Overlapping Matrix Pattern Visualization: a Hypergraph Approach

Ruoming Jin   Yang Xiang   David Fuhry   Feodor F. Dragan
Department of Computer Science
Kent State University, Kent, OH 44242
{jin,yxiang,dfuhry, dragan}@cs.kent.edu

Abstract

In this work, we study a visual data mining problem: Given a set of discovered overlapping submatrices of interest, how can we order the rows and columns of the data matrix to best display these submatrices and their relationships? *We find this problem can be converted to the* hypergraph ordering *problem, which generalizes the traditional* minimal linear arrangement *(or* graph ordering*) problem and then we are able to prove the NP-hardness of this problem. We propose a novel iterative algorithm which utilize the existing graph ordering algorithm to solve the optimal visualization problem. This algorithm can always converge to a local minimum. The detailed experimental evaluation using a set of publicly available transactional datasets demonstrates the effectiveness and efficiency of the proposed algorithm.*

1 Introduction

Given a set of discovered submatrices of interests, how can we order the rows and columns of the data matrix to best display these submatrices and their relationships? For example, the right matrix reveals much richer information about the four submatrix patterns than the left one. This is a central problem emerging from the visualization requirement of a wide range of data mining tasks [8; 13; 24]:

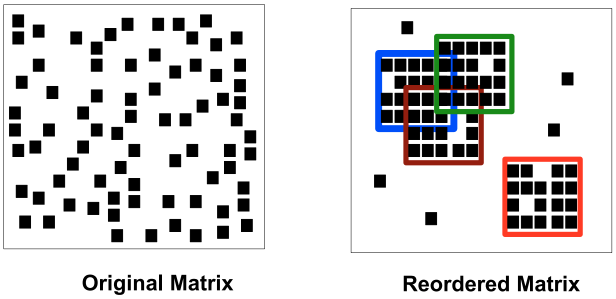


Figure 1. An example of matrix pattern visualization

Overlapping Bicluster Visualization: Gene-expression data is commonly represented as a matrix, where each gene corresponds to a row and each experimental condition corresponds to a column. Each element of this matrix represents the expression level of the gene under a specific condition. Often, this matrix can be converted into a binary matrix by considering that each gene is either "on" or "off". The typical pattern discovery task, often referred to as bi-clustering [14], would find "homogeneous" submatrices, which are composed subsets of genes and conditions: the genes are coregulated or coexpressed under the conditions in the corresponding submatrices. Recently, there is a lot of interest in discovering overlapping bi-clusters [8; 13]. Considering we have a list of most interesting submatrices (bi-clusters) which overlap each other, how can we reorder the rows and columns of the entire matrix so that we can visually inspect the relationship between these submatrices?

Transactional Data Visualization: The shopping-basket data is one of the most studied data types in data mining. Here each transaction corresponds to a row and each item corresponds to column. The element of the binary matrix records if the transaction purchased the item or not. Recently, there is an increasing interest in summarizing the data using a set of "dense" binary matrices [26; 5; 27]. In a nutshell, the dense submatrix contains almost all 1s, and a list of them can cover all the 1s in the entire matrix with small false positive rate. Thus, the dense submatrix is also closely related to the approximate frequent itemset pattern. Given this, a similar problem occurs: how can we visualize the entire matrix so that the dense submatrices of interests and their relationships can be inspected?

Clearly, this task is very important in its own right and complementary to some of the most critical and widely used data mining tasks, such as bi-clustering and association rule mining. However, it is not a typical data mining task, but belongs to the area of *visual data mining* or information visualization [11; 4]. Visual data mining can be largely partitioned into two categories: data visualization [11] and pattern visualization [28; 25]. In the first category, the goal is to provide the user an overview of the data. This is especially important for the high dimensional data and other types of structure or text data, where a direct $2D$ view does not exist. In this study, we will visualize high-dimensional transactional data through its matrix representation. Note that matrix visualization has been a useful tool for visualizing relational datasets, such as graphs [11]. In the second category, we are interested in a visual representation of those already discovered patterns or other mining models, such as association rules and decision trees [28; 25]. Interestingly, our problem is a combination of the two categories: 1) we are interested in matrix visualization, by applying discovered patterns to guide the visualization; and 2) we visualize the discovered patterns through the matrix representation.

Matrix visualization has a deep root in numerical computation [7] and is closely related to graph theory [16]. Mueller et al. [16] applied three major classes of algorithms for (symmetric) similarity matrix visualization: simple graph theoretic algorithms, symbolic sparse matrix reordering algorithms and spectral decomposition algorithms. Here, our problem setting is more general, in the sense that their symmetric matrix can be viewed as a special case of our transactional matrix. In addition, our problem is more challenging as our goal is to visualize the discovered dense submatrices. Their techniques do not consider such constraints, and our submatrices or substructures of interest cannot be meaningfully identified in their visual representation of the matrix.

1.1 Problem Definition

Let the transaction database DB be represented as a binary matrix such that a cell (i, j) is 1 if a transaction i contains item j, otherwise 0. For convenience, we also denote the database DB as the set of all cells which are 1, i.e., $DB = \{(i, j) : DB[i, j] = 1\}$. Let $\mathcal{T}$ denote the set of all transactions and $\mathcal{I}$ denote the set of all items in DB. We define the hyperrectangle H as the Cartesian product of a transaction set T and an item set I, i.e. $H = T \times I = \{(i, j) : i \in T \text{ and } j \in I\}$. Thus, the submatrix of DB can be represented as $DB[H]$, where H records the element location of the submatrix. Since in our visualization, our main target is the submatrix location in the entire matrix, we will simply use a hyperrectangle to represent the submatrix, and interchangeably use the terms submatrix, submatrix pattern, and hyperrectangle.

Let P be a set of discovered and user-specified submatrix patterns (hyperrectangles), $\{H_1, H_2, \cdots, H_p\}$. Note that these hyperrectangles may overlap with each other. We are interested in visualizing them in a matrix representation. Ideally, we hope all the elements of each hyperrectangle are aggregated together so that we can tell the rough shape of each hyperrectangle. This can also help us to visually discover their intricate relationships.

Given this, we introduce a goodness function to measure the matrix visualization as follows.

Definition 1 (Visualization Cost) *Given a database DB with a set of hyperrectangles P, and two orders σ_T (the order of transactions) and σ_I (the order of items), we define the visualization cost of $P = \{H_1 = \{T_1 \times I_1\}, H_2 = \{T_2 \times I_2\}, \cdots, H_p = \{T_p \times I_p\}\}$ to be*

$$visual\_cost(P, \sigma_T, \sigma_I) = \\ \sum_{j=1}^{p}(\max_{t_u \in T_j} \sigma_T(t_u) - \min_{t_w \in T_j} \sigma_T(t_w)) + \\ \sum_{j=1}^{p}(\max_{i_u \in I_j} \sigma_I(i_u) - \min_{i_w \in I_j} \sigma_I(i_w))$$

Here, for a hyperrentagle H_j, $\max_{t_u \in T_j} \sigma_T(t_u)$ and $\min_{t_w \in T_j} \sigma_T(t_w)$, are the last row and the first row of this submatrix appearing in the entire matrix, respectively. Similarly, $\max_{i_u \in I_j} \sigma_I(i_u)$ and $\min_{i_w \in I_j} \sigma_I(i_w)$, are the last column and the first column of H_j appearing in the entire matrix, respectively.

Basically, we define the visualization cost of each hyperrectangle using the *perimeter* of the rectangle to enclose all the rows and columns of the submatrix. More precisely, the cost is the half of the perimeter. Then, we define the total visualization of the set P as the sum of the cost of all the hyperrectangles in P. Another reasonable goodness measure is the *area* of each rectangle covering the submatrix. We choose the perimeter for technical purposes, which will allow us to process the row order and column order independently. The advantage of this choice will be seen more clearly later.

The central problem of this paper is thus formulated as follows.

Definition 2 (Matrix Optimal Visualization (MOV) Problem) *Given a database DB with a set of hyperrectangle P, we would like to find the optimal orders σ_T and σ_I, such that $visual\_cost(P, \sigma_T, \sigma_I)$ is minimized:*

$$argmin_{\sigma_T, \sigma_I} visual\_cost(P, \sigma_T, \sigma_I)$$

We note Grothaus *et. al* [8] consider a rather similar question to ours: to produce a two-dimensional data layout for discovered overlapping biclusters of gene expression. Their method would allow repeated rows and columns from the original matrix for display purposes. Their goal is to seek a layout which would result in the smallest size (in terms of area: the number of rows times the number of columns) of the visualization matrix. They provide a heuristic algorithm for this purpose. Our problem is clearly different since we do not allow repeated rows or columns in the matrix. We believe the redundant rows and columns would very likely make the relationship between different overlapping submatrices confusing.

1.2 Our contribution

In this paper, we make the following contributions:

1. We propose and formulate the submatrix pattern visualization problem as an optimization problem.

2. We discover an interesting link between the MOV problem and the *hypergraph vertex ordering* problem, which generalizes the traditional *minimal linear arrangement* (or *graph ordering* problem). We prove this problem is NP-hard. Based on our best knowledge, this is the first study of the hypergraph vertex ordering problem.

3. We present a novel iterative algorithm which utilizes existing graph ordering algorithm(s) to solve the optimal visualization problem. We prove our algorithm will always converge to a local minimum.

4. We perform a detailed experimental evaluation for our new algorithm.

2 NP-hardness and Minimal Linear Arrangement Problem

In this section, we will show our *Matrix Optimal Visualization Problem* is related to the *hypergraph ordering problem*, which is the generalization of the well-known *minimum linear arrangement problem* or MinLA. We will later reduce MinLA to our problem to prove its NP-hardness.

2.1 Hypergraph ordering and MinLA problem

Let $HG = (V, X)$ be a hypergraph, where $V = v_1, v_2, ..., v_n$ is the set of vertices and $X = x_1, x_2, ...x_m$ is the set of hyperedges. Given an order σ, the cost of a hyperedge $x \in X$ is defined as

$$hyperedge\_cost(x, \sigma) = (\max_{v_j \in x} \sigma(v_j) - \min_{v_k \in x} \sigma(v_k)),$$

and the cost of HG is defined as

$$\begin{aligned} hyper\_cost(HG, \sigma) &= \sum_{x \in X} hyperedge\_cost(x, \sigma) \\ &= \sum_{x \in X} (\max_{v_j \in x} \sigma(v_j) - \min_{v_k \in x} \sigma(v_k)) \end{aligned}$$

Thus, the cost of each hyperedge for a given order σ corresponds to the distance between the first vertex and the last vertex in the hyperedge. For example, in figure 2(a), the hypergraph contains four hyperedges $(0, 4)$, $(0, 2, 3, 4)$, $(1, 3, 5)$, $(2, 3, 6)$ shown as circles. Given an order $\sigma(0) \prec \sigma(1) \prec \sigma(2) \prec \sigma(3) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6)$, the hypergraph cost is $(4-0)+(4-0)+(5-1)+(6-2) = 16$. Thus, we define the *hypergraph ordering problem* as follows.

Definition 3 (Hypergraph Ordering Problem) *Given a hypergraph $HG = (V, X)$, the hypergraph ordering problem finds the order σ of V that can minimize $hyper\_cost(HG, \sigma)$:*

$$argmin_{\sigma} \sum_{x \in X} (\max_{v_j \in x} \sigma(v_j) - \min_{v_k \in x} \sigma(v_k))$$

As we will see, this problem is a generalization of the well-known MinLA problem.

Definition 4 (Minimum Linear Arrangement Problem [9]) *Given a graph $G = (V, E)$, the minimum linear arrangement problem finds the order σ of V that can minimize the $graph\_cost(G, \sigma)$:*

$$\sum_{(i,j) \in E} |\sigma(v_i) - \sigma(v_j)|$$

Note that this problem and the hypergraph ordering problem both can be extended to the weighted version. The minimum linear arrangement problem (or MinLA) was orignally formulated by Harper [9]. It has a lot of applications, including VLSI design [1], graph drawing [17], modeling of nervous activity in the cortex[15], single machine job scheduling[2][22], and etc.

It is well known that minimum linear arrangement problem is NP-hard [6]. There are polynomial time algorithms for computing exact solutions of MinLA for some special graph families. But for general graphs, as of this date the best algorithm with bounded results is an approximation algorithm with an $O(logn)$ approximation factor [21]. However, there are quite a few heuristic algorithms for the minimum linear arrangement problem. Among them [10; 3; 12; 19; 18; 20; 23] turn out to be very successful.

2.2 NP-hardness of Matrix Optimal Visualization Problem

We show that our matrix optimal visualization problem can be converted to a pair of hypergraph ordering problems in the following way: Given a database $DB = (\mathcal{T}, \mathcal{I})$ ($\mathcal{T}$ is the set of all transactions and $\mathcal{I}$ is the set of all items) with a set of hyperrectangles $P = \{H_1 = \{T_1 \times I_1\}, H_2 = \{T_2 \times I_2\}, \cdots, H_p = \{T_p \times I_p\}\}$, we create two hypergraphs in the following way:

Hypergraph $HG_1 = (V_1, X_1)$, where $V_1 = \mathcal{T}$ and $X_1 = \{T_1, T_2, ..., T_p\}$.

Hypergraph $HG_2 = (V_2, X_2)$, where $V_2 = \mathcal{I}$ and $X_2 = \{I_1, I_2, ..., I_p\}$.

Then we have

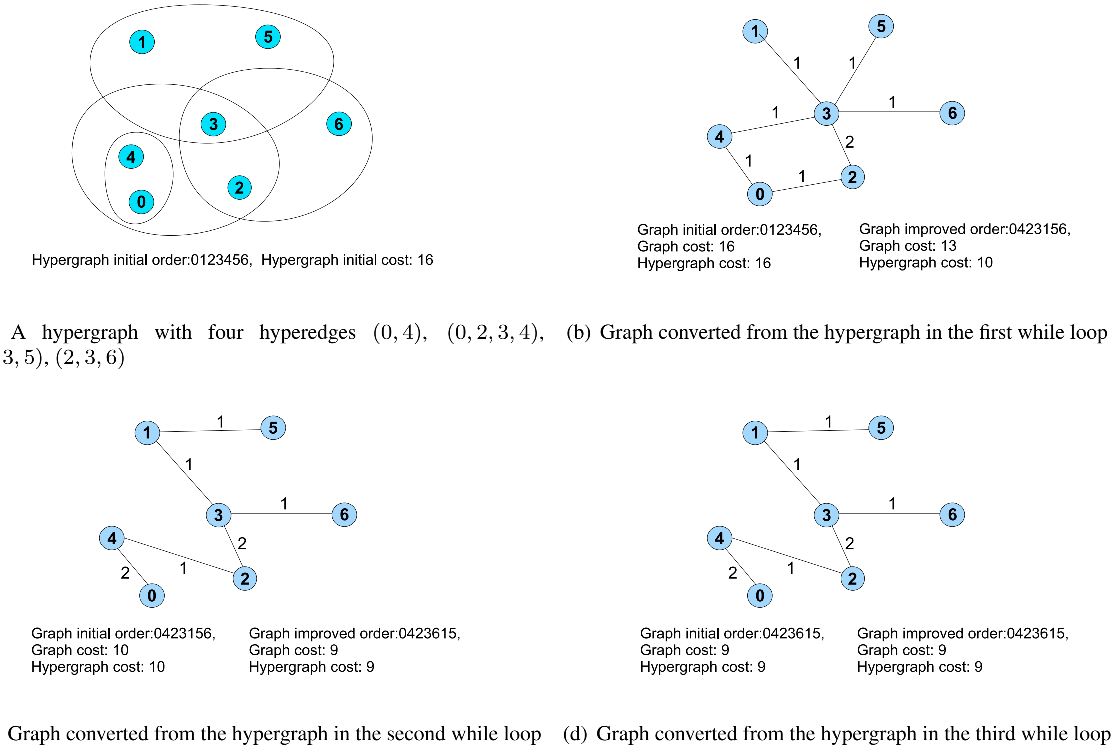
$$\begin{aligned} & visual\_cost(P, \sigma_T, \sigma_I) = \\ & \sum_{j=1}^{p} (\max_{t_u \in T_j} \sigma_T(t_u) - \min_{t_w \in T_j} \sigma_T(t_w)) \\ + \quad & \sum_{j=1}^{p} (\max_{i_u \in I_j} \sigma_I(i_u) - \min_{i_w \in I_j} \sigma_I(i_w)) \\ = \quad & \sum_{T_j \in X_1} (\max_{t_u \in T_j} \sigma_T(t_u) - \min_{t_w \in T_j} \sigma_T(t_w)) \\ + \quad & \sum_{I_j \in X_2} (\max_{i_u \in I_j} \sigma_I(i_u) - \min_{i_w \in I_j} \sigma_I(i_w)) \\ = \quad & hyper\_cost(HG_1, \sigma_T) + hyper\_cost(HG_2, \sigma_I) \end{aligned}$$

From the definitions we can see that $hyper\_cost(HG_1, \sigma_T)$ is not affected by σ_I and $hyper\_cost(HG_2, \sigma_I)$ is not affected by σ_T. Therefore, to solve our problem we can *convert it into two independent hypergraph ordering problems*.

Now we are interested in effectively solving the hypergraph ordering problem. But unfortunately, we find it is a NP-hard problem, as stated in the following lemma:

Lemma 1 *Given a hypergraph $HG = (V, X)$, it is NP-hard to find an order σ such that $hyper\_cost(HG, \sigma)$ is minimized.*

Proof: We can reduce the minimum linear arrangement problem, which is NP hard [6], to this problem. The minimum linear arrangement problem can be formulated as:


1
5
3
6
4
2
0
Hypergraph initial order:0123456, Hypergraph initial cost: 16
1
1
1
1
1
2
1
1
Graph initial order:0123456,
Graph cost: 16
Hypergraph cost: 16
Graph improved order:0423156,
Graph cost: 13
Hypergraph cost: 10
Graph initial order:0423156,
Graph cost: 10
Hypergraph cost: 10
Graph improved order:0423615,
Graph cost: 9
Hypergraph cost: 9
Graph initial order:0423615,
Graph cost: 9
Hypergraph cost: 9
Graph improved order:0423615,
Graph cost: 9
Hypergraph cost: 9

(a) A hypergraph with four hyperedges $(0,4)$, $(0,2,3,4)$, $(1,3,5)$, $(2,3,6)$

(b) Graph converted from the hypergraph in the first while loop

(c) Graph converted from the hypergraph in the second while loop

(d) Graph converted from the hypergraph in the third while loop

Figure 2. A running example of hypergraph ordering

Given a weighted graph $G = (V, E)$, what order σ can minimize $graph\_cost(G, \sigma) = |\sigma(v_i) - \sigma(v_j)|$? Note that this is basically a special case of the hypergraph ordering problem, where each hyperedge contains only two vertices. □

From the proof of lemma 1 one can conclude that the minimum linear arrangement problem is a special case of hypergraph ordering problem. There are many papers targeting the minimum linear arrangement problem, with both approximation results and heuristic results. However, applying these algorithms directly to hypergraphs do not work and it is not clear how these algorithms can be modified to work on the hypergraph.

3 Algorithms

In this section, we propose a novel scheme to tackle the hypergraph ordering problem. Note that this would directly provide a solution for the matrix optimal visualization problem based on the discussion in Subsection 2.2. Our basic idea is to build a bridge between the hypergraph ordering problem and the minimum linear arrangement problem. Our algorithm can employ any (approximated or heuristic) algorithm for the minimum linear arrangement problem to solve the hypergraph ordering problem.

At a high level, our algorithm works as follows. First, we transform a hypergraph $HG = (V, X)$ based on an order σ into a graph $G = (V, E)$ which has the same set of vertices as HG. Specifically, the graph G is a weighed undirected graph. Recall that given an order σ, the cost of a weighed graph G is defined as

$$graph\_cost(G, \sigma) = \sum w_{i,j} |\sigma(v_i) - \sigma(v_j)|$$

For example, in figure 2(b), given an order $\sigma(0) \prec \sigma(1) \prec \sigma(2) \prec \sigma(3) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6)$, the graph cost is $1*|4-0|+1*|2-0|+2*|3-2|+1*|4-3|+1*|6-3|+1*|5-3|+1*|3-1| = 16$. The cost of a subgraph of G, for example a path in G, can be defined the same way as the cost of G. Second, we apply a MinLA algorithm on G to find a good order σ'. Third, we convert hypergraph HG into another graph G' based on the new order σ'. Then we apply the MinLA algorithm on G' again and repeat these steps until we find a good enough order for HG, or there is no improvement.

Algorithm 1 formally describes this strategy. Note that our algorithm can start with any order σ. A detailed example is available in section 3.1.

Algorithm 1 Hyper_Ordering (Hypergraph HG, Order σ)

```
$old\_hyper\_cost \leftarrow MAXIMUM\_NUMBER$
$new\_hyper\_cost \leftarrow MINIMUM\_NUMBER$
$\sigma' \leftarrow \sigma$
while $old\_hyper\_cost > new\_hyper\_cost$ do
    $\sigma \leftarrow \sigma'$;
    $old\_hyper\_cost \leftarrow hypercost(HG, \sigma)$;{$hypercost$ returns the cost of hypergraph $HG$ with order $\sigma$.}
    Transform hypergraph $HG$ based on the order $\sigma$ into $G$;
    $\sigma' \leftarrow MinLA(G)$; {$graph\_cost(G, \sigma') \leq graph\_cost(G, \sigma)$}
    $new\_hyper\_cost \leftarrow hyper\_cost(HG, \sigma')$;
end while
```

Given this, we can see the critical question in the entire algorithm is what kind of graph the hypergraph should be converted to so that we can keep improving the cost of hypergraph ordering. In this paper we describe three kinds of interesting conversions which have such nice properties.

3.1 Converting hyperedge into path

To materialize step 7 of algorithm 1, our first strategy is to convert each hyperedge in $HG = (V, X)$ into a path in $G = (V, E)$. We will show by both analytical and empirical studies that this conversion is effective. The algorithm turns each hyperedge in HG into an unweighted path (*passing each vertex in the hyperedge exactly once in the order of* σ). Let's call it *hyperedge-path*. The weight of an edge in G is the number of hyperedge-paths it belongs to. This conversion implies that the cost of graph G is the sum of the cost of each hyperedge-path, i.e.

$$hyper\_cost(HG, \sigma) = \sum_{x \in X} graph\_cost(p_x, \sigma)$$

where p_x is a hyperedge-path corresponding to $x \in X$. Algorithm 2 formally describes the conversion.

Algorithm 2 Hyper_to_Graph_Path (Hypergraph HG, Order σ)

```
Create a graph $G$;{Let $w_{i,j}$ be the weight of the edge connecting vertex $i$ to vertex $j$. Initially for any $i, j$, $w_{i,j} == 0$, which means they are disconnected.}
for all $x \in X$ do
    sort vertices in $x$ according to their ranking in $\sigma$, i.e., $\sigma_{x_1} \prec \sigma_{x_2} \cdots \prec \sigma_{x_k}$;
    for $i = 1$ to $k - 1$ do
        $w_{x_i,x_{i+1}} \leftarrow w_{x_i,x_{i+1}} + 1$;
    end for
end for
```

A running example of Algorithm 2 which is employed by Algorithm 1 in step 7:

Figure 2(a) is a hypergraph with four hyperedges: $(0, 4), (0, 2, 3, 4), (1, 3, 5), (2, 3, 6)$. Initially the order is $\sigma(0) \prec \sigma(1) \prec \sigma(2) \prec \sigma(3) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6)$ and $hyper\_cost(HG, \sigma) = 16$. In the first while loop of algorithm 1, the hypergraph HG is converted to a graph by algorithm 2 as shown in figure 2(b). Then in step 8 of algorithm 1 we call some minimum linear arrangement algorithm and get an improved order $\sigma = \{0, 4, 2, 3, 1, 5, 6\}$. Thus $graph\_cost(G, \sigma) = 13$ and $hyper\_cost(HG, \sigma) = 10$.

Based on $\sigma(0) \prec \sigma(4) \prec \sigma(2) \prec \sigma(3) \prec \sigma(1) \prec \sigma(5) \prec \sigma(6)$, the hypergraph HG is converted to a graph in figure 2(c) in the second while loop of algorithm 1, and the steps in algorithm 1 are repeated. Figure 2(d) shows the graph generated in the third while loop and in that loop we get no improvement in reducing hypergraph cost. Thus algorithm 1 stops and the final order is $\sigma(0) \prec \sigma(4) \prec \sigma(2) \prec \sigma(3) \prec \sigma(6) \prec \sigma(1) \prec \sigma(5)$ and $hyper\_cost(HG, \sigma) = 9$.

To show the effectiveness of algorithm 1 combining with algorithm 2, we have theorem 1 showing that the cost of HG is always decreasing unless there is no improvement of cost reduction.

Theorem 1 *Assuming algorithm 1 calls algorithm 2 in step 7, then* $hyper\_cost(HG, \sigma') \leq hyper\_cost(HG, \sigma)$ *at the end of each while loop in algorithm 1.*

Proof: To prove this theorem, we will refer to lemma 2 and lemma 3, which are proved subsequently. First, by lemma 2 we have

$$hyper\_cost(HG, \sigma) = graph\_cost(G, \sigma) \quad (*)$$

Second, by calling a MinLA algorithm in step 8 of algorithm 1 [1], we have

$$graph\_cost(G, \sigma') \leq graph\_cost(G, \sigma) \quad (**)$$

Finally, Lemma 3 implies

$$hyper\_cost(HG, \sigma') \leq graph\_cost(G, \sigma') \quad (***)$$

Combining $(*)$, $(**)$ and $(***)$, we prove this theorem. □

Note that all of these steps are within one while loop. In the next iteration, a new graph G will be produced from the hypergraph HG using the newly generated order σ'. Thus, we can see the hypergraph order will keep improving in terms of the $hyper\_cost$ until no improvement is possible.

Lemma 2 *The graph* G *generated by algorithm 2 has the same cost as hypergraph* $HG = (V, X)$*, i.e.* $hyper\_cost(HG, \sigma) = graph\_cost(G, \sigma)$*.*

Proof: For a hyperedge $x \in X$ in HG where $|x| = k$ and x_i is the ith vertex in x, its corresponding hyperedge-path in G contains edge $x_i x_{i+1}$ where $1 \leq i \leq k - 1$ and $\sigma(x_i) \prec \sigma(x_{i+1})$. Therefore, we have

$$\begin{aligned}
& graph\_cost(G, \sigma) \\
= & \sum_{x \in X} \sum_{i=1 \ to \ k-1} (\sigma(x_{i+1}) - \sigma(x_i)) \\
= & \sum_{x \in X} (\sigma(x_k) - \sigma(x_1)) \\
= & \sum_{x \in X} (\max_{v_j \in x} \sigma(v_j) - \min_{v_k \in x} \sigma(v_k)) \\
= & hyper\_cost(HG, \sigma)
\end{aligned}$$

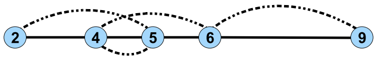
□

Lemma 3 *Given an order* σ*, the cost of a hyperedge* $x \in X$ *of hypergraph* $HG = (V, X)$ *is no more than the cost of a hamiltonian path* p *with the vertex set* x*, i.e.* $hyperedge\_cost(x, \sigma) \leq graph\_cost(p, \sigma)$*.*

[1] The MinLA algorithm can always produce a order which is better than or equal to the default order, and therefore, we apply σ as the default order

Proof: Given a hamiltonian path p with vertex set V_p and order σ, we sort V_p according to the vertex order σ. Then we can build a set of intervals such that every two adjacent vertices in V_p defines an interval. We claim that the hamiltonian path p' made up of these intervals is the minimum-cost hamiltonian path for V_p and σ. This is because any such interval must be covered by an edge in p, otherwise p is disconnected, a contradiction. Therefore, the cost of p' is no more than p. Considering p is an arbitrary hamiltonian path, we prove p' is the minimum-cost hamiltonian path for V_p and σ. From the definition of a hyperedge cost, one can conclude the cost of a hyperedge x with σ is exactly the cost of its corresponding hyperedge-path, which has the same cost as the minimum-cost hamiltonian path with vertex set x and order σ. Thus the lemma is proved. □

For example, in Figure 3 let's assume $\sigma(2) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6) \prec \sigma(9)$. Then the hyperedge-path (or minimum-cost hamiltonian path) is $2-4-5-6-9$, which has cost 7. For an arbitrary hamiltonian path $2-5-4-6-9$, its cost is 9, greater than 7.



Hyperedge path: 2-4-5-6-9, cost: 7
An arbitrary hamiltonian path: 2-5-4-6-9, cost: 9

Figure 3. Hyperedge-path and an arbitrary hamiltonian path, where $\sigma(2) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6) \prec \sigma(9)$.

3.2 Converting hyperedge into cycle or multi cycles

In this subsection, we will go beyond the path conversion for a hyperedge. We will show another type of transformation: converting the hypergraph into a cycle or multicycle.

Converting a hyperedge into a cycle is similar to converting a hyperedge into a path. Algorithm 3 formally describes this conversion. It turns each hyperedge in HG into an unweighted cycle by passing each vertex in the hyperedge exactly once in the order of σ and return to the starting vertex. Let's name it *hyperedge-cycle*. The weight of an edge in G is the number of hyperedge-cycles it belongs to. This conversion implies that the cost of graph G is the sum of the cost of each hyperedge-cycle, i.e. $hyper\_cost(HG, \sigma) = \sum_{x \in X} graph\_cost(c_x, \sigma)$ where c_x is a hyperedge-cycle corresponding to $x \in X$.

Similar theorem and lemmas hold for hyperedge to cycle conversion and we list them below. Proofs are similar and due to space constraints, we omitted them in this paper.

Theorem 2 *Assuming algorithm 1 calls algorithm 3 in step 7, then $hyper\_cost(HG, \sigma') \leq hyper\_cost(HG, \sigma)$ at the end of each while loop in algorithm 1.*

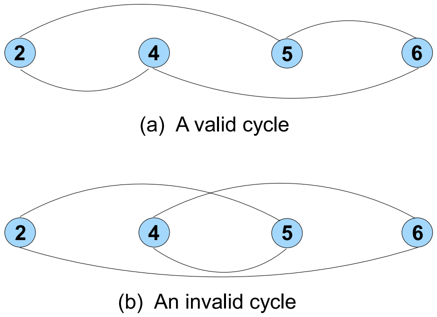
Lemma 4 *The graph G generated by algorithm 3 has twice the same cost as hypergraph HG, i.e. $hyper\_cost(HG, \sigma) = \frac{1}{2} \times graph\_cost(G, \sigma)$.*

Algorithm 3 Hyper_to_Graph_Cycle (Hypergraph HG, Order σ)

```
Create a graph $G$;{Let $w_{i,j}$ be the weight of the edge connecting vertex $i$ to vertex $j$. Initially for any $i, j$, $w_{i,j} == 0$.}
for all $x \in X$ do
    sort vertices in $x$ according to their ranking in $\sigma$, i.e., $\sigma_{x_1} \prec \sigma_{x_2} \cdots \prec \sigma_{x_k}$;
    for $i = 1$ to $k - 1$ do
        $w_{x_i,x_{i+1}} \leftarrow w_{x_i,x_{i+1}} + 1$;
    end for
    $w_{x_1,x_k} \leftarrow w_{x_1,x_k} + 1$;
end for
```

Lemma 5 *Given an order σ, the cost of an hyperedge x of hypergraph $HG = (V, X)$ is no more than $\frac{1}{2}$ cost of a hamiltonian cycle c with the vertex set x, i.e. $hyperedge\_cost(x, \sigma) \leq \frac{1}{2} graph\_cost(c, \sigma)$.*

Although for a hyperedge x, its corresponding hyperedge-cycle c contains only one more edge than its corresponding hyperedge-path p, c embodies richer information than p. In section 4, we show that hyperedge to cycle conversion is practically more efficient than hyperedge to path conversion.



(a) A valid cycle

(b) An invalid cycle

Figure 4. $\sigma(2) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6)$. (a) is a valid cycle. (b) is not a valid cycle because interval $4-5$ is covered by four edges.

A natural extension of the hyperedge to cycle conversion is converting a hyperedge into multi cycles. We would convert each hyperedge in HG into d *valid cycles* in G. A valid cycle should satisfy two conditions: (1) It passes each vertex in the hyperedge exactly once and return to the starting vertex, i.e. it is a hamiltonian cycle. (2) Exactly two of its edges contain an interval $\sigma_i \sigma_{i+1}$. See figure 4 for a valid cycle and invalid cycle. From the previous analysis, one can conclude that we can generate a graph G from a hypergraph HG and an order σ such that $hyper\_cost(HG, \sigma) = \frac{1}{d} \times graph\_cost(G, \sigma)$ for some d. Similar theorem (as to theorem 1 or 2) and lemmas (as to lemma 2,3 or lemma 4,5) also hold for the hyperedge to multi cycles conversion.

Now we are interested in exploring the extreme case: What is the maximum number of valid cycles a hyperedge in HG can be converted to? How effective is this hyperedge to max valid cycles conversion in solving our problem?

To understand this hyperedge to max valid cycles conversion, let's first see a simple example: For a hyperedge $x = \{2, 4, 5, 6\}$ with order $\sigma(2) \prec \sigma(4) \prec \sigma(5) \prec \sigma(6)$, we can have maximally 4 valid cycles, as shown in figure 5.

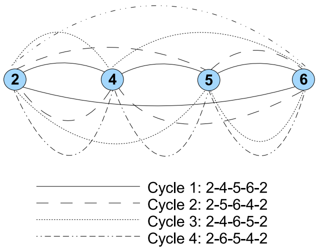


Figure 5. A hyperedge of 4 vertices has 4 valid cycles.

By combinatorial analysis, we have the following lemma for an arbitrary hyperedge with n vertices:

Lemma 6 *Given a hyperedge of k vertices with order σ, (1) the number of valid cycles containing edge $\sigma_i\sigma_j$ is $max(1, 2^{k-j-2}) \times max(1, 2^{i-1})$, and (2) the total number of valid cycles is 2^{k-2}.*

Therefore, we can use algorithm 4 to convert a hypergraph HG with σ into a graph G, by converting each hyperedge into a maximum number of valid cycles. In section 4, we will study empirically how effective this conversion is.

Algorithm 4 Hyper_to_Graph_Max_Multicycle (Hypergraph HG, Order σ)

```
Create a graph $G$;{Let $w_{i,j}$ be the weight of the edge connecting vertex $i$ to vertex $j$. Initially for any $i, j$, $w_{i,j} == 0$.}
for all $x \in X$ do
    sort vertices in $x$ according to their ranking in $\sigma$, i.e., $\sigma_{x_1} \prec \sigma_{x_2} \cdots \prec \sigma_{x_k}$;
    for $i \leftarrow 1$ to $k - 1$ do
        for $j \leftarrow i + 1$ to $k$ do
            $a \leftarrow max(1, 2^{k-j-2})$;
            $b \leftarrow max(1, 2^{i-1})$;
            $w_{i,j} \leftarrow ab$;
        end for
    end for
end for
```

3.3 Worst Case Time Complexities

In algorithm 1 either the hypergraph cost is reduced in each while loop or the while loop terminates. Therefore the maximum number of while loops are the initial cost of the hypergraph, which is no more than $|V||X|$. Algorithm 1 also calls two other functions. The first function converts the hypergraph into a graph. To do this conversion, algorithm 2 and algorithm 3 takes $O(|X||V|)$ time, and algorithm 4 takes $O(|X||V|^2)$ time. In addition, it takes $O(|X||V|d)$ time if a hyperedge is converted into d arbitrary valid cycles. The second function solves the minimum linear arrangement problem and its running time depends on what the minimum linear arrangement algorithm it is.

Datasets	$\mathcal{I}$	$\mathcal{T}$	Avg. Len.	$\lvert DB\rvert$	density
mushroom	119	8,124	23.0	186,852	dense
retail	16,470	88,162	10.3	908,576	sparse
kosarak	41,270	990,002	8.1	8,019,015	very sparse
BMS-WebView-1	497	59,602	2.5	149,639	very sparse
T40I10D100K	942	100,000	39.6	3,960,507	sparse

Table 1. dataset characteristics

Conclusively, the total running time of our hypergraph ordering algorithm (i.e. algorithm 1) is $O(|X|^2|V|^2) + O(|X||V|)O(MinLA)$ if we convert hyperedges into paths or cycles, or $O(|X|^2|V|^3) + O(|X||V|)O(MinLA)$ if we convert each hyperedge into maximum valid cycles, or $O(|X|^2|V|^2d) + O(|X||V|)O(MinLA)$ if we convert each hyperedge into d arbitrary valid cycles. $O(MinLA)$ is the time complexity of the minimum linear arrangement algorithm used by algorithm 1.

4 Experimental Results

In this section, we report our experimental evaluation on four real datasets and one synthetic dataset. All of them are publicly available from the FIMI repository [2]. The basic characteristics of the datasets are listed in Table 1. In our experiments, all hyperrectangles are generated by a submatrix pattern ranking algorithm [27], and we apply an state-of-the-art MinLA algorithm implemented by Safro et al. [23]. Its running time is linear to $|V| + |E|$, and thus, it can be applied to very large graphs. All algorithms were implemented in C++ and run on a 2.2 GHz Opteron with 2GB of memory.

In our experimental evaluation, we are interested in the following questions:

(1) How are the visualization effects of hyperrectangles on different datasets?

(2) How effective are the three different conversions, i.e. converting hyperedge into a path, a cycle, and maximum number of valid cycles?

(3) How good is the scalability of our algorithms?

To answer these questions, we performed a list of experiments, which we summarized as follows.

Group 1: We show and compare the visualization effects of the top 10 ranked hyperrectangles on five datasets by these three different conversions.

Group 2: We compare the $visual\_cost(P, \sigma_T, \sigma_I)$ (i.e. $hyper\_cost(HG_1, \sigma_T) + hyper\_cost(HG_2, \sigma_I)$ of three different conversions. They are the cost our algorithms try to minimize.

Group 3: We compare the running time of our algorithms on both small and large datasets, and on the same dataset with different number of hyperrectangles for the scalability study.

[2] http://fimi.cs.helsinki.fi/data/

4.1 Visualizing Hyperrectangles

Here, to visualize a large transactional data on a relatively small matrix, we apply the random sampling technique. Specifically, we sampled 250 transactions of each dataset, to bring the number of transactions more in line with the number of items. This makes two dimensional visualization succinct and also saves unnecessary computational cost.

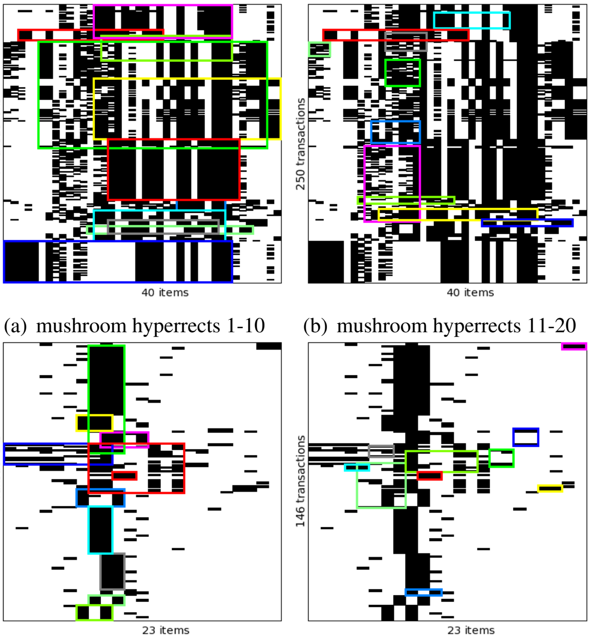
We visualize a transactional matrix in two dimensions as follows. If a transaction i contains item j, then the corresponding pixel (i, j) is black. We extract the top 10 hyperrectangles from each dataset, and visualize each hyperrectangle by drawing a minimum bounding rectangle - the smallest rectangle that covers all of its cells - around it. The denser (blacker) area a bounding rectangle has, the better the reordering is. In some cases the bounding rectangle is completely black, then it is equal to the corresponding hyperrectangle. Results are shown in figure 6.

For each sampled dataset, we display four figures. Figure (a) shows its appearance with original orders σ_T and σ_I. Figure (b) shows its appearance with updated orders by our proposed hypergraph ordering methods (using the hyperedge to cycle conversion) for the best visualization of these ten hyperrectangles. Figure (c) highlights the first five hyperrectangles by zooming in and drawing a colored rectangular boundary around each corresponding hyperrectangle. Figure (d) highlights the second five hyperrectangles in the same way as Figure (c) does. We can see these top 10 ranked hyperrectangles are actually relatively well-separated for the most of the cases. In return, this suggests there is little redundancy between these top ranked patterns. However, as the number of hyperrectangles of interests increase, we will see more overlapping among them. This is shown in Figure 7. Due to space constraints, we only provide two examples, each showing a reordering of the top 20 hyperrectangles for the mushroom and kosarak datasets.

4.2 Visual Cost Comparison

Since the goal of our algorithm is to minimize the visualization cost of a set of hyperrectangles, $visual\_cost(P, \sigma_T, \sigma_I)$, here we report $visual\_cost(P, \sigma_T, \sigma_I)$ of sampled datasets using three different conversions. The cost graphs in Figure 8 show the results. Here, we vary the number of hyperrectangles from 5, 10, to 20 for each conversion.

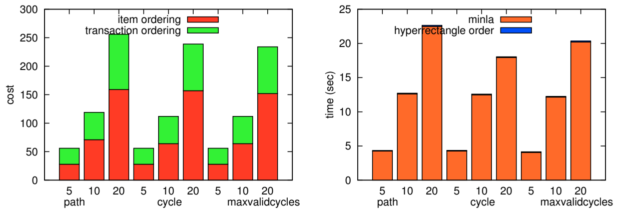
From these results we can see that algorithm 3 (hyperedge to cycle) and algorithm 4 (hyperedge to max valid cycles) are generally better than algorithm 2 (hyperedge to path). But interestingly, one can also observe that algorithm 3 is better than algorithm 4 in most cases (except for T40I10D100K, see Figure 9). As we know that converting hyperedge into one cycle and maximum valid cycles are just two extremes. We conjecture that there may exist a number k for a dataset such that hyperedge to k cycles conversion would yield the lowest visualization cost. How to choose k is an open question at this point.



(a) mushroom hyperrects 1-10 (b) mushroom hyperrects 11-20

(c) kosarak hyperrects 1-10 (d) kosarak hyperrects 11-20

Figure 7. experimental results reordering 20 hyperrectangles (hyperrectangles best viewed in color)

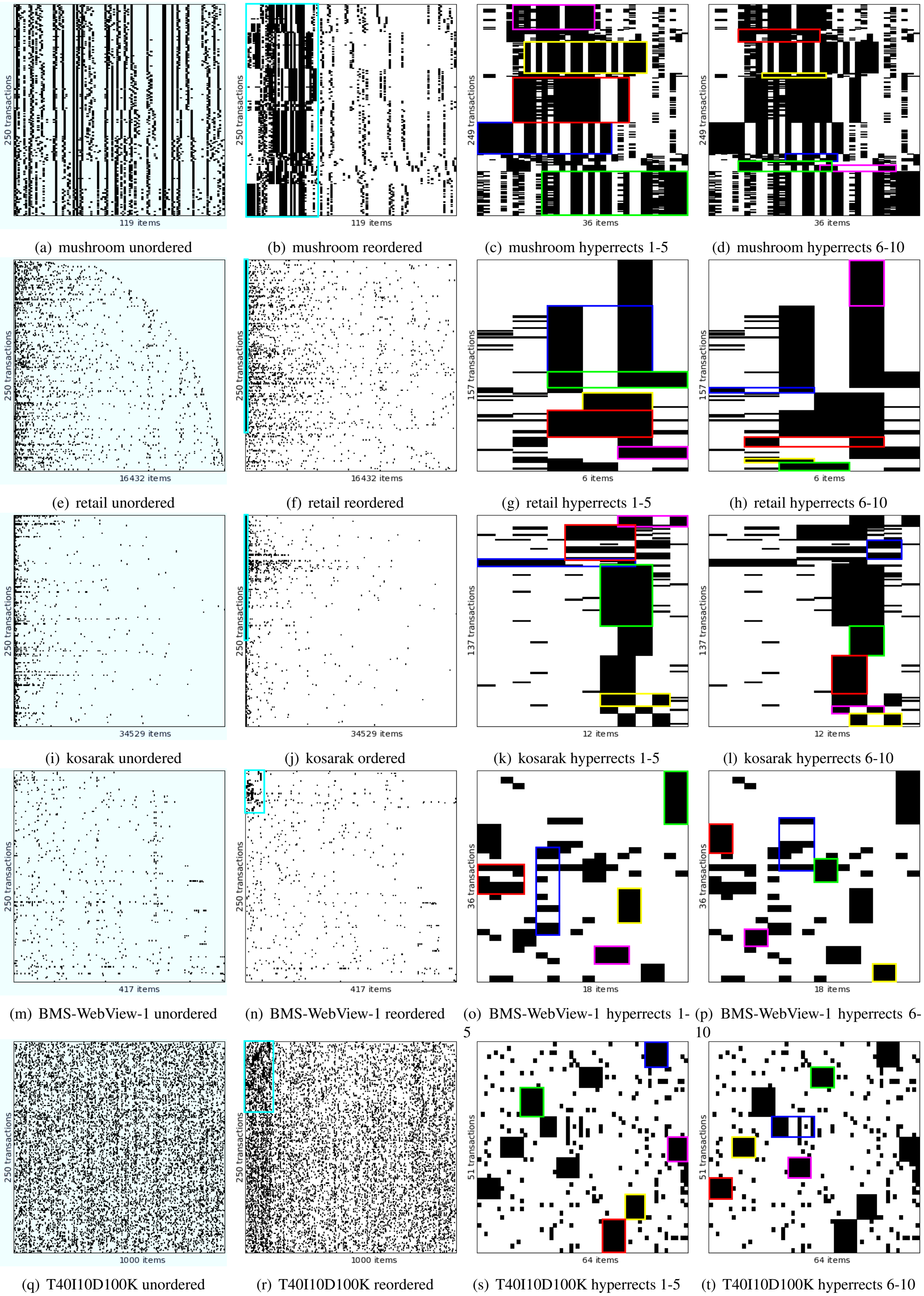


(a) T40I10D100K cost (b) T40I10D100K runtime

Figure 9. Cost and runtime (cont'd from Fig. 8)

4.3 Running time

From the analysis of section 3.3, our methods have a worst case polynomial time complexity. In experiments we find the while loop of the algorithms finishes in only a few rounds. Thus in practice our algorithms (not counting MinLA) finish in a reasonable time, approximately linear to $|V||X|$. The runtime graphs of Figure 8 show the running time related to three different conversions on different datasets. In our algorithms we call the MinLA algorithm many times to get an improved graph order. We distinguish the running time of our algorithm with the running time of MinLA (in multiple times) of [23] in figure 8. We can see that in most cases the running time of our algorithms is almost negligible in comparison with MinLA of [23] with only one exception, the kosarak dataset. We believe this is due to its unusually high number of unique items, which is much higher than other datasets. In addition, the running time of our algorithms remains almost constant when the number of hyperrectanges increases.



(a) mushroom unordered (b) mushroom reordered (c) mushroom hyperrects 1-5 (d) mushroom hyperrects 6-10

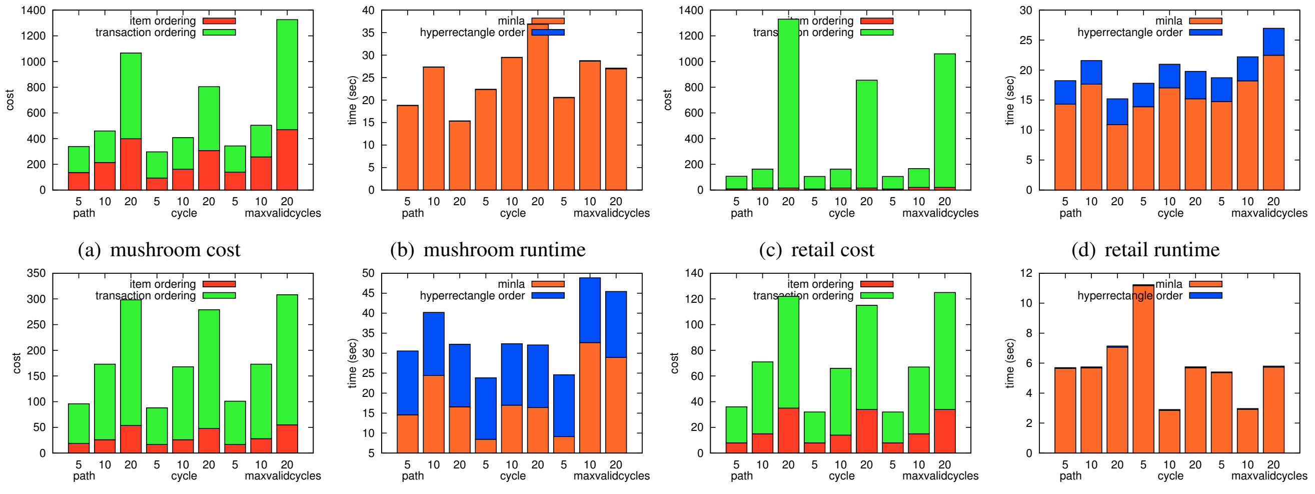
(e) retail unordered (f) retail reordered (g) retail hyperrects 1-5 (h) retail hyperrects 6-10

(i) kosarak unordered (j) kosarak ordered (k) kosarak hyperrects 1-5 (l) kosarak hyperrects 6-10

(m) BMS-WebView-1 unordered (n) BMS-WebView-1 reordered (o) BMS-WebView-1 hyperrects 1-5 (p) BMS-WebView-1 hyperrects 6-10

(q) T40I10D100K unordered (r) T40I10D100K reordered (s) T40I10D100K hyperrects 1-5 (t) T40I10D100K hyperrects 6-10

Figure 6. experimental results (hyperrectangles best viewed in color)



(a) mushroom cost (b) mushroom runtime (c) retail cost (d) retail runtime

(e) kosarak cost (f) kosarak runtime (g) BMS-WebView-1 cost (h) BMS-WebView-1 runtime

Figure 8. Cost and runtime by solution type (path / cycle / multipath) and num hyperrectangles (5 / 10 / 20)

5 Conclusions

In this work, we study a new variant of matrix visualization: given a set of submatrix patterns, how to reorder rows and columns so that a goodness function based on the enclosing rectangles for these submatrices can be minimized. We found an interesting link from this visualization problem to a well-known graph theoretical problem: the minimal linear arrangement (MinLA) problem. We have proposed an interesting algorithm framework to solve this problem. Empirically, we believe our algorithm can serve as a fundamental technique for many visual data mining tasks. For instance, our method can be incorporated into an interactive visualization environment to allow users to focus on different parts of the data and patterns. Theoretically, we introduce a generalization of the MinLA problem for the hypergraphs. There are many open problems of this generalization. For instance, how can we construct an approximate algorithm with nontrivial bound? Finally, many existing algorithms for the MinLA problem need to be revisited for the new generalized problem setting.

6 Acknowledgments

The authors would like to thank Ilya Safro for providing the Minimum Linear Arrangement source code and help on experimental setup.

References

[1] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[2] D. L. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[3] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *J. Graph Algorithms Appl.*, 5(4), 2001.

[4] C. Brunk, J. Kelly, and R. Kohavi. Mineset: An integrated system for data access, visual data mining, and analytical data mining,. In *Proceedings of the Third Conference on Knowledge Discovery and Data Mining (KDD-97)*, 1997.

[5] V. Chandola and V. Kumar. Summarization - compressing data into an informative representation. *Knowl. Inf. Syst.*, 12(3):355–378, 2007.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, New York, NY, USA, 1974. ACM.

[7] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981.

[8] G. Grothaus, A. Mufti, and T. Murali. Automatic layout and visualization of biclusters. 1:1748–7188, 2006.

[9] L. H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[10] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Appl. Math.*, 36(2):153–168, 1992.

[11] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 08(1):1–8, 2002.

[12] Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. In *WG '02: Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 296–309, London, UK, 2002. Springer-Verlag.

[13] C. Krumpelman and J. Ghosh. Matching and visualization of multiple overlapping clusterings of microarray data. In *IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology, CIBCB'07*, pages 121–126, 2007.

[14] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 1(1):24–45, 2004.

[15] G. Mitchison and R. Durbin. Optimal numberings of an n n array. *SIAM J. Algebraic Discrete Methods*, 7(4):571–582, 1986.

[16] C. Mueller, B. Martin, and A. Lumsdaine. A comparison of vertex ordering algorithms for large graph visualization. In *APVIS*, pages 141–148, 2007.

[17] J. Pach, F. Shahrokhi, and M. Szegedy. Applications of the crossing number. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 198–202, New York, NY, USA, 1994. ACM.

[18] J. Petit. Combining spectral sequencing and parallel simulated annealing for the minla problem. *Parallel Processing Letters*, 13(1):77–91, 2003.

[19] J. Petit. Experiments on the minimum linear arrangement problem. *ACM Journal of Experimental Algorithms*, 8, 2003.

[20] T. Poranen. A genetic hillclimbing algorithm for the optimal linear arrangement problem. *Fundam. Inf.*, 68(4):333–356, 2005.

[21] S. Rao and A. W. Richa. New approximation techniques for some ordering problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 211–218, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[22] R. Ravi, A. Agrawal, and P. N. Klein. Ordering problems approximated: Single-processor scheduling and interval graph completion. In *ICALP '91: Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 751–762, London, UK, 1991. Springer-Verlag.

[23] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *J. Algorithms*, 60(1):24–41, 2006.

[24] A. Strehl and J. Ghosh. Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS J. on Computing*, 15(2):208–230, 2003.

[25] E. Vityaev and B. Kovalerchuk. Inverse visualization in data mining, 2002.

[26] J. Wang and G. Karypis. On efficiently summarizing categorical databases. *Knowl. Inf. Syst.*, 9(1):19–37, 2006.

[27] Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *KDD*, pages 758–766, 2008.

[28] K. Zhao, B. Liu, T. M. Tirpak, and W. Xiao. A visual data mining framework for convenient identification of useful knowledge. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 530–537, 2005.