

Automatic Modeling of Frequent Behaviors of Avatars and Players in a On Line Game



Chih Ming Chiu¹, Shao-Shin Hung² and Jyh-Jong Tsay¹

¹Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan

²Department of Computer Science and Information Engineering, WuFeng University, Chiayi, Taiwan

Abstract—Millions of people now participate in on line games, placing tremendous and often unpredictable maintenance burdens on their operators. Thus, understanding the dynamic behaviors of a player is critical for the systems, network, and designers. To the best of our knowledge, little work builds character interaction model based on the data stream mining. This work improves our understanding the behaviors of avatar/player in a on line game by collecting the behavior data, extracting frequent behavior patterns, learning the hidden hints and making good prediction on responses to the unexpected impacts. Besides, we develop two efficient approaches for mining the behavior data to find the interesting behavior pattern for future prediction on responses of opponents. Our novel findings include the following: One, due to the constraints of limited resources of time, memory, and sample size, MSS-MB was proposed to meet these conditions; the other, due to the constraints of real-time and on-line, there may have some errors occurred in the processing period, MSS-BE was proposed to control the errors as needed. Finally, based on the experimental results, we can predict the responses of opponents efficiently in the on line game.

Index Terms— Frequent behavior, on-line game, avatar, stream

I. INTRODUCTION

Character interaction model has become prevalent in the computer graphics industry, the general adaptation and re-use of the data is limited by the available techniques for editing and modifying data in controllable ways. In many games and movies, when impacts are detected, motion-capture driven characters are turned over to a dynamics engine to accept the impulse [1, 14]. The problem with this method is that their models and approaches are based on vector models [1, 2, 3, 14] or final state machines [2, 3, 14].

To gain a better understanding of the patterns of player interaction (or NPC: non-Player Character) and their implications for game design, we analyze “how and what players interact”. Analyzing user behavior based on database-level traces is particularly useful for our purpose, since the inferred user behavioral patterns would naturally connect to interactions factors, such as styles of attacking (and

defense) between participating parties. Based on an empirical analysis of player interaction inferred from network traces, this research work aims to put forward architectural design recommendations for online games.

Also the nature of adaptive behavior needs to be decided. As Mozer [5] states in his criticism of smart houses, it is extremely hard to infer the state of mind of the user from observing her behavior, so a more subtle approach needs to be considered, similar to that of recommender systems [6], where the information provided is filtered, according to patterns of similar behavior of other users. Therefore, our User Model (UM) constitutes a basic component of every adaptive game system. The UM represents user characteristics that enable the system distinguish among different players. The UM is built using data that are requested either directly by the players or obtained by logging player interaction. There are different types of player behavior that can be used for building the UM, for example:

- Player’s characteristics (such as age, gender, location, etc.);
- Player’s preferences and interests;
- Player’s knowledge and skills;
- Player’s behavioral patterns.

On the other side, a data stream is an ordered sequence of items that arrives in timely order [6, 7, 8, 3]. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [6, 14]. Under careful observations, our high speed game play data (e.g. continuous sequential player’s styles) coincide to have the same characteristics. Besides, there are some inherent challenges for data stream association rules mining [3, 12]. First, owing to the unlimited amount of stream data and limited system resources, such as memory space, a mining mechanism that adapts itself to available resources is needed. Second, due to the characteristics of data streams, there is a huge amount of data in both offline and online data streams, and thus, there is not enough time to rescan the whole database or perform a multi-scan as in traditional data mining algorithms whenever an update occurs.

Because of these constraints, traditional data mining methods developed for static databases cannot be applied for mining data streams. Here, in our action MMORPG genre, the style data is generated as a stream, yet we would like to find sequential patterns from the players for analysis. However, there has little work on finding effective methods for prediction of actions of opponents in MMORPG. In this paper, we adopt

the algorithms from L. Mendes .et. al [8] and make necessary modifications for our work. We propose two effective approaches for mining sequential patterns from continuous player's style data: MSS-BE (Motion-based Stream Sequence miner using Bounded Error) and MSS-MB (Motion-based Stream Sequence miner using Memory Bounds).

The remainder of this paper is organized as follows. Related works are discussed in Section 2. Section 3, describes the proposed problem definition. Section 4 presents our system framework and 5 explain the recommended data streaming mining algorithms with illustrative examples. Section 6 then presents the experiment results. Conclusions are finally, drawn in Section 7, along with recommendations for future research.

II. RELATED WORKS

2.1 Character Interaction Models

Many virtual humans are used in computer games like RPGs, sports games, and action games. In these games, the reactions of virtual humans are created by connecting motions in a motion database such as Motion Graph [1, 2, 3, 9, 14, 15, 16]. Therefore, variations among the reactions are limited, and there is a large cost to creating a motion database. Using a database approach, Lee and Lee [10] created realistic motions for boxers; but their method did not create dynamic reaction for contacts. Advances in human interface and virtual reality technologies make possible a wide variety of user inputs. Hence virtual humans are required to perform a wide variety of reactions. Dynamic control and simulation methods use controllers to compute joint torques based on current states and desired actions. These methods create dynamically correct motions from specified motions [1, 14], and state machines [2, 14]. Those methods generate appropriate motions for variations of the input. State machines are used to create the motions and actions of virtual characters that correspond to environments Lattner. et. al. [3] present an approach which applies unsupervised symbolic learning off-line to a qualitative abstraction in order to predict future situation using frequent patterns. However, these researches did not support on-line and real-time direct character interactions.

2.2 Data Streaming Mining

Several novel algorithms have been proposed in the literature to tackle this problem. There are generally two approaches: counter-based methods, and sketch-based methods. Counter-based algorithms maintain counters for a fixed number of elements of the stream, and only this limited number of elements is monitored. If an item arrives in the stream that is monitored, the associated counter is incremented, else the algorithm decides whether to discard the item or reassign an existing counter to this item. The prominent counter-based algorithms include Sticky Sampling and Lossy Counting (LC) [8], Frequent (Freq) [6, 8], and Space-Saving (SS) [8].

The other approach is to maintain a sketch of the data stream, using techniques such as hashing, to map items to a reduced set of counters. Sketch-based techniques maintain approximate frequency counts of all elements in the stream, and they can also support deletions. As such, these algorithms are much more flexible than the counter-based methods. The prominent sketch-based algorithms include CountSketch1 (CCFC) [6, 7], GroupTest (CGT) [6, 7], Count-Min Sketch (CM) [8], and hCount (hC) [8].

III. PROBLEM DEFINITIONS







Let $I = \{i_1, i_2, \dots, i_j\}$ denote a set of literals, called items. A sequence s contains an ordered set of items X from I denoted by $\langle s_1, s_2, \dots, s_j \rangle$. A sequence $s = \langle a_1, a_2, \dots, a_p \rangle$ is a subsequence $s' = \langle b_1, b_2, \dots, b_n \rangle$ if there exist integer $i_1 < i_2 < \dots < i_p$ such as $a_1 = b_{i_1}, a_2 = b_{i_2}, \dots, a_p = b_{i_p}$.

A data stream of sequences is an arbitrarily large list of sequences. A sequence s contains another sequence s' if s' is a subsequence of s . The count of a sequence s , denoted by $\text{count}(s)$, is defined as the number of sequences that contains. The support of a sequence s , denoted by $\text{supp}(s)$, is defined as $\text{count}(s)$ divided by the total number of sequences contained. If $\text{supp}(s) \geq \sigma$, where σ is a user-supplied minimum support threshold, then we say that s is a frequent sequence, or a sequential pattern.

For example, suppose the length of our data stream is only 3 sequences: $S1 = \langle a, b, c, d \rangle$, $S2 = \langle a, c, d \rangle$, and $S3 = \langle b, c, d \rangle$. Let us assume we are given that $\sigma = 0.5$. The set of sequential patterns and their corresponding counts are as follows: $\langle a \rangle:2$, $\langle b \rangle:2$, $\langle c \rangle:3$, $\langle d \rangle:3$, $\langle a, c \rangle:2$, $\langle a, d \rangle:2$, $\langle b, c \rangle:2$, $\langle b, d \rangle:2$, and $\langle b, c, d \rangle:2$.

■ *Style Definition:* In this work, we define six different styles for distinguishing responses from opponents and NPC. The styles are shown in Table 1.

TABLE 1: DIFFERENT STYLES IN OUR CHARACTER INTERACTION MODEL.

| Id | Style | Name | Id | Style | Name |
|----|--|---------------|----|---|----------------|
| a |  | Down-Attack | d |  | Down-Defense |
| b |  | Middle-Attack | e |  | Middle-Defense |
| c |  | Up-Attack | f |  | Up-Defense |

IV. SYSTEM FRAMEWORK

Our system is divided into four phases including input data stream sequence, data batching mechanism, data stream mining and style prediction mechanism, and NPC animation control. In the beginning, player's on-line data are logged and broken into fixed-sized batches. Next, the data stream mining algorithms are utilized to build the lexicographic tree T_0 . Once the tree is built, we use the on-line play's style input to query the tree and make predictions. Finally, the returned next possible player's style and its probability of appearance are used to play the NPC's animation. Please note that, before the tree is built, we have no idea about what player's next style is and just play the default corresponding attacked animations. The framework is illustrated in Fig. 1. In the following section, the detail algorithms are described.

V. STREAM SEQUENTIAL PATTERN MINING ALGORITHMS

Inspired by concepts of [8, 12], we make necessary modifications and propose two novel streaming mining algorithms, namely SS-BE and SS-MB. These algorithms can break up the stream into fixed-sized batches and perform sequential pattern mining on each batch. They use a lexicographic tree T_0 to store the subsequences seen in the data stream.

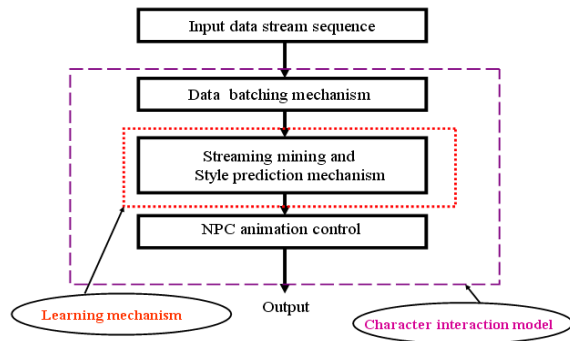


Fig. 1. Our system framework.

5.1 The Motion-based SS-BE (MSS-BE) Algorithm

Now, we will explain our modified mining algorithms. For the modified method, each node below the root in the tree T_0 will have four attributes:

Style: denotes a player's style represented by this node;

Count: denotes the count of the sequence corresponding to the path from the root to this node;

PID: denotes the last pruning batch number before this node was inserted in the tree;

batchCount: indicates how many batches have contributed to the count of this node since the last time it was introduced to the tree

The MSS-BE algorithm will given below and demonstration example will.

Motion-based SS-BE (MSS-BE) Algorithm

// Tree T_0 is set initially to have the root node with its count that denotes the number of sequences occurred set to 0.

Input:

D : an incoming stream of sequences S_1, S_2, \dots ;

σ : the minimum support threshold;

ε : the significance threshold where $0 \leq \varepsilon < \sigma$;

L : the length of batch;

α : the batch support threshold where $0 \leq \alpha \leq \varepsilon$;

δ : the pruning period.

Output: tree T_0

Begin

1. Break the data stream into fixed-sized batches, where each batch contains L sequences;
 2. **While** (each batch B_k that arrives) **do**
 3. **Begin**
 4. Apply *PrefixSpan* [1] to it, using support α ;
 5. **For** each frequent sequence s_i that is found, having count c_i **do**
 6. **If** a path corresponding to this sequence does not exist in the tree **Then**
 7. Create one new path, setting the *batchCount* and *count* values of the new nodes to 0 and the PID values to the last pruning batch number;
 7. **Else**
 8. Increment nodes' count by c_i and their *batchCount* by 1;
 9. **EndIf**
 10. **EndFor** // For
 11. **If** the number of batches seen is a multiple of the pruning period, δ **Then**
 12. **For** each node in the tree **do**
 13. Let B equals total number of batches elapsed minus the node's PID value;
 14. Let $B' = B - \text{batchCount}$;
 15. **If** $\text{count} + B'(\lceil \alpha L \rceil - 1) \leq \varepsilon B L$ **Then**
 16. delete the entire sub-tree rooted at that node from the tree;
 17. **EndIf**
 18. **EndFor**
 19. Update the last pruning batch number;
 20. **EndIf**
 21. return T_0 ;
 22. **EndWhile**
- End** // procedure MSS-BE ends

Our modifications are in underline and using the node's PID value to compute the B 's value instead of using the timestamp approach. This improves the efficiency of tree pruning.

Whenever the set of sequential patterns are needed, we can output all sequences corresponding to nodes in the tree having $\text{count} \geq (\sigma - \varepsilon) * N$, where N is the number of sequences occurred.

Example 1 (MSS-BE). Suppose the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ε is 0.5, and the batch support threshold α is 0.4, and the pruning period δ is 2. We assume that player's styles arrive in the following data stream shown in Fig. 2.

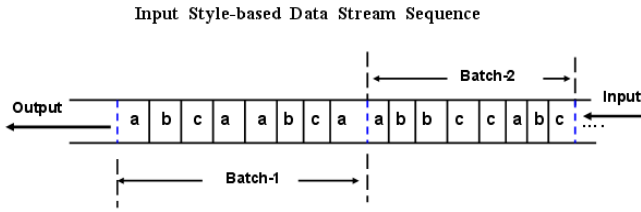


Fig. 2. Overview of incoming stream sequence for MSS-BE.

S1. Break the data stream into fixed-sized batches, where each batch contains L sequences. Let batch B_1 be consisted of sequences $\langle a,b \rangle, \langle c,a \rangle, \langle a,b \rangle$ and $\langle c,a \rangle$. Let batch B_2 be consisted of sequences $\langle a,b \rangle, \langle b,c \rangle, \langle c,a \rangle$ and $\langle b,c \rangle$.

S2. The algorithm begins by applying PrefixSpan [12] to the first batch with minimum support 0.4. The frequent sequences found, followed by their support counts, are: $\langle a \rangle:4$, $\langle b \rangle:2$, $\langle c \rangle:2$, $\langle a,b \rangle:2$, and $\langle c,a \rangle:2$. Nodes corresponding to each of these sequences are inserted into the tree T_0 with the respective counts, a batchCount of 1 and a PID of 0. The state of the tree at this point is shown in Fig. 3(a).

We then moves on to the next batch, applying PrefixSpan [12] to B_2 , again with support 0.4. The frequent sequences found, followed by their support counts, are: $\langle a \rangle:3$, $\langle b \rangle:3$, $\langle c \rangle:3$, and $\langle b,c \rangle:2$. This causes the nodes corresponding to sequences $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$ to have their counts incremented by 2, 3, and 3, respectively. In addition, each of these nodes has its batchCount variable incremented to 2. A new node is created corresponding to the sequence $\langle b,c \rangle$, having count of 2 and batchCount of 1. The state of the tree at this point is shown in Fig. 3(b).

S3: Because the pruning period is 2, we must now prune the tree. For each node, B is the number of batches elapsed since the last pruning before that node was inserted in the tree. In this case, the last pruning can be thought of as occurring at time 0. Thus, $B = 2$ for all nodes. For each node, $B' = B - \text{batchCount}$. In this case, some nodes have $B' = 0$, whereas others have $B' = 1$. According to the algorithm, we prune the tree all nodes satisfying:

$$\text{count} + B'(\lceil \alpha L \rceil - 1) \leq B * \epsilon * L$$

In this case, that reduces to: $\text{count} + B' \leq 4$. The state of the tree after pruning is shown in Fig. 4.

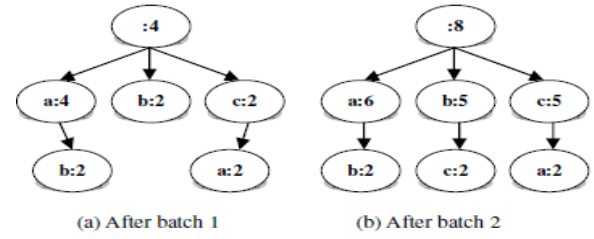
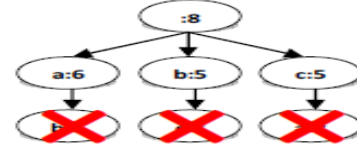
If the user requests the set of sequential patterns now, the algorithm outputs all sequences corresponding to nodes having count at least $(\sigma - \epsilon) * N = (0.7 - 0.5) * 8 = 1.6$. The output sequences and counts are: $\langle a \rangle:6$, $\langle b \rangle:5$ and $\langle c \rangle:5$.

5.2 The Motion-based SS-MB Algorithm

Style: denotes an player's style represented by this node.

Over_estimation: denotes an upper bound on the over-estimation of the count of this node in the tree compared to the true count of the sequence corresponding to this node.

Count: denotes the count of the sequence corresponding to the path from the root to this node;

Fig. 3. States of T_0 in MSS-BEFig. 4. States of T_0 in MSS-BE after pruning

Each node below the root in the tree T_0 will contain only three attributes. Another algorithm—MSS-MB algorithm and their pseudo codes are as follows.

Motion-based SS-MB (MSS-MB) Algorithm

// Tree T_0 is set initially to have the root node with its count that denotes the number of sequences occurred set to 0.

Input:

D : a incoming stream of sequences S_1, S_2, \dots ;

σ : the minimum support threshold;

ϵ : the significance threshold where $0 \leq \epsilon < \sigma$;

L : the length of batch;

m : the maximum number of nodes in the tree.

Output: Tree T_0

Begin

1. break the data stream into fixed-sized batches, where each batch contains L sequences;
2. $\text{min} = 0$;
3. **While** (each batch B_k that arrives) **do**
4. **Begin**
5. Apply PrefixSpan [1] to it, using support ϵ ;
6. **For** each frequent sequence s_i that is found, having count c_i **do**
7. **If** a path corresponding to this sequence does not exist in the tree
8. **Then** Create one new path, setting the *over_estimation* values of the new nodes to *min* and *count* values to *count + min*;
9. **Else**
10. Increment nodes' count by c_i ;
11. **EndIf**
12. **EndFor**
13. **If** the number of nodes in the tree exceeds m **Then**
14. Remove from the tree the node of minimum *count*;
15. Set *min* to equal the *count* of the last node removed.
16. **EndIf**
17. return T_0 ;
18. **EndWhile**

End // end of *MSS-BE* algorithm

Our modifications are in underline and we found that it is necessary for correctness of the algorithm according to the proof. Whenever the set of sequential patterns are needed, we can output all sequences corresponding to nodes in the tree having $count > (\sigma - \epsilon) * N$, where N is the number of sequences occurred.

Example 2 (MSS-MB). Suppose the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ϵ is 0.5, and the maximum number of nodes allowed in the tree after processing any given batch is 6. We assume that player's styles arrive as the Fig. 5 shown.

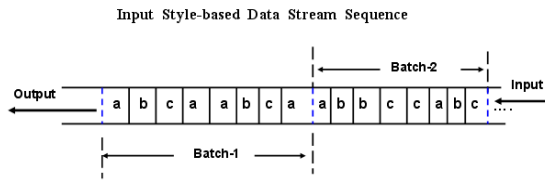


Fig. 5. Overview of incoming stream sequence for MSS-MB.

S1 Break the data stream into fixed-sized batches, where each batch contains L sequences. Let batch B_1 be consisted of sequences $\langle a,b \rangle, \langle c,a \rangle, \langle a,b \rangle$ and $\langle c,a \rangle$. Let batch B_2 be consisted of sequences $\langle a,b \rangle, \langle b,c \rangle, \langle c,a \rangle$ and $\langle b,c \rangle$.

S2. The algorithm begins by applying *PrefixSpan* to the first batch with minimum support $\epsilon = 0.5$. The frequent sequences found, followed by their support counts, are: $\langle a \rangle:4$, $\langle b \rangle:2$, $\langle c \rangle:2$, $\langle a,b \rangle:2$, and $\langle c,a \rangle:2$. Nodes corresponding to each of these sequences are inserted into the tree T_0 with the respective counts and an *over_estimation* of 0. The state of the tree after processing this batch is shown in Fig. 6(a). We then moves on to the next batch, applying *PrefixSpan* to B_2 , again with support 0.5. The frequent sequences found, followed by their support counts, are: $\langle a \rangle:2$, $\langle b \rangle:3$, $\langle c \rangle:3$, and $\langle b,c \rangle:2$. So the nodes corresponding to sequences $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$ to have their counts incremented by 2, 3, and 3, respectively. A new node is created corresponding to the sequence $\langle b,c \rangle$, having *count* as $2 + \min = 2$ and *over_estimation* as $\min = 0$.

S3: Because there are now 7 nodes in the tree and the maximum is 6, we must remove the sequence having minimum *count* from the tree. Breaking ties arbitrarily, the node corresponding to the sequence $\langle c, a \rangle$ is removed. The variable \min is set to 2 where this sequence's *count* before being deleted. The current state of the tree is shown in Fig. 6(b).

If the user requests the set of sequential patterns now, the algorithm outputs all sequences corresponding to nodes having $count > (\sigma - \epsilon)N$.

5.3. Action Prediction and Response Algorithm

Once the frequent pattern tree T_0 is created and dynamically maintained by the data stream algorithms. With a input of the player's styles in continuous game play, we can predict his next

possible style by traversing the tree and compute the probability of its appearance. The following are our algorithms.

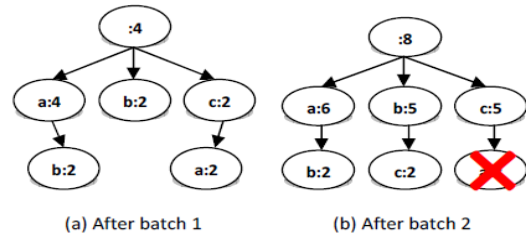


Fig. 6. States of T_0 in MSS-MB

2.3.2 Our Style Prediction Algorithm

Style Prediction(SP) Algorithm

// T_0 is the lexicographic tree to store the subsequences seen in the data stream. S_i is a player's style seen in the data stream. S_j denotes the player's possible style that follows S_i and $prob(S_j)$ is its probability of appearance.

Input: Tree T_0 and some style S_i .

Output: Predicted style S_j and its probability $Prob(S_j)$

Begin

1. **IF** there exists first level node T_k that its style match S_i **Then**
2. Look for the child node of T_k with largest *count* and the sequence corresponding to this node is frequent;
3. **IF** found **Then**
4. **Return** the style value S_j and its appearance probability $Prob(S_j) = count / N$ where N is the total sequences seen;
5. **EndIf**
6. **EndIf**
7. **Return** "Unknown" and probability value 0;
8. **End** // end of SP algorithm

To speed up the prediction and lower the errors, the tree structure was implemented and shown below.

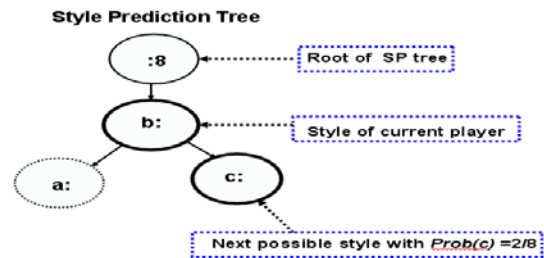


Fig. 7. Tree traversal of our SP algorithm

5.3.2 NPC Response Algorithm

// A streaming sequence of player's style such as $D = S_1, S_2, \dots$ (assumed that this is infinite)

Input: D .

Output: NPC animations results

Begin

1. **While** (each S_k that arrives) **do**
2. **Begin**
3. Predict player's possible next style and its appearance probability P using SP algorithm;
4. **If** $P \geq 0.5$ **Then**

5. Play the corresponding defensive animation followed by a attacking animation;
 6. **Else If** $P > 0$ and $P < 0.5$ **Then**
 7. Play the corresponding defensive animation;
 8. **Else**
 9. Play the corresponding attacked animation;
 10. **EndIf**
 11. **EndWhile**
- End;** // End of NPC response algorithm

Intuitively, this algorithm will mimic a human opponent's behavior during a hand-to-hand combat situation.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, the effectiveness of the proposed stream mining algorithm is investigated. All algorithms were implemented in C++ in Torque Game Engine Advanced [14] and The award winning COA Online Game environment. The experiments were run on a PC with a AMD Athlon II X3 2.8GHz CPU and 4 GB main memory, running Microsoft Windows 7 64bit.

The player's and NPC's attacking and defensive style animations are recorded using Moven motion-capture system [15]. All the styles are listed in Table 1 and can apply to both player and non-player characters.

The simulation model we used and the experimental results are provided in Section 6.1 and Section 6.2, respectively.

6.1 Test data and Simulation Model

First, our on line game model, named as Caotic Age Online, and its link is as http://coaonline.mes.stut.edu.tw/coa_imagepage/imagepage_index.php. Therefore, we can implement these algorithms in this on line game, and observe all possible results and make necessary improvements. To ensure our algorithms feasible, we use the following data stream sequence as a sample player's style data shown in Fig. 8.

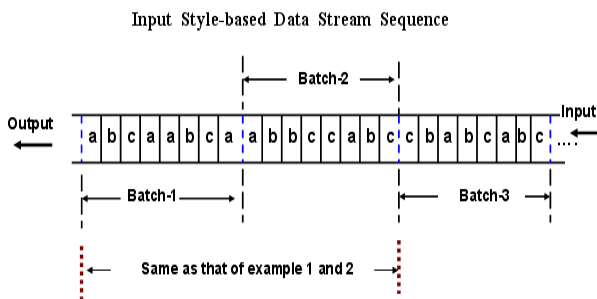


Fig. 8. Overview of incoming stream sequence for NPC response algorithm.

As a result, they can be broken into total 3 batches and each batch is composed of 4 sequences. The first 16 styles and the order are the same as listed in Example 1 where each style is illustrated in Table 1. For the MSS-BE algorithm, we assume that the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ϵ is 0.5, the batch support threshold α is 0.4, and the pruning period δ is 2. For the

MSS-MB method, the maximum number of nodes allowed in the tree after processing any given batch is 7 and the other parameters' values are same as above.

6.2 Experimental Results

In this subsection, we report our experimental results both on the MSS-BE and the MSS-MB algorithms. Both algorithms can achieve real-time response running in a separate working thread.

By observing both Fig. 9 and 10, we can easily realize that, in the first 16 styles that player performs NPC responses in just the same behavior for both methods. The main difference shows afterward in the third batch coming up. Because all the second level nodes of MSS-BE tree are pruned after batch 2, therefore we are not able to predict the next possible style during the third batch period.

In summary, the final comparisons are listed in Table 2. Besides, these parameters emphasize the significance and can derive different results in our implementation. In addition, the MSS-MB algorithm is better overall for our style prediction algorithm by exploiting all of the available system memory. The interesting relationship between the minimum support threshold σ and the significance threshold ϵ is yet to be determined, and will be an interesting topic.



Fig.9. Legends shown in the attached media files.(Color Plate 2)



Fig. 10. The simulation in action using MSS-BE algorithm.



Fig. 11. The simulation in action using MSS-MB algorithm.

TABLE 2: RESULTS COMPARISONS

| | | Parameter-Setup | | |
|---------|------------|---|--|--|
| | | $\sigma \geq 0.75,$ $\delta = 1 \text{ or } 2$ | $\sigma = 0.7,$ $\delta = 1$ | $\sigma = 0.7,$ $\delta = 2$ |
| Methods | MSS- BE | Unpredictable- after Batch-1 . | Unpredictable- after Batch-1 . | Unpredictable- after Batch-2 . |
| | MSS- MB | Unpredictable- after Batch-2 . | OK | OK |

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we propose two efficient tree-based algorithms, called MSS-BE, for mining the set of frequent stream patterns over data streams with bounded-error; the other is called MSS-MB, for mining the set of frequent stream patterns over data streams with memory-bounded. The differences between them lie on the purpose. MSS-BE focus on the number of errors under the user control; on the other side, MSS-MB pay attentions on the upper bound of used main memory. Based on these algorithms, we develop an on-line and real-time model for predicting response actions of opponents in on line game. Experiments show that the proposed algorithms not only attain highly accurate mining results, but also run significant faster and consume less memory than do existing algorithms, such as vector-based on line game.

In the future work, we still investigate correlations among the parameters suggested in our approaches. Besides, more complex situations between NPC and opponents will be our next goals and researches.

REFERENCES

- [1] V.B. Zordan, A. Majkowska, B.Chui, M. Fast. Dynamic response for motion capture animation. *ACM Transaction on Graphics*, 24(3): pp. 697-701, 2005.
- [2] Hubert Shum, Taku Komura, and Shuntaro Yamazaki. Simulating interactions of avatars in high dimensional state space. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D 2008)*, pp.131-138, 2008.
- [3] A.d. Lattner, A. Miene, U. Visser, and O. Herzog: Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In: *RoboCup 2005: Robot Soccer World Cup IX*, 4020, pp.118-129, 2005.
- [4] M. C Mozer: Lessons from an adaptive home. In: *Smart Environments: Technology, Protocols, and Applications*, ed. D. Cook and R. Das, 2005, 273-298, 2005.
- [5] G. Adomavicius and A Tuzhilin.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 2005, 17(6): 734-749, 2005.
- [6] M. Charikar, K. Chen, F Marach-Colton: Finding frequent items in data streams. In *ICALP'02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming 2002*, London, UK, pp.693-703, 2002.
- [7] H. Shum, and T. Komura. Generating realistic fighting scenes by game tree. *SIGGRAPH/ Eurographics Symposium on Computer Animation (SCA 2006)*, 2006.
- [8] L.F. Mendes, B., Ding, and J. Han. Stream sequential pattern mining with precise error bounds, in *Eighth IEEE International Conference on Data Mining (ICDM'08)*, pp. 941-946, 2008.
- [9] M. Sužnjević, O. Dobrijević, M. Matijašević. MMORPG player actions: network performance, session patterns and latency requirements analysis, *Multimedia tools and applications*, 45(1-3): pp.191-214, 2009.
- [10] J. Lee, and K.H. Lee. Precomputing avatar behavior from human motion data. In *Proc of SCA 2004*, pp.79-87, 2004.
- [11] Xsens Moven, <http://www.xsens.com>

- [12] J. Pei, et. All. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, *IEEE Trans. on Knowledge and Data Engineering*, 16(11): pp. 1424-1440, 2004.
- [13] Torque Game Engine Advanced, <http://www.torquepowered.com/>
- [14] P.H. Shum, Hubert, Komura, Taku, and Yamazaki, Shuntaro. Multiple character interactions with collaborative and adversarial goals, *IEEE Trans. on Visualization and Computer Graphics 2011*, accepted.
- [15] K.J. Shim, R. Sharan, and J. Srivastava. Player performance prediction in massively multiplayer online role-playing games (MMORPGs). *PAKDD (2) 2010*, pp.71-80, 2010.
- [16] K.J. Shim and J. Srivastava. Sequence alignment based analysis of player behavior in massively multiplayer online role-playing games (MMORPGs). *ICDM Workshops 2010*, pp. 997-1004, 2010.



Chih Ming Chiu received the MS degree in Computer Science and Information Engineering from Arizona State University in 1993. He is currently working toward to the Ph.D. degree at the National Cheng Chung University. His research interests include computer animation, MMORPG, virtual reality, P2P, and Game AI. He is a member of the ACM and the IEEE Computer Society.



Shao-Shin Hung received the MS degree and Ph.D. degree in Computer Science and Information Engineering from National Cheng Chung University in 1993 and in 2007, respectively. His research interests include spatial data bases, data mining, MMORPG, P2P, intrusion detection system, knowledge management, ontology system, and social mining. He is a member of the ACM, EG, and the IEEE Computer Society.



Jyh-Jong Tsay is an Associate Professor at the Department of Computer Science and Information Engineering at National Chung Cheng University, Chiayi, Taiwan, Republic of China. He completed his PhD Degree at Purdue University, USA, in 1990. His research interests include data mining, machine learning, information retrieval and game AI.