Evolving and discovering Tetris gameplay strategies

Sommuk Phon-Amnuaisuk

*Media Informatics Special Interest Group,
School of Computing & Informatics,
Institut Teknologi Brunei, Brunei Darussalam

Abstract

This work is motivated by one of the important characteristics of an intelligent system: the ability to automatically discover new knowledge. This work employs an evolutionary technique to search for good solutions and then employs a data mining technique to extract knowledge implicitly encoded in the evolved solutions. In this paper, Genetic Algorithm (GA) is employed to evolve a solution for randomly generated tetromino sequences. In contrast to previous works in this area where an evolutionary strategy was employed to evolve weights (i.e., preferences) of predefined evaluation functions which were then used to determine players’ actions, we directly evolve the gameplay actions. Each chromosome represents a plausible gameplay strategy and its fitness is evaluated by simulating the actual gameplay using gameplay instructions from each chromosome. In each simulation, 13 attributes relevant to the gameplay, i.e., contour patterns and actions of each tetromino, are recorded from the best evolved games. This produces 6583 instances which we then apply Apriori algorithm to extract association patterns from them. The result illustrates that sensible gameplay strategies can be successfully extracted from evolved games even though the GA was not informed about these gameplay strategies.

Keywords: Evolving a Tetris player, Genetic algorithms, Discovering Tetris gameplay strategies

1. Introduction

This paper investigates the knowledge discovery of game playing strategies in Tetris. Tetris is a well-known tile-jigsaw puzzle game created by Pajitnov in 1984 [1]. The aim of the Tetris game is to rearrange a falling tetromino on a 20 × 10 tiles board such that there are minimum unfilled tiles on the board. 7 kinds of tetrominoes in Tetris are named according to their similarity to the corresponding alphabets: I, J, L, O, T, S and Z. Although Tetris has a simple gameplay, plausible combinations of tetrominoes’ positions and rotations produce a huge number of game states. With the board size of 20 × 10 tiles, the upper bound of possible game states is in the order of 7 × 2^{200} states. It is suggested by [2] that finding a Tetris controller is an NP complete problem.

The Tetris game has become a popular board game explored by many AI game researchers since it has simple game rules but a complex gameplay strategy. Emulating players’ behaviours using handcrafted rules and heuristics is one of the established research themes investigated by many colleagues [3].

Rules for players’ actions are commonly formulated based on the fact that tetrominoes should be placed in such a way that all the empty spaces in each row should be fully filled, or the placement does not create unreachable holes, or other desired/undesired properties. This concept is common in all previous works. For example, the following properties: the top-most contour formed by the filled tetrominoes, the number of unreachable holes, the maximum and the minimum height of the filled tetrominoes, etc., have been employed...
to represent the board states. The association between the board states and actions can then be determined using the generate-and-test tactic. All plausible next states are attempted and the best choice is taken [4]. The performance of the model depends on how far ahead the program looks into the future.

Although the traditional knowledge engineering approach is effective and able to compactly encode expert knowledge, the knowledge acquisition bottle-neck in the traditional knowledge engineering approach poses a big challenge to many problem domains. The process is time-consuming and effective heuristics are not always achievable. Despite those limitations of the traditional knowledge engineering approach, very few game AI researchers have explored the possibility of extracting playing strategies using knowledge discovery via soft computing techniques.

Here, we investigate the application of evolutionary computing techniques to discover the control knowledge of how to play Tetris. We are interested to find out patterns that emerge from gameplay that evolved using only a general fitness function. In the Tetris game implemented here, the fitness function reflects the information about the number of unfilled tiles. The intuition behind this is to have a simple evolutionary system that evolves solutions and then extracts higher level knowledge from the solutions.

The rest of the materials in the paper are organized into the following sections; Section 2: Related works; Section 3: Problem formulation; Section 4: Experimental results & Discussion; and Section 5: Conclusion.

2. Related works

Handcrafted rule-based Tetris controllers have been investigated and have matured for decades [3]. In a rule-based approach, player’s actions are determined based on the information extracted from the game. Properties such as the top-most contour formed by the filled tetrominoes, the number of unreachable holes, etc., have been employed to represent the board status (readers can find a good summary in [5]). These desired board properties can be quantified as a value function \( V(s) \) [6] which is commonly expressed as a weighted linear sum of the fitness of the desired board properties \( f_i \), e.g., \( V(s) = \sum_{i=1}^{N} w_i f_i(s) \).

Applying these rules and heuristics to the game could generate different controllers’ behaviours as a result of various weighted combinations of those rules and heuristics\(^1\). As the number of rules and heuristics increases, it is not a trivial matter to tune these parameters manually. Hence, evolutionary computing has been popularly employed to search for an optimal weighted combination of these rules [6, 7]. Recent advances in soft computing approach has also been explored by many researchers, the Reinforcement Learning (RL) technique learns the state-actions policy by playing many games and learns the associations between the actions and the accumulated board values \( V(s) \) of the current action and the sequence of future actions [8, 9]. Relational reinforcement learning (RRL), cross entropy RL, and cross entropy RRL have been recently investigated by [10].

Tetris has also been the domain for cognitive scientists who want to study how we learn problem-solving skills from the cognitive science perspective [11]. However, attempt to discover Tetris gameplay strategies through knowledge discovery technique has not captured the intention of most researchers yet. To our best knowledge, this work represents one of the early works in this area.

The idea of a self learning system capable of discovering important concepts by itself has been proposed and discussed for decades. A self-learning system capable of learning and discovering new concepts must perform the following essential tasks: (i) pattern generation, (ii) pattern discovery, and (iii) concept formation. Although the framework has been laid out and discussed in many places, a full-scale self-learning system capable of forming new concepts in this fashion has not been implemented. In this paper, we discuss pattern generation and pattern discovery components from the perspective of a Tetris domain. Genetic Algorithm (GA) is employed to evolve Tetris gameplay of randomly generated tetromino sequences. The evolved gameplay patterns are then analyzed using association rule mining to find gameplay patterns i.e., playing strategy\(^2\).

\(^1\)see http://www.colinfahay.com/tetris/tetris.html

\(^2\)Although association rules mining can be seen as concept formation, we box all data mining techniques under pattern discovery. In our view, concept formation should involve abilities to automatically generate new knowledge units, new relations between these units, and hierarchically generating new concepts from existing concepts.
3. Our approach

GA is employed to evolve a player’s gameplay to correspond to the sequences of tetrominoes. For each given sequence, GA found a sequence of actions that would optimally place the tetrominoes such that the actions created minimum unfilled holes. We hoped to see a playing strategy emerged from the gameplays recorded from many different games. The playing strategies were then extracted using the association rule mining technique. The following assumptions were made to the Tetris gameplay in this implementation:

1. For each time step, information of a single tetromino was revealed to a player. The information of the next tetromino would only be revealed after the player had played the current tetromino.
2. For each tetromino, only one action (a combination of rotation and translation) was allowed.
3. No row elimination was carried out during the game play. Hence, a sequence of 50 tetrominoes were generated for each game. The sequence is randomly generated by uniformly chosen from the set \{I, J, L, O, T, S, Z\}.

Let us formally define a Tetris game using a tuple \((S, A, F, V)\), where \(S\) is a set of game states; \(A\) is a set of a player’s actions; \(F\) is a transition function; and \(V\) is a value function [12].

- **Game States** \(S\): A state \(s \in S\) is a plausible arrangement of tetrominoes on the Tetris board area. In this implementation, the state \(s \in \{0, 1\}^{20 \times 10}\) is represented as a binary matrix, where the entry 1 denotes a filled tile and the entry 0 denotes an empty tile.

- **Player’s Actions** \(a \in A\): During the gameplay, each new tetromino \(\tau \in \{I, J, L, O, T, S, Z\}\) is placed on the board at the topmost row. At each time step, the tetromino piece falls down one tile due to its gravity. A player may rotate the tetromino (i.e., 90, 180, 270 degree) and translate the tetromino in the horizontal axis (i.e., move left/right). Here, \(A\) denotes a sequence of actions \(\{a_1, a_2, ..., a_n\}\) a player has taken from the start of the game till the end of that game. An action \(a\) is expressed as a tuple \((\tau, r, x)\) where \(\tau\) indicates the desired tetromino, \(r \in \{1, 2, 3, 4\}\) where \((r - 1)90\) indicates the desired rotation and \(x \in \{1, 2, ..., 10\}\) indicates the left most position of the tetromino.

- **Transitional function** \(F\): The transitional function \(F(s, a) : s_t \to s_{t+1}\) maps the current board state \(s_t\) to the new board state \(s_{t+1}\) as a result of applying a **valid action** \(a\). A valid action is an action that conforms to the rules of Tetris.

- **Value Function** \(V\): The value function \(V(s) : s \to \mathbb{R}\) maps the board state \(s\) to a real value. The value of a state is defined as a weighted sum of various predefined fitness criteria \(f_i\).

\[
V(s) = \sum_{i=1}^{n} w_i f_i
\]  

In [6], various features (i.e., fitness criteria) have been proposed. Our fitness criteria is a simple unfilled tiles measures which will be discussed in the next section.

3.1. Representing and evolving a gameplay

A perfect game in our implementation only happened when all 50 tetrominoes fill the 200 tiles exactly. Therefore, the objective of the game was formulated as to maximise the filled tiles (or minimise the unfilled tiles). Each game was represented as a chromosome \(c_n\) which represented a sequence of actions \(\{a_1, a_2, ..., a_m\}\).

A population of chromosomes \(C\) was a \(m \times n\) matrix where \(m\) denoted the length of the sequence of actions and \(n\) denoted the number of chromosome in the population. Each chromosome \(c_n\) represented a different playing strategy of a given sequence of tetrominoes. These chromosomes were evolved. For each generation, the fitness of each chromosome was evaluated by simulating the actual Tetris gameplay. The chromosomes with lesser unfilled tiles were considered better and were desired to be reproduced in the next generation.
The total amount of unfilled tiles at the end of each game was a good indicator of the overall game quality. A better game should have less unfilled tiles. However, the number of unfilled tiles was not a very good measure to evaluate a player’s action during the gameplay since it did not have enough expressive power to describe the consequence of different actions. It was, however, decided to evaluate chromosomes’ fitness using a measure that reflects the number of unfilled tiles. This was motivated by our curiosity to see whether complex gameplay strategies could emerge from games evolved with just a simple fitness scheme that did not describe any deep knowledge about the game.

3.1.1. Fitness functions
We defined partial unfilled tiles as those unfilled tiles in a desired area. We counted all the partial unfilled tiles in a desired area which grew as generation progressed [13] (see Eq 2). In this way, we placed more emphasis on the bottom area of the board in the early stage of the game and gradually enlarged the area to full board toward the end of the game. Let us described the process using these parameters: \( h \), step and \( w \). The \( h \) denoted the number of rows on the Tetris board, step denoted the position along the row of the Tetris board as the GA generation, \( n \), increased toward the maximum generation (here set at 500):

\[
step = h(maxGeneration - n)/maxGeneration
\]

The weight \( w \) denoted the importance of row \( i \) and was defined as

\[
w = e^\alpha \text{ where } \alpha = \begin{cases} 0 \text{ if } i > \text{step}, & \text{else} \\ (10 \times i)/\text{step} - 10 & \end{cases}
\]

\[
f = \sum_{i} \left(10 - w \sum_{j} s(i,j)\right) \tag{2}
\]

3.1.2. Reproduction process
In this implementation, the fittest 10 % of the whole population was selected and continued to the next generation without going through the crossover and mutation process. The rest of the population would go through a standard one point crossover and one point mutation. Two parents would be selected in pair: \( c_x \) according to its rank in the whole population and \( c_y \) randomly selected from the rest of the population. Then, a standard one point crossover was applied to \( c_x \) and \( c_y \). This produced two offsprings and the better offspring \( c_z \) was retained for the next generation. Then a one point mutation was randomly applied to the chromosome \( c_z \). The process continued until all the termination criteria were met or the GA reached the maximum generation.

3.2. Simulation
Each chromosome was evaluated by simulating an actual game, see Figure 1. The SIMULATE function took a sequence of tetrominoes and the evolved actions (i.e., position and rotation) as its input. The game was simulated and the fitness was computed.

```plaintext
function SIMULATE(SEQ, ACTIONS) return v
Input : SEQ = \{\tau_1, ..., \tau_{50}\} \text{ where } \tau \in \{I, J, L, O, T, S, Z\}
       ACTIONS = \{a_1, ..., a_{50}\} \text{ where } a_n = (r, x)_n
Output : v \in R
Var : s \in S; r \in \{1, 2, ..., 4\}; \text{ and } x \in \{1, 2, ..., 10\}
Initialize a board s \leftarrow \text{ a zero matrix } 0^{20 \times 10}
Initialize v \leftarrow 0
for each \( \tau_n, a_n \) {
    s \leftarrow F(s, \tau_n, a_n) \text{ where } a_n = (r, x)_n
    v \rightarrow v + V(s)
}
return v
```
3.3. Mining playing strategies from evolved games

3.3.1. Recording patterns

It was decided that the following attributes be recorded: (i) top tile contour pattern, (ii) tetromino \( \tau \), and (iii) action \( a \) taken for each tetromino. There was a total of 145 evolved gameplays and therefore 7,250 episodes for all actions \((50 \times 145)\). However, only a perfect game used all 50 tetrominoes. From our experiment, we recorded a total of 6,585 episodes, 13 attributes describing contour patterns, tetromino type and actions (i.e., position and rotation) were recorded in each episode. Figure 2 illustrates some contour patterns examples. We hypothesized that contour patterns were dependent on the logical decision of how to place the tiles. Therefore, we expected to see some association patterns emerged from the recorded data.

3.3.2. Association rules mining

The \textit{Apriori algorithm} is a popular association analysis algorithm [14]. The idea behind is to list out possible combinations of attributes in the dataset; then, interesting relationships in the form of implication
of those attributes can be evaluated. But this would result in a large number of possible combinations. Two constraints: support \( s \) and confidence \( c \) are commonly employed to prune out combinations that are less significant.

\[
s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \quad (3)
\]

\[
c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \quad (4)
\]

where \( X \) and \( Y \) denote disjoint sets \( (X \cap Y = \emptyset) \) of items in the domain of interest i.e., contour patterns, position and rotation of a tetrominos; and \( \sigma(\cdot) \) is a function that return the count of items in the set. A support \( s(X \rightarrow Y) \) is the ratio the number of occurrence of the set \( X \cup Y \) over the total observation count (i.e., \( N \) transactions). A confidence \( c(X \rightarrow Y) \) is the ratio the number of occurrence of the set \( X \cup Y \) over the total occurrence of the set \( X \), note that the set of attributes \( X \cup Y \) is always a subset of the set of attributes \( x \) in the Apriori algorithm context. Figure 1 graphically summarises the evolutionary process, simulation, pattern recording and association rule mining process implemented here.

4. Experimental design and results

Each GA population evolved a gameplay for a randomly generated tetromino sequence. 145 tetromino sequences, each with 50 tetriminoes uniformly sampled from the set \{I, J, L, O, T, S, Z\}, we employed in this study\(^3\). Hence 145 GA population were created, each population evolved within a maximum of 500 generations used the following parameters: 200 chromosomes, 10% elitism scheme, one point crossover, and one point mutation. The pseudocode below highlights the evolutionary process employed in this implementation.

\[
\text{function } \text{GA-Tetris}(C, \text{FITNESS-FN}) \text{ returns } c \\
\text{// } C : \text{ is an } m \times n \text{ matrix represent a set of individual } \{c_1, ..., c_n\} \\
\text{// Each } c_n = [a_1, ..., a_m] \text{ where } a = (r, x); r \in \{12, ..., 4\}; x \in \{1, 2, ..., 10\}; \\
\text{repeat} \\
\quad \text{newC} \leftarrow \phi \\
\quad \text{evaluate } C \text{ according to \text{FITNESS-FN}} \\
\quad \text{add top 20\% individuals to newC} \\
\quad \text{while } \text{size(newC)} < \text{size(C)} \text{ do} \\
\quad \quad c_x \leftarrow \text{SELECTION}(C), \text{ according to its fitness ranking} \\
\quad \quad c_y \leftarrow \text{RANDOM-SELECTION}(C) \\
\quad \quad c_z \leftarrow \text{CROSSOVER}(c_x, c_y) \\
\quad \quad c_z \leftarrow \text{MUTATE}(c_z) \\
\quad \quad \text{if } c_z \text{ is well-formed } // \text{ actions can be carried out on a Tetris board} \\
\quad \quad \text{then } \text{newC.APPEND}(c_z) \\
\quad \quad \text{else } c_z \leftarrow \text{REPAIR}(c_z) \text{ and newC.APPEND}(c_z) \\
\quad C \leftarrow \text{newC} \\
\text{until } \text{termination criteria are met} \\
\text{return the best individual } c \in C, \text{ according to \text{FITNESS-FN}}
\]

4.1. Results and discussion

The fitness profiles of all 145 sequences shared a similar pattern. Figure 3 shows the average values of unfilled and partial unfilled tiles observed in the GA populations over the whole evolution process. The numbers of unfilled tiles starts at around 80-100 in the first generation and at the game over, there was an average of 20 unfilled tiles. The numbers of partial unfilled tiles starts at around 10 in the first generation and converges to an average of 20 unfilled tiles at the game over.

It is interesting to compare the gameplays of both evolution strategies to a human’s. Examples of gameplays by the system and a human playing on the same tetromino sequence is shown in Table 1). as

\[^3\text{This constitutes a total number of 7,250 tetrominoes episodes.}\]
well as in Figure 3. A human player played the same sequence and it was decided that the human player could take as much time as he/she needed to play each tetromino since we were only interested in comparing gameplay strategy of a human and a computer. The result shows that the quality of GA solutions are comparable to a human’s gameplay.

The fact that a human player cannot perform better than the evolved gameplay from GA implies that the evolved solutions are of reasonable quality. In this experiment, we decided to record (i) the board environment as a contour pattern formed by top-most tiles and (ii) the action corresponding to the environment.

---

4Sequences are randomly taken from the pool of 145 sequences employed in this study.
Table 1. a) Examples of tetromino sequences employed in this experiment; b) Evolved gameplay and a human gameplay for an instance of a sequence.

| a Sequence | ILISLTJLZZTTLLJISLZZHTZLZSOJJJSLSIIZTZJITTOSJSZLOL JLSZLIZOJILTJLZZTSLJISLZITJLTTJTISZSOZTHLSTIJSZ SIZOTSTEZSTTTOILJILJISLZIZTZEJITLZITLZITLZOTLZI JIOISLZSOJJIIIOOIJITLJLZISLZILOOTJITJIOOZIOLOISSL IOILLZTSLJITTOLOJLTOZSOOSLJILJLOZOTTTISZSJTLJLO SSZTIZTJTOJLZLJTSJTTSSIZZZJSST0OSSZIOJSOIOLOJLIJ SILZISTZOLJLJLOZZLSITTTI0ZLIOJZZZSSSSTOLSSTITTIL JISTTTJSTOSLIZOTILLLJSLITTLLOJTITOSJJOSIJJSTSSS S0ISLJISLISTISJILJISLZSLZOTJSTIJLLJSLZOTJLLO0S0OL ISTSSOJZZOSJ0ITJ0JZIJSZOTJZZ0SSOLJLLITTLJJLJLT S0JITOISSLZLIJSJIZZSSL0STTJLTTJL05OHTZOSOT IITLSTJLJSTLO0O0TOI0LTOOTJS0OSSZTTITZTTITHIISZTJS |
| b Sequence | OTLTILJZTJTZOLSTOTILOTOISILJLTOISTLZSOOIJZII0 GA | x translation | 68919513886136423643361A96182741986513413696 |
| Rotation | 134342442422222222222441242244424412212242443 | |
| x translation | 13643194779447722237647969971335113A55178637 |
| Rotation | 13422432422222222222242244424244242222222222 |

given a type of tetromino. There were a total of 6583 instances recorded from 145 games, each game had a different tetromino sequence.

In our experiment, each instance had 13 recorded attributes i.e., a contour pattern (10), position and rotation (2), a tetromino (1). However, we chose not to perform association rule mining on these attributes since (i) 6583 instances were too small a data size considering all plausible combinations from 13 attributes, and (ii) it might not be fruitful to have the basic knowledge unit at the contour patterns. We decided to combine the contour information and the position together. This preprocessing reduced 13 attributes to only 4 attributes:

1. s1: the difference between the contour of the position \( x_L \) that the token has landed \( x_{L+1} - x_L \) and IF L == 10 THEN s1 = 0
2. s2: the difference between \( x_{L+2} - x_{L+1} \) and IF L \( \geq \) 9 THEN s2 = 0
3. rotation: the rotation of the token i.e., 1 = 0 degree, 2 = 90 degree
4. token: tetromino \( \tau \in \{I, J, L, O, T, S, Z\} \)

With 85% confidence level, the Apriori algorithm revealed 44 patterns. Examples of association patterns are listed below:

Figure 4. Different rotations, 1, 2, 3 and 4 denote 0, 90, 180 and 270 degree respectively in clockwise rotation.
Although association rules computed by the Apriori algorithm explicitly show an implication, it should be pointed out that the interpretation of a rule is not a direct translation in most cases. For example, the first pattern above (\(s_1 = 0 \text{ rotation}=3 \text{ token}=I \implies s_2=0\)) should be interpreted as:

\[
\text{IF token == I AND x}_2-x_1 == 0 \text{ AND } x_3-x_2 == 0 \text{ (the contour has a flat pattern, i.e., } s_1 = 0 \text{ and } s_2 = 0) \text{ THEN position I horizontally at the position } x_1.
\]

The interpretations of these 44 patterns as rules for each tetromino type are summarized below:

- **I**
  - IF token == I AND x\(_2\)-x\(_1\) == 3 or 4 or 5
    THEN position I at x\(_1\) with rotation = 2 or 4.
  - IF token == I AND x\(_2\)-x\(_1\) == 0 and x\(_3\)-x\(_2\) == 0
    THEN position I at x\(_1\) with rotation = 1 or 3.
- **J**
  - IF token == J AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 2
    THEN position J at x\(_1\) with rotation = 2.
  - IF token == J AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 0
    THEN position J at x\(_1\) with rotation = 3.
  - IF token == J AND x\(_2\)-x\(_1\) == 2 AND x\(_3\)-x\(_2\) == -1 or 0 or 1
    THEN position J at x\(_1\) with rotation = 4.
- **L**
  - IF token == L AND x\(_2\)-x\(_1\) == -2
    THEN position L at x\(_1\) with rotation = 2.
  - IF token == L AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 0
    THEN position L at x\(_1\) with rotation = 3.
  - IF token == 0 AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 0 or 1 or 2 or 3
    THEN position 0 at x\(_1\) with rotation = 1 or 2 or 3 or 4.
- **O**
  - IF token == S AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 0 or 1
    THEN position S at x\(_1\) with rotation = 1 or 3.
  - IF token == S AND x\(_2\)-x\(_1\) == -1 AND x\(_3\)-x\(_2\) == 2 or 4
    THEN position S at x\(_1\) with rotation = 2 or 4.
  - IF token == T AND x\(_2\)-x\(_1\) == -1 AND x\(_3\)-x\(_2\) == 2
    THEN position T at x\(_1\) with rotation = 2.
  - IF token == T AND x\(_2\)-x\(_1\) == 0 AND x\(_3\)-x\(_2\) == 0
    THEN position T at x\(_1\) with rotation = 3.
  - IF token == T AND x\(_2\)-x\(_1\) == 1 AND x\(_3\)-x\(_2\) == 1 or 2
    THEN position T at x\(_1\) with rotation = 4.
  - IF token == Z AND x\(_2\)-x\(_1\) == -1,0 AND x\(_3\)-x\(_2\) == 0
    THEN position S at x\(_1\) with rotation = 1 or 3.
Figure 4 provides the readers with a reference on rotation information. Upon examining these rules, one realizes that all these extracted rules are logical description of how to place a tetromino on the existing board such that the piece fits the contour without creating a hole. This shows that the evolved gameplay has successfully captured these rules. Would it be possible to extract a more complex rule that describes a more complex gameplay strategy? We believe the current paradigm can be exploited further. In future work, more handcrafted features and their relations that can expressively describe a gameplay will be computed. Interesting gameplay patterns should emerge provided that enough transactions are available.

5. Conclusion & future work

In this paper, we investigated the knowledge discovery approach that automatically discovered Tetris gameplay strategies. The approach employed GA to evolve gameplays of 145 different games. The best solution found in each evolved game was employed to simulate a Tetris game and their game parameters were recorded. This process generated 6583 tetromino episodes. The board state and actions during the simulation were recorded using 13 attributes: a contour pattern (10), position and rotation (2), a tetromino (1). The original 13 attributes were preprocessed and re-represented using 4 attributes (i.e., s1, s2, position and rotation). We performed association rules mining on these data and successfully discovered rules that could correctly rotate and place a given tetromino to the board. Although it is quite simple to manually handcraft rules that could correctly rotate and place a given tetromino in a suitable position, it should be pointed out that our system was not fed with this knowledge explicitly. The GA evolved with only the knowledge of unfilled tiles.

Data mining techniques only finds patterns of existing attributes (i.e., concepts). However, new concepts could be hierarchically constructed from old concepts, or derived from old concepts through extra external knowledge. We hope to pursue the study of concept formation in our future work.

Acknowledgement

We wish to thank anonymous reviewers for their comments, which help improve this paper. We would also like to thank GSR, Institut Teknologi Brunei for their partial financial support given to this research.

References