudo depth-4 search.

Figure 5 shows the histogram of the chain lengths executed by pseudo depth-4 search with $T_{space} = 16$. The most executed length is 10, but the distribution is a bit wide, from only 3 to 14.

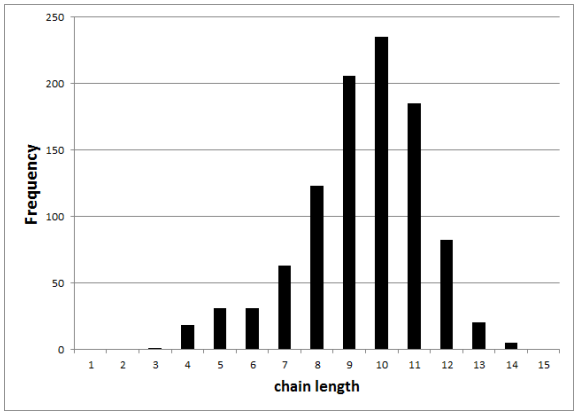

Fig. 5. Histogram of chain lengths achieved by PMS, in 1000 games.

The result is reasonable. However, the average length of chains is still not sufficient compared to advanced players. Figure 6 (left) is an example of a 10-chain created by hand, and Figure 6 (right) is an example of a 10-chain created by PMS. We can see a big difference between them. Chains created by PMS are often very complex and include many over-four eliminations. In this case the numbers of eliminated blocks are $\{5, 4, 4, 6, 9, 4, 10, 4, 4, 4\}$ and the sum of them is 54, though the sum is only 41 in the case of the left figure. It is necessary to use the space efficiently for constructing longer chains, so PMS is not adequate, if alone. In addition, such a complicated or unnatural form of a chain is usually not welcomed by human players. For these reasons, we propose in the next section a new method to construct more human-like chains.

IV. TEMPLATE MATCHING SEARCH FOR FORMAL CHAIN CONSTRUCTION

In potential maximization search (PMS), the "goodness" of a node (a state or a board) is decided only by the number of garbage blocks to be produced. This method is very simple, and the average length of chains is not so bad, but a more

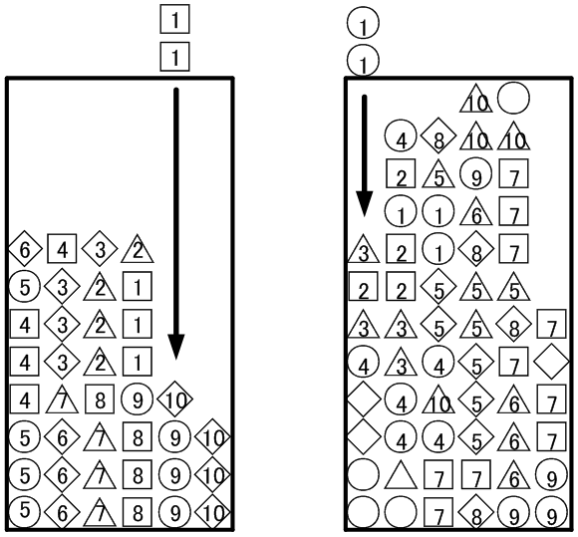

Fig. 6. Examples of 10-chains. Smart one created by hand (left) and dirty one created by PMS (right)

sophisticated method is required for two purposes, to construct longer chains and to construct natural or human-like chains.

This issue is common to AIs for classical board games such as chess and Go. In such board games, the AI player usually searches a game tree by using $\alpha\beta$ method or Monte-Carlo method [3], but such method is often not good at the beginning of the game. Then, an "opening book" that includes many possible variations from the initial state is often utilized in strong AIs. It can be created by experts, gathered from expert records, or optimized by some learning [2]. Also in PuyoPuyo, several "**formal chains**" such as Figure 7 are often used by human players[11].

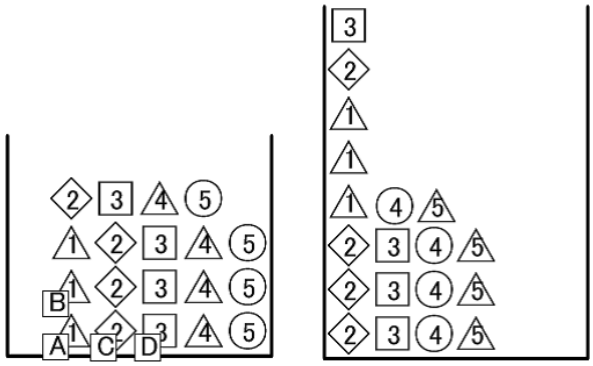

Fig. 7. Examples of formal 5-chain. The simplest but dead-end one (left) and a bit difficult but expansive one (right).

There are several important differences between the opening book of chess/Go and PuyoPuyo, then the conventional methods for creating/using an opening book are not adequate to use directly. At first, the board is common to both players in chess/Go, but is used only by each player in PuyoPuyo. So the beginning moves of PuyoPuyo are not disturbed by the opponent. Next, the PuyoPuyo player cannot select the same

sequences every time because of the randomness of the given colors. Finally, in PuyoPuyo, the actual color itself has no characteristic then some of them can be replaced by different colors (this is similar in the case of Go, white and black stones can be normally exchanged). For example, in Figure 7 (left), the first three triangle-color blocks (tagged 1) can be square-color or circle-color, though they cannot be diamond-color. The number of all variations is 4×3^4, so it is not efficient to prepare all the patterns for strict matching.

In PuyoPuyo, it is not important which color a block is, but it is important whether two colors of blocks are the same or not. Then, we propose a new method to exploit this property. The state of the board will be described by a matrix, that we call an **association matrix**.

A. Overall framework

The overall framework of the proposed method is as follows (see Figure 8 also). We call it **template matching search** (TMS) in this paper. The making of the template is possible by manual labor in two hours. I think using the technique of the data mining.But, there is not culture to take log in puyopuyo.

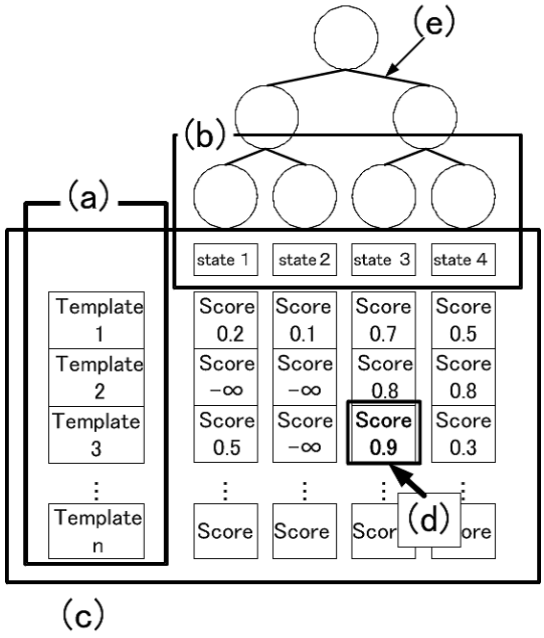

Fig. 8. Framework of Template Matching Search, in depth-2 case.

1) A formal chain such as shown in Figure 7 is represented in the form of an association matrix. We call it the **template matrix** (a). Several template matrices are prepared, before playing.
2) In a game, depth-3 search is done, and each leaf node is represented in the form of an association matrix. We call it the **state matrix** (b).
3) The accordance score between each template matrix and each state matrix is calculated (c).
4) The node with the best accordance score is found (d), then only the first action/edge of sequences to the node is selected (e).

B. State matrix and template matrix

A board is represented by an association matrix, where the actual colors of cells are ignored, and only the relationships between two cells are represented as the following equations (1) and (2). Please note that this is not a 6×13 matrix that represents the actual colors of the board, but an $n \times n$ matrix where $n = 6 \times 13$.

$$S = \begin{pmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nn} \end{pmatrix} \tag{1}$$

$$s_{ij} = \begin{cases} +1, & \text{two cells are filled by a same color} \\ -1, & \text{two cells are filled by different colors} \\ 0, & \text{one of them is empty} \end{cases} \tag{2}$$

The requirement for a formal chain is represented also by an association matrix as the following equations (3) and (4). For example in Figure 7, $t_{AB} > 0$ because cells A and B must be the same color, $t_{AC} < 0$ because cells A and C must be of a different color, and $t_{AD} = 0$ because it's no problem whether cells A and D are the same color or not. The sign of t_{ij} is the most important, further, the absolute value of t_{ij} is appended to express how preferentially they should be placed.

$$T = \begin{pmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{n1} & \cdots & t_{nn} \end{pmatrix} \tag{3}$$

$$\begin{cases} t_{ij} > 0, & \text{if two cells must be the same color} \\ t_{ij} < 0, & \text{if two cells must be different colors} \\ t_{ij} = 0, & \text{if both cases can be accepted} \end{cases} \tag{4}$$

C. Accordance score

Accordance score of each template T and each leaf state S, denoted as $score_T(S)$, is defined by the following equation (5). If a template matrix requires two certain cells to be the same color and they are different in a state matrix, the requirement is violated and the score is negative infinity. If no violation is found, the score is decided mainly depending on whether the important cells are filled or not. As the denominator of this equation is the regularization term, the score becomes 1.0 when the template is completely achieved.

$$score_T(S) = \begin{cases} -\infty, & \text{if } \exists i,j \text{ s.t. } s_{ij}t_{ij} < 0 \\ \dfrac{\sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij}t_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} |t_{ij}|} & otherwise \end{cases} \tag{5}$$

D. Complexity

Assuming depth-d search, at most k^d nodes are expanded and compared to all templates, where k is the number of legal actions (usually 22). Calculation of an accordance $score_T(S)$ requires n^2 comparison operations, where n is the number of cells (usually 78). So, if the number of templates is noted by m, the complexity of TMS is $O(k^d n^2 m)$. If TMS is naively implemented, the computational cost is a bit higher than the acceptable level.

Then, in order to reduce the computational cost, some optimizations should be done. The candidates are as follows:

- "for-loop" of $\sum_{j=1}^{n}$ can be skipped at all if the i-th cell is empty.
- If a violation is found, $score_T(S) = -\infty$, then $score_{T_i}(S)$ must also be $-\infty$ for the children $\{T_i\}$ of T.
- Parallelization of calculation is easily implemented at least by using multi-core CPU.

We confirmed that the maximum search time is under 1.0 second, in our environment using C# .Net and Intel Core i5-2310.

V. Experiments on Template Matching Search

In this section, we introduce how to prepare the template matrices. We compare the long chain construction speed of TMS with that of human experts and PMS, and we also show how to combine TMS and PMS.

A. Preparation of the template matrix

TMS requires a set of template matrices. The preparation may seem to be very hard work, because a template is a 78×78 matrix. But by using a semi-automation tool, the standard cost of preparation of one template can be reduced to around 5 to 10 minutes. The procedures is as follows (see Figure 9 also).

1) **Labelling of cells:** A formal chain is fixed (a). A set of cells that must be the same color, is called a group and labelled with a capital letter A,B,... (b). They must be edited by hand. After that, the cells connected to capital letters are automatically labelled with lower case letters a,b,... (b). The other cells are labelled by 0.
2) **Determining the relationship of two groups:** For example in (b), the colors inside group A must be the same, and the colors of group A and group B must be different. Such requirements can be automatically extracted from the labelled board to a matrix (c). But some further manual modifications are often required. For example, the colors of group A and group C must be different (d), considering the situation where group D is eliminated and two blocks fall.
3) **Determining weights:** Advanced players know which group is the key of the formal chain and should be placed preferentially. Then a weight for each group is determined, by hand (e). Normally, the weight of capital letters is around 1000, and the weights of small case letters is fixed to 1.

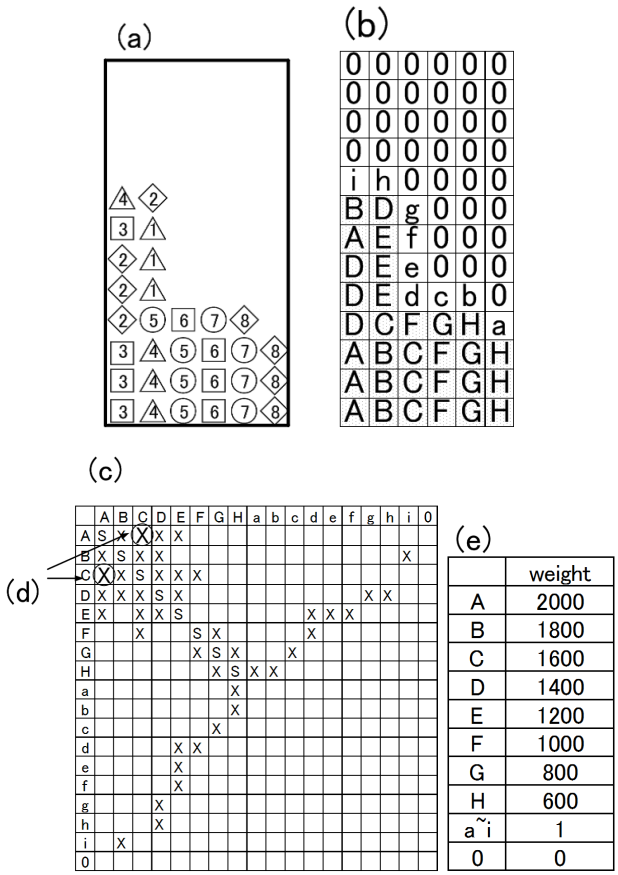

Fig. 9. Intermediate representations of a template. A formal chain (a), labelled cells (b), the relationship matrix (c), required modifications (d) and weights (e).

4) **Conversion to the template matrix:** An element t_{ij} of the template matrix is automatically calculated by the following equation, where i or j is a cell, l_i is its label or group, $rel(l_i, l_j)$ is the required relationship between two labels, and w_l is the weight of label l.

$$t_{ij} = \begin{cases} \min(w_{l_i}, w_{l_j}) & \text{if } rel(l_i, l_j) \text{ is S (same)} \\ -\min(w_{l_i}, w_{l_j}) & \text{if } rel(l_i, l_j) \text{ is X (different)} \\ 0, & otherwise \end{cases} \tag{6}$$

B. Performance Evaluation of TMS

In this subsection we evaluate how well TMS constructs a long chain. 7 templates of an 8-chain including the one of Figure 9 are prepared at first. The least number of blocks needed to execute an 8-chain is 32, or, 16 actions are required at least. However, it is almost impossible to execute an 8-chain with only 16 actions, because of the randomness of the given colors. So, to evaluate the performance of TMS, the numbers of actions should be compared not to 16 but to that of other players.

We employ two other players, who also try constructing and executing an 8-chain. One is the depth-3 PMS, modified

a bit so that if it finds a node with an 8-chain (or longer), it is immediately selected. Another is a human expert, over thousands of hours of training, who can usually construct around 12 to 14-chain, but is asked here to execute an 8-chain (or longer) as fast as possible.

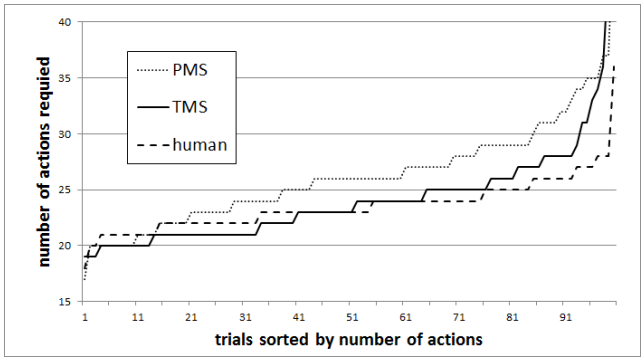

Fig. 10. Number of actions required for executing an 8-chain, sorted from smaller (left) to bigger (right). Comparison of PMS (dot), TMS (solid) and a human expert (broken)

Figure 10 shows the sorted result of 100 games, x-axis is the trials sorted by number of actions, and y-axis is the number of actions required. For example, the number of trials where an 8-chain was achieved under 25 actions was only 36 by PMS but 75 by TMS. It is clear that TMS is superior to PMS at least in the ability of constructing an 8-chain quickly.

TMS is a bit inferior to the human player especially in the lower ranks (right side), but please note that *any* form of 8-chain is permitted in the case of PMS and the human player, whereas only *7 forms* of 8-chains are permitted in the case of TMS.

C. Combination of TMS and PMS

Considering the cost of preparation of the templates and the cost of computation of TMS, it is not realistic to prepare a lot of variations of very long (such as 15) chains. So, we simply propose an idea to combine TMS and PMS: (1) at first TMS is used to prepare a formal 8-chain, (2) after the accordance score is over 0.99, PMS is used to make the chain longer. As the formal chain uses the space efficiently, it is expected that the chain will be extended longer than in the case of PMS alone. This algorithms time per turn is 242.3 msec

Figure 11 (black) shows the histogram of chain lengths executed by this method, that we call TMS+PMS. As for PMS on Figure 5, pseudo depth-4 search with $T_{space} = 16$ is used. The average length and standard deviation of TMS+PMS is 10.96 (2.21), where that of PMS is 9.43 (1.91), so it is clear that the performance is greatly improved.

Figure 12is an example of 15-chain achieved by TMS+PMS. Such length is not so easy to construct even for human experts. Finally, 1000 games were played between TMS+PMS vs PMS. TMS+PMS won 689 games (in other words, about +130 Elo rating). This result also supports the efficiency of combining TMS and PMS.

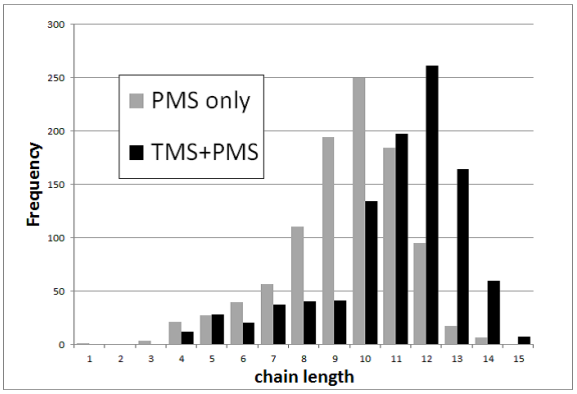

Fig. 11. histogram of chain lengths, in the case of TMS+PMS(black) and PMS(silver), 1000 games.

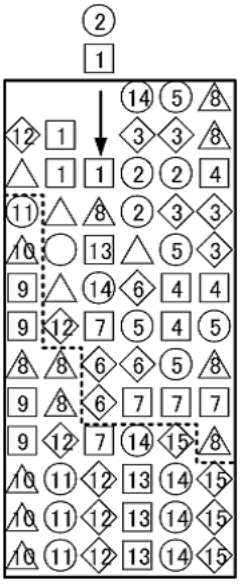

Fig. 12. 15-chain created by TMS+PMS.

VI. Tactical Heuristics

From Section III to Section V, we have mainly discussed how to construct a long chain, because it is the core skill required for PuyoPuyo AIs. However, since PuyoPuyo is a two-player game and the player can affect the opponent's board indirectly, some tactics are necessary in advanced games. In this section, we briefly introduce some tactical heuristics and evaluate their effectiveness.

A. Satisfactory Attack

Very short chains are found in histogram of Figure 11. In such cases, a long (such as 9 or 12) chain is *once* constructed but spoiled by trying to extend more to the threshold of remaining empty space T_{space}. If a comparatively long (such as 12 or 13, depending on the main engine) chain is constructed, it may be better to execute it immediately rather than trying to extend it and fail.

The following condition can be appended to the PMS procedure 3) : "if the number of garbage blocks produced

by the best node is bigger than $T_{satisfy}$". The adequate $T_{satisfy}$ strongly depends on the base engine. For example $T_{satisfy} = 400$, corresponding approximately to a 12-chain, is too high for depth-2 search because it is almost impossible to achieve a 12-chain for it. On the other hand, $T_{satisfy} = 120$, about 7-chain, is too small for pseudo depth-4 search because in many cases more than 7-chain can be achieved by it.

1000 games were done between $T_{satisfy} = 270$ vs original version, where the base is PMS depth-3 search with $T_{space} = 16$, and PMS with this heuristic won 563 games. The effectiveness is confirmed with 99% confidence.

B. Sufficient Counter Attack

Consider the case of Figure 4, and that in addition, the opponent attacked and 80 garbage blocks will fall in the near future. By the first condition of PMS procedure 3), the right-side action is selected and 60 garbage blocks is produced. However, as the opponent's attack is stronger, 20 garbage blocks fall on the player's board, and the opponent will probably win the game.

In such case, one possible option is to skip the immediate counter attack and try constructing stronger chain than the opponent's. With this heuristic, the first condition of PMS procedure 3) is modified (restricted) as follows: "if the opponent already executed a chain, and its amount of garbage blocks is smaller than that of mine". Of course, sometimes this heuristic may lead to a worse result if the chain is not extended successfully.

Here, the reference opponent is set to PMS depth-3 search with $T_{space} = 40$, which tends to attack comparatively earlier. Against this opponent, PMS depth-3 search with $T_{space} = 16$ won 666 games in 1000 games. The reason is that it can make sufficient counter attacks against the opponent's first attack. Further, against the same reference opponent, the PMS with the heuristic proposed here won 713 games in 1000 games. The result implies that the counter skill is improved. The effectiveness is confirmed with 95% confidence.

C. Other Heuristics

There are many other tactics used in advanced games between human players, as follows. In this paper we do not show the implementation of these tactics, but we believe they are important for winning, against human experts.

- If the potential of the opponent board is clearly lower, for example there are few blocks or there are many garbage blocks, it is better to start the chain immediately instead of trying to construct a longer one.
- In the middle stage of the game, 2-chains or 3-chains are often used as a feint. The strength of such attacks is not so big, around 10 to 20, but enough to disturb the construction of long chains.
- There are three possible reactions to the feint attack. To suffer these 10 or 20 garbage blocks is not good in many cases. And, to counter by a long chain such as a 10-chain is sometimes not the best option, because the opponent may re-counter with a longer chain. To counter with short chains such as 2-chains to 4-chains is often the best reaction.

VII. Conclusion

In this paper, we introduced a synchronized version of PuyoPuyo, a popular two-player tile-matching game. For PuyoPuyo AI, construction of "chain" is the core skill, then we introduced two search algorithms, using potential maximization and template matching. The combination of these methods greatly improve the performance of PuyoPuyo AI. 11-chains are achieved on the average. Two tactical heuristics are also introduced and the effectiveness is confirmed. Though there are still some problems to be solved, we believe that these ideas can be useful to many developers, not only of PuyoPuyo series, but also of similar games.

Appendix

Here we summarize the differences between the original PuyoPuyo-2 and the synchronized version used in this paper. These differences affect much the tactical heuristics, but not so much the construction of chains.

- The original version is a real-time game, and the actions of the two players are executed in an asynchronized manner.
- The amount of garbage blocks produced by a chain is a bit different. In the original version, one garbage block can fall only by dropping the pair fast. After a time passed from the start of game, the amount of garbage blocks increase gradually.
- In the original version, the player loses when and only when the cell $(3, 12)$ is filled.
- The blocks at the height 13 are hidden and not connected together (nor disappear) in the original rule.
- The numbers of possible actions after the player notices the opponent's attack are a bit different, caused by the difference between real-time and turn-based games.
- The bonus for the case where all blocks are cleared is ignored in this paper.
- The penalty (delay) for breaking the given pair by putting it sidewise to surfaces with different heights, is ignored in this paper.

References

[1] Boumaza, A., "On the evolution of artificial Tetris players", *IEEE Computational Intelligence and Games*, pp 387-393, (2009)

[2] Buro, M., "Toward Opening Book Learning", *ICGA Journal*, 22:98–102, (1999)

[3] Gelly, G., Wang, W., Munos, R. and Teytaud, O., "Modification of UCT with Patterns in Monte-Carlo Go", Technical Report, No.6062, INRIA (2006)

[4] Ikeda, K., and Kita, H., "State Evaluation Strategy for Exemplar-Based Policy Optimization of Dynamic Decision Problems", *IEEE Congress on Evolutionary Computation*, (2007)

[5] van der Wal, J., "Stochastic dynamic programming", *Mathematical Centre Tracts No. 139*, Mathematisch Centrum, Amsterdam, (1981)

[6] http://en.wikipedia.org/wiki/List_of_best-selling_video_game_franchises

[7] http://en.wikipedia.org/wiki/List_of_Tetris_variants

[8] http://en.wikipedia.org/wiki/Tile-matching_video_game

[9] http://www.geocities.co.jp/lockitjapan/atgc/ (in Japanese)

[10] http://www.vector.co.jp/soft/dl/win95/game/se095051.html

[11] http://alg-d.com/game/puyo/ (in Japanese)