Intelligent-based Latency Reduction in 3D Walkthrough

Tsou Tsun Fu1, Shao-Shin Hung2, Hui-Ling Lin3, Derchian Tsaih4 and Jung Tzung Chen5
1Department of Information Management, WuFeng Institute of Technology, Taiwan, R.O.C
2Department of Computer Science and Information Engineering, WuFeng Institute of Technology, Chiayi, Taiwan, R.O.C
3Department of International Business, Ling Tung University, Taiwan, R.O.C
4Department of Electronic Commerce Management, Nanhua University, Chiayi, Taiwan, R.O.C
5Department of Electronic Engineering, WuFeng Institute of Technology, Chiayi, Taiwan, R.O.C
1ttfu2004@gmail.com; 2hss@cs.ccu.edu.tw; 3ljoyce@mail.ltu.edu.tw; 4dtsaih@mail.nhu.edu.tw; 5jtchen@mail.wfc.edu.tw

Abstract: In many visualization applications (VA), the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off storage media is expensive. However, object correlations are common semantic patterns in VA. They can be exploited for improve the effectiveness of storage caching, prefetching, data layout, and disk scheduling. However, little approaches for discovering object correlations in VA to improve the performance of storage systems. In this paper, we develop a class of view-based projection-generation method for mining various frequent sequential traversal patterns in the VA. The frequent sequential traversal patterns are used to predict the user navigation behavior. Furthermore, the hypergraph-based clustering scheme can help reduce disk access time with proper placement patterns into disk blocks. Finally, we have done extensive experiments to demonstrate how these proposed techniques not only significantly cut down disk access time, but also enhance the accuracy of data prefetching.

Keywords: Visualization, prefetching, correlation, hypergraph, view, projected, storage

1. Introduction
The information revolution is changing the way many people live and think. Vast quantities and diverse types of information are being generated, stored, and disseminated, raising serious issues about how to make such information usable. The need to understand and extract knowledge from stored information is becoming a ubiquitous task. As data management problems have grown, I/O performance has deteriorated, primarily due to the enlarging gap between the performance of the CPU and secondary storage system. Meanwhile, higher processing power and larger storage capacity enable users to produce and save more data, so visualization tools need to process larger volumes of time-series data and their periodic input operations are now more bottleneck-prone than ever [1,2,6,17,24]. On the other side, to satisfy the growing demanding for fidelity, there is a need for interactive and intelligent schemes that assist and enable effective and efficient storage management.

Unfortunately, it is not an easy task to exploit the intelligence in storage systems. One primary reason is the system latency between VA and storage systems. In such a case, VE do not consider the problem of access times of objects in the storage systems. They always simply concerned about how to display the object in the next frame. As a result, the VE can only manage data at the rendering and other related levels without knowing any semantic information such as semantic correlations between data. This motivates a more powerful analysis tool to discover more complex patterns, especially semantic patterns, in storage systems. Therefore, the aim of our work is to decrease this latency through intelligent organization of the access data and enabling the clients to perform predictive prefetching. Of course, selecting the right patterns
requires knowledge of the access patterns and the storage resources. On the other side, data placement requires similar knowledge plus information about workloads will interact when sharing some patterns [6,13]. In this paper, we consider the problem and solve this using data mining techniques [5]. Clearly, when users traverse in a virtual environment, some potential semantic characteristics will emerge on their traversal paths. If we collect the users’ traversal paths, mine and extract some kind of information of them, such meaningful semantic information can help to improve the performance of the interactive VA. For example, we can reconstruct the placement order of the objects of 3D model in disk according to the common section of users’ path. Exploring these correlations is very useful for improving the effectiveness of storage caching, prefetching, data layout, and disk scheduling [1,2,6,13,17,24].

This paper uses our previous mining method-VSPM [25](Viewed-based Sequential Pattern Mining), a method which applies a data mining technique called frequent sequential pattern mining to discover object correlations in VE. Specially, we have modified several recently proposed data mining algorithms called FreeSpan [5] and PrefixSpan [11] to find object correlations in several traversal traces collected in real systems. To the best of our knowledge, VSPM is the first approach to infer object correlations in a VA. Furthermore, VSPM is more scalable and space-efficient than previous approaches. It runs reasonably fast with reasonable space overhead, indicating that it is a practical tool for dynamically inferring correlations in a VA. Besides, we have also proposed two clustering methods to cluster the similar patterns for reducing the access time. One is based on the idea of co-occurrence of transaction data have developed. They are usually measured by Jaccard coefficient \( \text{SIM}(T_1, T_2)= \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} \) [22]. The other clustering scheme is based on the hypergraph-based model. In this model, the vertex set corresponds to the distinct objects in the VA and the hyperedges correspond to the frequent sequential patterns. Both of them will make similar objects much closer to be accessed in one time. These result in less access times and much better performance. We also compare the distinctions between them. Moreover, we also have evaluated the benefits of object correlation-directed prefetching and disk data layout using the real system workloads.

The rest of this paper is organized as follows. Related works are given in Section 2. In Section 3, we describe our problem formulation. Our algorithms are suggested in Section 4. Section 5 presents our experiment results. Finally, we summarize our current results with suggestions for future research in Section 6.

### 2. Related work

In this subsection, we will briefly describe related works about data layouts, sequential pattern mining and pattern clustering, respectively.

#### 2.1 Data Layout in Visualization Methods

Current researches focus on visibility-based prefetching algorithm for retrieving out-of-core 3D models and rendering them at interactive rates [16]. The goal of prefetching through the multithreading mechanism is to have the geometry already in memory by the time it is needed. But the threads will occupy some of the main memory and this strategy need well-planned switching mechanism to handle threads. Especially, for large datasets in virtual environments, this scheme can not scalable. P. J. Rhodes et. al. [17] proposes iterators and threaded prefetching scheme based on the concept of spatial prefetching for improvement on I/O performance. Yoon et. al. [13] discuss the cache-efficient layout of bounding volume hierarchies (BVHs) of polygonal models. They also introduce a new probabilistic model to predict the running access patterns of a BVH. Since such large BVH-based \textit{kd}-trees will be stored in the storage system for access, this will result in large I/O times. Chisnall et. al. [1] present knowledge-based out-of-core prefetching algorithms without using hard-coded rendering-related logic. By utilizing the access history and patterns dynamically, and adapt their prefetching strategies accordingly. However, it seems to be weak for the basis for such knowledge-based out-of-core algorithm of LRU-related schemes. Semantic correlations seem lack in this scheme.
To meet these requirements, an appropriate data structure and an efficient technique should be developed with the constraints of memory consumptions.

2.2 Sequential Pattern Mining Methods
Sequential pattern mining was first introduced in [8], which is described as follows. A sequence database is formed by a set of data sequences. Each data sequence includes a series of transactions, ordered by transaction times. This research aims to find all the subsequences whose ratios of appearance exceed the minimum support threshold. In other words, sequential patterns are the most frequently occurring subsequences in sequences of sets of items. A number of algorithms and techniques have been proposed to deal with the problem of sequential pattern mining. Many studies have contributed to the efficient mining of sequential patterns [5,11]. Almost all of the previously proposed methods for mining sequential patterns are apriori-like [5]. Sequential pattern mining algorithms, in general, can be categorized into three classes: (1) Apriori-based: horizontal partition methods and GSP [7] is one known representative; (2) Apriori-based: vertical partition methods and SPADE [6] is one example; (3) projection-based pattern growth method, such as the famous FreeSpan [11] and PrefixSpan algorithms [5].

2.3 Hypergraph-based Clustering Methods
The fundamental clustering problem is to partition a given data set into groups (clusters), such that data points in a cluster are more similar to each other (i.e., intra-similar property) than points in different clusters (i.e., inter-similar property) [10]. Although there are many clustering algorithms presented above, they can not be applied to our data set directly. These discovered clusters are used to explain the characteristics of the data distribution [26]. However, these schemes fail to produce meaningful clusters, if the number of objects is large or the dimensionalities of the VE (i.e., the number of different features) are diverse and relatively large.

In this paper, we propose a new methodology for clustering correlated objects using frequent sequential patterns, and clustering related patterns using clusters of objects. These frequent sequential patterns are used to group objects into hypergraph edges, and a hypergraph partition algorithm is used to find the clusters. The knowledge that is represented by clusters of related objects can also be used to effectively cluster the actual semantic patterns by looking at the clusters that these objects belong to. Therefore, we can layout these clusters onto the storage systems for prediction of user traversal.

3 Problem Formulation
In this section, we extract the useful information in the access history in the form of sequential patterns. In order to mine for sequential patterns, we assume that the continuous client requests are organized into discrete sessions. Sessions specify user interest periods and a session consists of a sequence of client requests for data items ordered with respect to the time of reference. The client request consists of the objects which a client browse and traverse at will in the VE. We denote this type of clients request as view. A session consists of one or more views. In correspond to with terminologies used in data mining, a session can be considered as a sequence. The whole database is considered as a set of sequences. Formally, let \( \Sigma = \{l_1, l_2, \ldots, l_m\} \) be a set of \( m \) literals, called objects (also called items) [18]. The view \( v \) is defined as snapshot of sets of objects which a user observes duration the period. A view (also called itemset) is an unordered, non-empty set of objects. A sequence is an ordered list of views. We denote a sequence \( s \) (also called transaction) by \( \{v_1, v_2, \ldots, v_n\} \), where \( v_j \) is a view and ordered property is obeyed. We also call \( v_j \) an element of the sequence. An item can occur only once in an element of a sequence, but can occur multiple times in different elements. We assume, without loss of generality, that items in an element of a sequence are in lexicographical order.

A sequence \( <a_1, a_2, \ldots, a_n> \) is contained in another sequence \( <b_1, b_2, \ldots, b_m> \) if there exist integers \( i_1 < i_2 < \ldots < i_n \) such that \( a_i \subseteq b_{i_1} ; a_2 \subseteq b_{i_2} ; \ldots ; a_n \subseteq b_{i_n} \). For example, \( <(a, b, c)(a, d, e)> \) is contained in \( <(a, b) (b, c)(a, b, d, e, f)> \), since \( (a) \subseteq (a, b) \), \( (b, c) \subseteq (b, c) \), and \( (a, d, e) \subseteq (a, b, d, e, f) \).
\( \subseteq (a, b, d, e, f) \). However, the sequence \(<(c)(d)>\) is not contained in \(<(c, d)>\) and vice versa. The former represents objects \( c \) and \( d \) being observed one after the other, while the latter represents objects \( c \) and \( d \) being observed together. In a set of sequences, a sequence \( s \) is maximal if \( s \) in not contained in any other sequence. Let the database \( D \) be a set of sequences and ordered by increasing recording time. Each sequence records each user’s traversal path in the walk through system. The \textit{support} for a sequence is defined as the fraction of \( D \) that “contains” this sequence. A \textit{sequential pattern} \( p \) is a sequence whose \textit{support} is equal to or more than the user-defined threshold. \textit{Sequential pattern mining} is the process of extracting certain sequential patterns whose support exceeds a predefined minimal support threshold. Given a database \( D \) of client transactions, the problem of mining sequential patterns is to find the maximal sequences among all sequences that have a certain user-specified minimum support. Each maximal sequence represents a \textit{sequential pattern}.

Finally, we will define our problem in two phases. Phase I: given a sequence database \( D = \{s_1, s_2, \ldots, s_n\} \), we design an efficient mining algorithms to obtain our sequential patterns \( P \); phase II: In order to reduce the disk access time, we distribute \( P \) into a set of clusters, such that minimize inter-cluster similarity and maximize intra-cluster similarity.

4 \textbf{Our Algorithms}

In this section, we will present our algorithms to support our arguments through demonstrations.

4.1 \textbf{Pattern-based Clustering for Disk Layout}

Clustering is a good candidate for inferring object correlations in storage systems. As the previous sections mentioned, object correlations can be exploited to improve storage system performance. First, correlations can be used to direct prefetching. For example, if a strong correlation exists between objects \( a \) and \( b \), these two objects can be fetched together from disks whenever one of them is accessed. The disk read-ahead optimization is an example of exploiting the simple data correlations by prefetching subsequent disk blocks ahead of time. Several studies \([10,15]\) have shown that using these correlations can significantly improve the storage system performance. Our results in Section 6.2 demonstrate that prefetching based on object correlations can improve the performance much better than that of non-correlation layout in all cases.

A storage system can also lay out data in disks according to object correlations. For example, a object can be collocated with its correlated objects so that they can be fetched together using just one disk access. This optimization can reduce the number of disk seeks and rotations, which dominate the average disk access latency. With correlation-directed disk layouts, the system only needs to pay a one-time seek and rotational delay to get multiple objects that are likely to be accessed soon. Previous studies \([9]\) have shown promising results in allocating correlated file blocks on the same track to avoid track-switching costs.

The hypergraph was introduced by Berge \([3]\) and has been considered as a useful tool to analyze the structure of a system and to model a partition, covering, and clustering. A hypergraph \( H=(V, N) \) is defined as a set of vertices and a set of hyperedges (nets) \([26]\). Every net \( n_j \in N \) is a subset of vertices, i.e., \( n_j \subseteq V \). The size of a net \( n_j \leq N \) is equal to the number of vertices it has, i.e., \( s_j=|n_j| \). Weight \( (w_i) \) and cost \( (c_i) \) can be assigned to the vertices \( (v_i \in V) \) and edges \( (n_j \in N) \) of the hypergraph, respectively. \( K = \{V_1, V_2, \ldots, V_k\} \) is a \textit{K-way partition} of \( H \) if (a) each partition is a nonempty subset of \( V \), (b) partitions are pairwise disjoint, and (c) union of \( K \) partitions is equal to \( V \).

4.2 \textbf{Similarity Measure for Jaccard function}

In the simplified hypothesis that frequent patterns do not contain frequencies, but behave simple as Boolean vectors (like a value 1 corresponds to the presence and a value 0 corresponds to the absence), and a more intuitive but equivalent way of defining the \textit{Jaccard distance function} can be provided. This measure captures our idea of similarity between items, which are directly
proportional to the number of common values, and inversely proportional to the number of different values for the same item.

**Definition 1:** Intra-distance measure (Co-occurrence)
Let \( P_1 \) and \( P_2 \) be two sequential patterns. We can represent \( D(P_1, P_2) \) as the normalized difference between the cardinality of their union and the cardinality of their intersection:

\[
D(P_1, P_2) = 1 - \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|}
\]

(1)

**Example 2 (Intra-distance measure).**
Let \( P_1 \) and \( P_2 \) be two sequential patterns: \( P_1 = \langle a, b, c \rangle, \langle b, c, d \rangle, \langle e, f \rangle \rangle \) and \( P_2 = \langle a, b, c, d \rangle, \langle e, f, g \rangle \rangle \). The distance between \( P_1 \) and \( P_2 \) is

\[
D(P_1, P_2) = 1 - \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|} = 1 - \frac{|a, b, c, e, f|}{|a, b, c, d, e, f, g|} = 1 - \frac{5}{7} = \frac{2}{7}
\]

**4.3 Sequential Pattern-based Clustering Algorithms**
Intuitively, a cluster representative for virtual environment data should model the content of a cluster, in terms of the objects that are most likely to appear in a pattern belonging to the cluster. A problem with the traditional distance measures is that the computation of a cluster representative is computationally expensive. As a consequence, most approaches [6] approximate the cluster representative with the Euclidean representative. However, those approaches may suffer the following drawbacks:

- Huge cluster representatives cause poor performances, mainly because as soon as the clusters are populated, the cluster representatives are likely to become extremely huge.
- For different kinds of patterns, it seems to be difficult to find the proper cluster representatives.

In order to overcome such problems, we can compute an approximation that resembles the cluster representatives associated to Euclidean and mismatch-count distances. Union and intersection seem good candidates to start with. Since our clustering operations are based on set operations, we ignore the order of frequent patterns.

**4.3.1 Jaccard-based Clustering Algorithms**
To avoid these undesired situations, we supply three tables. The first table is \( FreqTable \). It records the frequency of any two patterns co-existing in the database \( D \). The second table is \( DistTable \). It records the distance between any two patterns. The last table is \( Cluster \). It records how many clusters are generated. The following is our clustering algorithm.

**Pattern Clustering Algorithm for Jaccard Function**

```
Input: P and T
Output: T
Begin
1. \( FreqTable = \{ ft_{ij} | \text{the frequency of pattern } i \text{ and pattern } j \text{ co-existing in the database } D \}; \)
2. \( DistTable = \{ dt_{ij} | \text{the distance between pattern } i \text{ and pattern } j \text{ in the database } D \}; \)
3. \( C_1 = \{ C_i \} \text{ At the beginning each pattern to be a single cluster } \}
4. // Set up the Extra-Similarity Table for evaluation
5. \( M_1 = \text{Intra-Similar } (C_1, \emptyset); \)
6. \( k = 1; \)
7. while \( |C_k| > n \) do Begin
8. \( C_{k+1} = \text{PatternCluster } (C_k, M_k, FreqTable, DistTable); \)
9. \( M_{k+1} = \text{Intra-Similar } (C_{k+1}, M_k); \)
10. \( k = k + 1; \)
11. End;
12. return \( C_k \);
13. End;
```

**4.3.2 Hypergraph_based Clustering Algorithms**
As mentioned before, the vertex set corresponds to the distinct objects in the VE and the hyperedges correspond to the frequent sequential patterns [20,23]. The weight of hyperedge is the support of that sequential pattern. In this paper, we adopt the hypergraph partition algorithm in [20]. Since there are no expensive data structures or special constraints hidden, we can implement them very efficient and time/space complexity also meet our demands.

**5 Performance Evaluation**
A traversal path database was recorded each user’s traversal path and used for mining target. The simulation model we used and the experimental results are provided in Section 5.1 and Section 5.2, respectively.

5.1 Test data and Simulation Model

We use the virtual power plant model from http://www.cs.unc.edu/~walk/ created by Walkthrough Laboratory of Department of Computer Science of University of North Carolina at Chapel Hill. The Power Plant Model is a complete model of an actual coal fired power plant. The model consists of 12,748,510 triangles. Its size is 128 MBytes. Our traversal database keeps track of the traversal of the power plant by many anonymous, randomly users. For each user, the data records list all the areas of the power plant that user visited in a one week timeframe. Each path consists of 30 ~ 40 views. Each view consists of 20~30 objects on average. The number of objects is 11,949, where each object is a some meaningful combination of triangles of power plant and it is considered as a data item.

5.2 Experimental Results and Studies

In this subsection, we report our experimental results on the VSPM algorithm. Since GSP and SPADE are the two most important sequential pattern mining algorithms, we conduct an extensive performance study to compare VSPM with them. To evaluate the effectiveness and efficiency of the VSPM algorithm, we performed an extensive performance study of GSP, SPADE, FreeSpan, and PrefixSpan, on real data sets, with various kinds of sizes and data distribution. Besides, these four algorithms, GSP, SPADE, FreeSpan, and PrefixSpan were implemented in Java.

Table 3: Parameters for our traversal data set

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of data sequences (i.e., size of database)</td>
</tr>
<tr>
<td>C</td>
<td>Average number of transactions per data sequence</td>
</tr>
<tr>
<td>F</td>
<td>Average number of items per transaction</td>
</tr>
<tr>
<td>A</td>
<td>Average length of maximal possible frequent sequences</td>
</tr>
<tr>
<td>D</td>
<td>Average size of itemsets in maximal possible frequent sequences</td>
</tr>
<tr>
<td>M</td>
<td>Number of maximal potentially frequent sequences</td>
</tr>
<tr>
<td>N0</td>
<td>Number of maximal potentially frequent itemsets</td>
</tr>
<tr>
<td>N1</td>
<td>Number of items</td>
</tr>
</tbody>
</table>

Figure 4: Execution time with respect to various support thresholds using our real data set-1.

Virtual environment traces

The performance of our traversal data base is reported as follows. First, we follow the procedure described in [4] to set up out data set parameters. The meanings of all parameters are listed in Table 3. Figure 4 and 5 show the performance comparison among the five algorithms for our virtual environment data set. From Figure 4 and 5, we can see that VSPM is as efficient as PrefixSpan does, but it is much more efficient than SPADE, FreeSpan, and GSP.

5.2.1 Experimental Results on Mining Unit
5.2.2 Experimental results on clustering unit

For quality measure of clustering result, we adopted the cluster cohesion and the inter-clustering similarity. All are defined as follows.

Definition 2: Large item
Given a pattern, and a user-defined threshold \( \theta \), if it satisfies the following criterion:

\[
0 < \text{minimum support threshold} < \theta \leq \text{support(pattern)} \leq 1.
\]

We call the pattern \( i \) as a large item.

Definition 3: Cluster Cohesion (Cluster-Coh(\( C_i \)))
It is the ratio of the large items to the whole items \( T(C_i) \) in the cluster \( C_i \). This is calculated by the following formula, and if it is near 1, it is a good quality cluster; otherwise, it is not.

\[
\text{Cluster-Coh}(C_i) = \frac{C_i(L)}{T(C_i)}
\]

where \( C_i(L) \) is the number of large item in cluster \( C_i \) and \( T(C_i) \) is number of all items in cluster \( C_i \).

Definition 4: Inter-Cluster Similarity (\( \text{inter-sim}(C_i, C_j) \))
It is based on the large items is the rate of the common large items of the cluster \( C_i \) and \( C_j \). We calculate the inter-cluster similarity by the following formula, and if it is near 0, it is the good clustering. Otherwise, it is not.

\[
\text{Inter-sim}(C_i, C_j) = \frac{\text{LarCom}(C_i \cap C_j)}{C_i(L) + C_j(L)} \times \frac{\text{LarCom}(C_i \cap C_j)}{\text{LarCom}(C_i \cup C_j)}
\]

where \( \text{LarCom}(C_i \cap C_j) \) is the number of common large items in the cluster \( C_i \) and \( C_j \), \( |\text{LarCom}(C_i \cap C_j)| \) is the total occurrence number of the common large items, and \( |\text{LarCom}(C_i \cap C_j)| \) is the total occurrence number of the large items in the cluster \( C_i \) and \( C_j \).

Definition 5: View-radius
The view radius is defined as the radius of the visible circle in the virtual environments. As the radius increases, the more objects are observed. In other words, it controls how many objects are observed at the same time in one view.

In the meanwhile, we select the different support threshold for comparison. Figure 6 and 7 show the results. Algorithms with clustering outperforms other algorithms without clustering. Since the clustering mechanisms can accurately support prefetching objects for future usage. Not only the access time is cut down but also the I/O efficiency is improved. Note that HG_clustering represents the Hypergraph clustering scheme.

Figure 6: Comparison of different algorithms on the number of objects retrieved under the same view_radius.

Figure 7: Comparison of different algorithms on system response time under the same view_radius.

Figure 8: Comparison of different support threshold on cluster cohesion under the same view_radius.

Figure 9: Comparison of different support threshold on inter-cluster similarity under the same view_radius.

By observing both Figure 8 and 9, we can easily realize that there exist relations between the number of clusters and inter-cluster similarity, and also between the number of clusters and...
cluster cohesion. Among them, the HG_clustering outperforms the other two. Since HG_clustering scheme can capture the inter-/intra- relationships in clusters as many as possible, the Jaccard clustering does less and the last one does nothing, just based on its random behavior.

In summary, we can determinate that our HG_clustering algorithm is better overall at cluster cohesion and inter-cluster similarity. This means that our HG_clustering algorithm can groups more similar patterns together and do more improvements on the efficiency of storage systems.

6 Conclusion

In this paper, we have designed a frequent projection-based sequential pattern mining algorithm to find correlations among objects. Using the VA traces, our experiments show that VSPM is an efficient algorithm. Besides, we have evaluated correlation-directed prefetching and data layout. Our experimental results have shown that correlation-directed prefetching and data layout can improve I/O average response time by 1.998 to 3.201 compared to no-prefetching, and 3.102 to 9.121 compared to the number of retrieved objects. Finally, we have also designed two criteria to verify the validity of clustering method.

Our study has several limitations. One important limitation is that our disk layout was not especially designed for the extra long frequent sequential patterns. This direction will enhance the system performance.

References


