On the Deep Order-Preserving Submatrix Problem: A Best Effort Approach

Byron J. Gao, Obi L. Griffith, Martin Ester, Hui Xiong, *Senior Member*, *IEEE*, Qiang Zhao, and Steven J.M. Jones

Abstract—Order-preserving submatrix (OPSM) has been widely accepted as a biologically meaningful cluster model, capturing the general tendency of gene expression across a subset of experiments. In an OPSM, the expression levels of all genes induce the same linear ordering of the experiments. The OPSM problem is to discover those statistically significant OPSMs from a given data matrix. The problem is reducible to a special case of the sequential pattern mining problem, where a pattern and its supporting sequences uniquely specify an OPSM. Unfortunately, existing methods do not scale well to massive data sets containing thousands of experiments and hundreds of thousands of genes, which are common in today's gene expression analysis. In particular, *deep OPSMs*, corresponding to long patterns with few supporting sequences, incur explosive computational costs in their discovery and are completely pruned off by existing methods. However, it is of particular interest of biologists to determine small groups of genes that are tightly coregulated across many experiments, and some pathways or processes may require as few as two genes to act in concert. In this paper, we study the discovery of deep OPSMs from massive data sets. We propose a novel best effort mining framework `KiWi` that exploits two parameters k and w to bound the available computational resources and search a selected search space, and does what it can to find as many as possible deep OPSMs. Extensive biological and computational evaluations on real data sets demonstrate the validity and importance of the deep OPSM problem, and the efficiency and effectiveness of the `KiWi` mining framework.

Index Terms—Order-preserving submatrix, OPSM, deep OPSM, deep pattern, subspace clustering, pattern-based clustering, sequential pattern mining, scalability, best effort, gene expression analysis, negative correlation, data mining.

1 Introduction

As an unprecedented breakthrough in experimental molecular biology, DNA microarrays, next-generation sequencing, and similar technologies have enabled a wave of extraordinary genomewide investigations of gene expression, allowing simultaneous monitoring of activities of many genes over many experiments. The resulting expression data can be viewed as an $n \times m$ matrix with n genes (rows) and m experiments (columns), in which each entry represents the expression level of a given gene under a given experiment.

Clustering, as a major tool for gene expression analysis, is an unsupervised way of grouping objects based on their similarity in feature space to achieve internal cohesion and external isolation. Coexpression of genes in a cluster can be used to infer functional associations between genes and identify coregulation [27]. Traditional methods such as k-means [34] and hierarchical clustering [29] do not work well for high-dimensional gene expression data because most genes are tightly coregulated only under a certain subset of experiments. Thus, subspace clustering has gained increasing popularity in recent years [1], [3].

Pattern-based subspace clustering, where clustering is performed using pattern similarity rather than distance similarity, is particularly meaningful in gene expression analysis [31], [35]. This is due to the fact that coregulated genes are not necessarily expressed at the same (or even similar) absolute expression level. For example, a transcription factor may be able to perform its function at a very different concentration from its target genes.

Recently, *order-preserving submatrix* (OPSM) [8] has been introduced and accepted as a biologically meaningful cluster model. An OPSM, essentially a pattern-based subspace cluster, is a subset of rows and columns in a data matrix where all the rows induce the same linear ordering of the columns, as shown in Fig. 1. An OPSM cluster may arise when the expression levels of the coregulated genes rise and fall synchronously in response to a sequence of environment stimuli. The *OPSM problem* is to discover significant OPSMs from a given data matrix, which can play an essential role in inferring gene regulatory networks.

- *B.J. Gao is with the Department of Computer Science, Texas State University, 601 University Drive, San Marcos, TX 78666. E-mail: bgao@txstate.edu.*
- *O.L. Griffith is with the Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Mailstop 977-250, Berkeley, CA 94720. E-mail: olgriffith@lbl.gov.*
- *M. Ester is with the School of Computer Science, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada. E-mail: ester@cs.sfu.ca.*
- *H. Xiong is with the Rutgers, The State University of New Jersey, 1 Washington Street (Room 1076), Newark, NJ 07102. E-mail: hxiong@rutgers.edu.*
- *Q. Zhao is with the Department of Mathematics, Texas State University, 601 University Drive, San Marcos, TX 78666. E-mail: qiang.zhao@txstate.edu.*
- *S.J.M. Jones is with the Genome Sciences Centre, British Columbia Cancer Agency, Vancouver, BC Canada V5Z 4S6. E-mail: sjones@bcgsc.ca.*

Manuscript received 18 Apr. 2010; revised 22 June 2010; accepted 18 July 2010; published online 17 Nov. 2010.
Recommended for acceptance by A. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-04-0222.
Digital Object Identifier no. 10.1109/TKDE.2010.244.

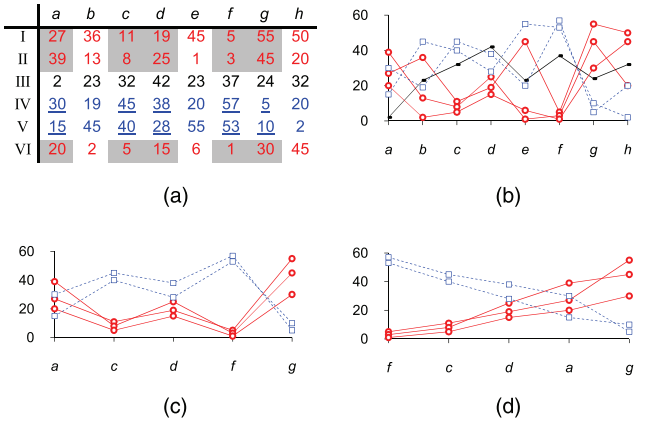


Fig. 1. OPSM and GOPSM. The raw gene expression data matrix in (a) Exhibits no obvious pattern when plotted in (b). (c) Shows an GOPSM consisting of two OPSMs corresponding to the submatrixes with shaded entries and underlined entries in the data matrix respectively. (d) Shows a permutation of columns of the GOPSM, under which the row sequences are in either strictly ascending or descending order. (a) Data matrix. (b) Data matrix plotted. (c) GOPSM consisting of two OPSM. (d) GOPSM rearranged.

The OPSM cluster model focuses on the relative order of columns rather than the uniformity of actual values in data matrixes [35]. By sorting the row vectors and replacing the entries with their corresponding column labels, the data matrix can be transformed into a sequence database, and OPSM mining is reduced to a special case of the sequential pattern mining problem with some distinctive properties. In particular, the sequence database is extremely dense since each column label appears exactly once (assuming no missing values) in each sequence. A sequential pattern uniquely specifies an OPSM with all the supporting sequences as the rows. The number of supporting sequences is the *support count*, or simply *support*, for the pattern.

As costs of gene expression analysis continue to decrease, the numbers and sizes of expression data sets have been growing at an ever-increasing rate. To give just two examples, the Gene Expression Omnibus (GEO) currently contains over 10,000 experiments comprising 400,000 samples and 16 billion individual abundance measurements, for over 500 organisms [7] and the Stanford Microarray Database (SMD) contains approximately 20,000 public experiments for over 20 organisms [26]. Expression data sets can easily be obtained with hundreds, thousands, and soon tens of thousands of experiments. The number of rows is determined by the number of trascripts/genes being investigated. Generally, the approach has been to include as many genes as possible. As array designs continue to improve, it has become possible to include probes for essentially all known genes. While the number of known genes has stabilized around 30,000, the continual discovery of new alternative splice forms indicates that 70 percent or more of these may be alternatively spliced into two or more unique transcripts [20], [38]. Thus, estimates of the number of different proteins encoded by the 30,000 human genes now exceed 100,000. Sequencing-based experiments and whole-genome tiling arrays that measure noncoding transcription can identify even greater numbers of expressed sequences for which we might wish to identify coexpression. Thus, with expression data sets of potentially tens or hundreds of thousands of both rows and columns, there is a need for algorithms that can handle not only "large" but "massive" data sets.

Most existing sequential pattern mining methods, breadth-first or depth-first, are *exact* in the sense that they find a complete set of frequent patterns. A pattern is *frequent* if its support is beyond a predetermined minimum support threshold, min_sup. Such methods rely on min_sup to prune the search space. A low setting of min_sup incurs explosive computational (combinatorial explosion of the number of candidate patterns) cost, making efficient sequential pattern mining infeasible for massive data sets. The situation is aggravated in OPSM mining due to the fact that the transformed sequence database is extremely dense. Note that an infrequent pattern, even with the lowest possible support of 2, can still be statistically significant if it is long. We use *deep patterns* to refer to such patterns that are long (thus significant) and with support values below reasonable min_sup settings (thus cannot be efficiently mined by exact methods). In OPSM mining, a deep pattern corresponds to a *deep OPSM* that appears wide and short.

Deep OPSMs are of particular interest to biologists as they may represent small groups of genes that are tightly coregulated under many conditions. Previous studies have tended to focus on larger clusters by design or necessity. While these large clusters have been shown to be of interest or value, there is no reason to expect that all or even most biological processes or disease mechanisms would involve tight coregulation of large groups of genes. In fact, protein-protein interactions, biological pathway membership, and coregulation demonstrate "power law" relationships [5]. That is to say, we tend to observe a small number of very large gene groups and a large number of very small gene groups. In many cases, biologists are looking for tissue- or experiment-specific gene regulation related to some highly specialized process or disease. They expect that many important processes will involve relatively small numbers of genes. Some pathways or processes might require only two genes to act in concert. The meaningfulness of deep OPSMs is also clearly validated by our own biological assessments.

Thus, the *deep OPSM problem*, discovering deep OPSMs from massive data sets, is of essential biological significance. While exact methods fail, we propose a novel best effort mining framework `KiWi` that does what it can to find as many as possible deep OPSMs for a given amount of available computational resources. `KiWi` exploits two parameters k and w to bound the computational resources and search a selected search space that are likely to contain deep OPSMs. It performs a beam search, at each level maintaining only the k most promising patterns as seeds for extension in the next level. In extending a pattern, `KiWi` looks for the new element in a vertical slice of width w (instead of the entire suffixes) of the supporting sequences. This *biased sampling* approach favors items that come early. It is not only essential for efficiency, but also instrumental to the effective discovery of deep OPSMs, ensuring that the k seats are reserved for seeds that are likely to be extended into longer patterns. The principles behind `KiWi` are applicable to sequential pattern mining in general.

We also generalize OPSM by including genes whose expression levels induce the opposite linear ordering of the experiments. The so-called *generalized OPSM* (GOPSM) model is able to capture negative correlations among genes,

which can imply common process/pathway membership or negative regulation [13]. KiWi is capable of mining GOPSMs.

In addition, we provide substantial technical extensions to KiWi for improved efficiency, effectiveness, and usability. Notably, we estimate the statistical significance of the discovered OPSMs so that they can be totally ordered. This line of study is of fundamental importance to not only OPSM mining, but pattern mining in general. We make initial contributions to this respect.

Contributions.

1. We identify and validate the deep OPSM problem that is of essential biological significance.
2. We propose a novel best effort mining framework KiWi for the deep OPSM problem. The KiWi principles are applicable to mining deep sequential patterns from dense, massive data sets in general.
3. We generalize OPSM and introduce GOPSM, capturing negative correlations among genes as well.
4. We provide substantial technical extensions to KiWi for improved efficiency, effectiveness, and usability, including estimation of statistical significance of OPSMs.
5. We present extensive biological and computational experiments on real data sets, validating the deep OPSM problem and the KiWi mining framework.

Outlines. The rest of the paper is organized as follows: Section 2 introduces the deep OPSM problem. Section 3 reviews related work. Section 4 presents the KiWi mining framework. Section 5 discusses the technical extensions. Section 6 reports the experimental results. Section 7 concludes the paper.

2 The Deep OPSM Problem

In this section, we introduce the background of OPSM mining and identify the deep OPSM problem.

2.1 OPSM, GOPSM, and Sequential Pattern Mining

Gene expression data can be represented as an $n \times m$ matrix D containing expression levels for n genes (rows) under m experiments (columns). A submatrix (R, C), where R is a subset of rows and C a subset of columns in D, is called an *order-preserving submatrix* [8] if there is a permutation of the columns in C under which the sequence of expression values in each row of R is strictly ascending.

Now we generalize OPSM and further define the so-called *generalized order-preserving submatrix* (GOPSM). If the above constraint is relaxed such that the sequence of expression values can be either strictly ascending or descending, (R, C) becomes a GOPSM. GOPSMs are able to capture anticorrelations among genes, which can imply common process/pathway membership or negative regulation [13]. One gene may repress the expression of other genes (negative regulators) and anticorrelated genes may represent members of opposing pathways [43]. The concepts of OPSM and GOPSM are illustrated in Fig. 1.

The OPSM problem [8] is to discover those statistically significant OPSMs from the data matrix D. The problem is closely related to sequential pattern mining. Conventional sequential pattern mining [4] was motivated and introduced in the context of transaction databases, where a sequence is an ordered list of itemsets. A common subsequence with *support* (number of sequences containing the subsequence) beyond a minimum support threshold, min_sup, is called a frequent sequential pattern. The sequential pattern mining problem is to find the complete set of frequent sequential patterns with respect to min_sup.

OPSM mining can be reduced to a special case of the sequential pattern mining problem. If we sort each row in the data matrix D in ascending order, and replace the entries with their corresponding column labels, then D is transformed into a sequence database, as shown in Fig. 3b. Each sequential pattern uniquely specifies an OPSM, where the pattern specifies the columns and the supporting sequences specify the rows.

Comparing to conventional sequential pattern mining, OPSM mining bears some special properties. First, each sequence in the transformed sequence database is an ordered list of 1-item itemsets. Second, each column label (item) in the alphabet appears at most once in a sequence, or exactly once in the absence of missing values. Third, since missing values are rare in expression data, sequences can be long, most containing all items (experiments) in the alphabet. These properties imply that the sequence database in OPSM mining is extremely dense and of high dimensionality.

In sequential pattern mining, mining efficiency is very sensitive to min_sup [40], [52]. A slightly lower min_sup will bring in significantly more new frequent patterns. The dense and high-dimensional nature of OPSM mining aggravates the case. Thus, exact methods do not scale well for massive data sets with low min_sup settings.

2.2 The Deep OPSM Problem

We are interested in statistically significant OPSMs. While various significance measures can be defined, it is consensus that for fixed # cols (number of columns), larger # rows (number of rows) lead to more significance, and for fixed # rows, larger # cols lead to more significance. To compare OPSMs with different # rows and # cols, we can use their p-values assuming the null hypothesis of random matrix.

In statistical hypothesis testing, the *probability value*, p-value for short, is the probability of obtaining a value of the test statistic at least as extreme as the one observed by chance alone, assuming that the null hypothesis is true. The *critical value* is a threshold to which the value of the test statistic is compared to determine whether or not the null hypothesis is rejected. It depends on a selected significance level. Let $p(R, C)$ denote the p-value for OPSM (R, C). The significance level at which we reject the null hypothesis can be set at, e.g., 0.001. In that case, (R, C) is significant if $p(R, C) < 0.001$, meaning there is less than 0.1 percent probability that (R, C) occurs at random.

In Fig. 2a, the curve represents the critical values w.r.t. a given significance level for different # cols values. The region above the curve is the *significant region*. An OPSM (R, C) is *significant* if it falls in this region. Since in an OPSM, the order of columns matters whereas the order of rows does not, its significance is much more sensitive to # cols than # rows.

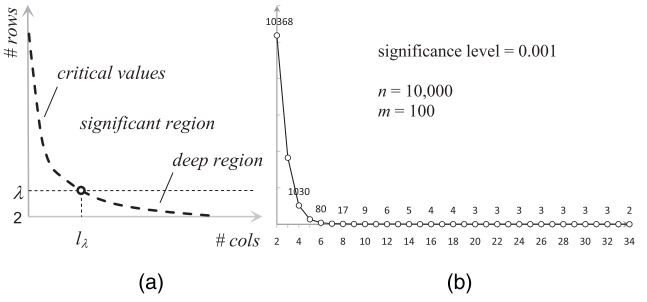


Fig. 2. Significance of OPSM. (a) Deep region. (b) Critical values.

Example 1. Fig. 2b shows estimated critical values for # cols $= 2, \ldots, 34$ w.r.t. significance level of 0.001 for $n =$ 20,000 and $m = 100$. Our experiments show that for long patterns, e.g., # cols ≥ 6, these values do not vary significantly w.r.t. n and m. Note that when the pattern is sufficiently long (e.g., 10 or longer), even two to five genes can form significant OPSMs.

In Fig. 2a, λ represents an imaginary lower bound of min_sup, for which the region above λ can be mined within a reasonable amount of time using exact methods and the region below it cannot. The *deep region* refers to part of the significant region that is below λ. An OPSM is *deep* if it falls in this region.

Definition 1 (Deep OPSM). *Deep OPSMs are OPSMs with many columns and few rows that are statistically significant but cannot be efficiently discovered using exact methods.*

The *deep OPSM problem* is to efficiently mine deep OPSMs from massive data sets. It can be reduced to a sequential pattern mining problem that mines long patterns with few supporting sequences.

Many significant OPSMs are deep. In Fig. 2a, the λ line crosses the significance border curve at # cols of l_λ, which means if the pattern length goes beyond l_λ, there will occur deep OPSMs. With the increase of the pattern length, most significant OPSMs will be deep. In other words, exact methods will miss most of the wide significant OPSMs that correspond to long patterns.

Example 2. In SPADE [52] and PrefixSpan [40] experiments, the lowest min_sup values are set to 0.25 percent and 0.2 percent of the total number of rows, corresponding to support of 50 and 40, respectively, for $n =$ 20,000. It can be said that for a data set with 20,000 genes, these methods cannot find OPSMs with less than 40 genes efficiently.

λ may move down with the growth of computational power, or move up due to the ever-increasing complexity of data. In this paper, we take λ as 0.2 percent of the total number of rows. Then in Fig. 2b, roughly $l_\lambda = 7$, where the critical value is 31 (not shown in figure). Note that this estimation is very conservative (lower than it should be). A practical λ can be much higher (see Example 3), as in OPSM mining; the sequence database is extremely dense. This means *in real cases, a large number of significant OPSMs are deep*.

Example 3. In our experiment, PrefixSpan, one of the fastest sequential mining algorithms, did not return after 48 hours on real data set human-SMD-cDNA (12,671 genes and 2,852 experiments) [26] for $min\_sup = 2,535$. It returned after 36 hours on GPL96 (12,332 genes and 1,640 experiments) [7] for $min\_sup = 1,234$ with the maximum pattern length of 7, while the longest pattern with support ≥ 2 has a length of 161. Such a long pattern corresponds to an extremely significant OPSM.

In summary, deep OPSMs are statistically significant. They are also biologically significant as we argued in Section 1 and will show in the experimental evaluation. In practice, a large number of desirable significant OPSMs are deep and missed out by existing exact methods. Thus, the deep OPSM problem is of critical importance.

3 Related Work

Clustering is the process of dividing data objects into subsets, i.e., clusters, so that objects in the same cluster are as similar as possible, and objects in different clusters are as dissimilar as possible. Traditional clustering methods, e.g., k-means [34], Agnes [29], CLARANS [39], BIRCH [54], and DBSCAN [15], work on full feature spaces. They do not perform well if the feature space is high dimensional.

Based on the observation that data objects may form clusters in a subset of features, subspace clustering has been introduced [3] and received much attention due to its wide applications as well as technical complexity. Subspace clustering can be described as finding a set of pairs (X, Y), where X is a subset of data objects and Y is a subset of features, such that the objects in X are close when projected onto Y, but not close when projected onto the remaining features [30].

Various subspace clustering algorithms have been proposed in recent years, including CLIQUE [3], ENCLUS [9], MAFIA [19], PROCLUS [1], ORCLUS [2], DOC [42], FPC [51], FINDIT [48], HARP [50], EWKM [28], and COSA [16]. In particular, CLIQUE [3] is a density- and grid-based bottom-up method that uses the Apriori principle to mine dense cells beyond a density threshold in subspaces.

Pattern-based clustering aims at finding patterns formed by subsets of objects in subsets of features. Each pattern corresponds to a subspace cluster. Thus, pattern-based clustering is indeed subspace clustering. However, it is a different variant because it measures similarity between objects based on the patterns they exhibit over the subsets of features, instead of the euclidean distance between them in the subspace as in normal subspace clustering.

Most pattern-based clustering methods treat data features and data objects as interchangeable concepts. Thus, they are also referred to as biclustering, coclustering, two-mode clustering, or two-way clustering algorithms. Biclustering algorithms are surveyed in [35]. Pattern-based clustering has proved to be particularly meaningful in gene expression analysis. Most proposed methods have utilized such applications for their validation.

Preceded by direct clustering [23] and block clustering [37], the δ-bicluster model [9] was introduced for gene expression analysis that discovers local coherence of genes and experiments in a submatrix of a DNA array. In the biological sense, genes in such clusters have the same

amount of response to the experiments. They assess the quality of a bicluster (I, J), i.e., a submatrix, by *mean squared residue* defined as

$$H(I,J) = \frac{1}{|I| \cdot |J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2.$$

Here, (I, J) qualifies as a δ-bicluster if $H(I, J) \leq \delta$.

Based on the δ-bicluster model, Yang et al. [49] introduced the δ-cluster model by allowing missing values. They also proposed an improved algorithm FLOC, which is a randomized move-based algorithm efficiently approximating k δ-clusters. The same authors further introduced the δ-pCluster [47] model and proposed a deterministic approach. The δ-pCluster model captures clusters that exhibit shifting or scaling patterns. Scaling pattern can be transformed into shifting pattern by applying a logarithmic function on the raw data. Under shifting pattern, the expression levels of all genes in a cluster rise and fall coherently under a subset of experiments.

The above pattern-based cluster models capture biclusters with coherent values. They can be restrictive in many applications. As a relaxation, Ben-Dor et al. [8] proposed the *order-preserving submatrix* model that captures biclusters with coherent evolutions. In an OPSM, the expression levels of all genes induce the same linear ordering of the experiments. The OPSM discovery problem is NP-hard; a greedy bottom-up algorithm was given in [8] to find the "best" OPSM that has the largest statistical significance, i.e., having the smallest prior probability. The algorithm starts with small OPSMs and iteratively extend the best l of them. The algorithm favors OPSMs with large support values [30]. It requires excessive computational resources and is not scalable to large gene expression matrixes, especially when used in finding OPSMs with small support values, i.e., deep OPSMs.

The OP-cluster (order-preserving cluster) model [32], [33] generalizes the OPSM model by grouping attributes into equivalent classes. Their proposed algorithm, OPC-tree, reports all (maximal) submatrices with # rows and # cols beyond given thresholds. The algorithm first groups together similar columns and then creates a nondecreasing order of columns for each row. The resulting set of column sequences are mined to obtain a complete set of frequent sequential patterns. As in conventional sequential pattern mining, the algorithm is exact and fails for the same reason to discover deep OPSMs from massive data sets.

Cheung et al. [10] showed that OPSM can be generalized to cover most existing pattern-based cluster models and proposed a number of extensions. Prelic et al. [41] assessed existing subspace cluster models and showed that OPSM had the highest proportion of clusters with significant enrichment of one or more Gene Ontology (GO) categories. They also showed that OPSM had good correspondence with known pathways according to their analysis of model-system protein-protein interaction networks. Zhao et al. [55] proposed a maximal subspace clustering algorithm that mines both positive and negative regulated gene clusters. A survey on subspace clustering, pattern-based clustering, and correlation clustering can be found in [31].

The OPSM mining problem can be reduced to a special case of the sequential pattern mining problem that was introduced in [4]. Existing methods are exact in the sense that they find a complete set of frequent patterns. These methods can be categorized as either breadth-first as in GSP [45] and SPADE [52], or depth-first as in PrefixSpan [40]. In particular, GSP uses the downward-closure property of sequential patterns and adopts a multiple-pass, candidate generate-and-test approach. SPADE uses the same Apriori-like approach, but takes advantage of vertical data format and ID lists to reduce scans of the sequence database. PrefixSpan uses pattern growth, a frequent-pattern mining paradigm that does not require candidate generation. Prefixspan performs similarily to GSP for $min\_sup > 1.5\%$ but much more efficient for low min_sup settings. It is generally considered one of the most efficient algorithms.

A preliminary version of this paper appears in [17]. The main additions include definition of the deep OPSM problem in a statistically rigorous manner (Section 2) and biological justification on the meaningfulness of deep OPSMs (Section 6). Theoretical analysis for observations used in `KiWi` is also added (Appendix A). Technically, substantial extensions are added for improved efficiency, effectiveness, and usability (Sections 4 and 5), leading to the release of `KiWi` 1.0, a comprehensive deep OPSM mining package. These extensions include parameter tuning, backward mining, pattern extension, and estimation of statistical significance of OPSMs. In particular, our study of statistical significance of OPSMs makes initial efforts to the fundamental problem of estimating statistical significance of sequential patterns.

4 The KiWi Mining Framework

Exact methods fail to discover deep OPSMs. We propose a best effort mining framework `KiWi` that does what it can to find as many as possible deep OPSMs for available computational resources. The key idea is to utilize two parameters k and w to bound the number of candidates and perform a biased sampling in candidate testing. By doing so, `KiWi` substantially reduces the search space to a selected part, targeting highly promising subpatterns that lead to long patterns. Such long patterns form deep OPSMs.

4.1 Principles

`KiWi` performs level-wise search from short to long patterns, growing one item at each level. However, it does not perform typical candidate generation and pruning as in other level-wise search methods. First, in `KiWi` min_sup is set to the lowest possible value of 2; there is no practical benefit in pruning. Second, the Apriori property that such pruning relies on does not hold in `KiWi`. `KiWi` takes a best effort approach making use of two observations.

Observation 1. A frequent subpattern will likely grow into a frequent superpattern.

Based on Observation 1, at each level i of the level-wise search, only the top k patterns (in terms of support count) are kept and used to grow the level $i + 1$ patterns. By doing so, mining time is spent on those promising subpatterns only, leading to significant efficiency gain. This breaks down the Apriori property, which does not help anyway due to the setting of $min\_sup = 2$.

Observation 2. A long pattern segments its supporting sequences into small sections.

Appendix A has theoretical analysis justifying the observation by deriving the cumulative distribution function (cdf) of the maximum length of sections segmented by a pattern. An R program was also written to compute the cdf for a given pattern length.

Based on Observation 2, the adjacent elements of a long pattern should not appear far away from each other in a supporting sequence. Thus, in counting the test statistic (e.g., support) of a candidate pattern, KiWi utilizes a biased sampling technique. It samples (on average) w of the m items. The sampling is biased because the w items are the ones in the sequences immediately following the pattern from the previous level. For example, in counting the support of $[a, b, c, d, e]$, KiWi examines if e appears in the w items immediately following d in the sequences supporting $[a, b, c, d]$.

The biased sampling technique gains in both efficiency and effectiveness. For efficiency, it reduces the time complexity of KiWi from relying on m to relying on w, an adjustable constant. For effectiveness, should we consider the entire suffixes of the sequences, some unpromising candidates would gain high ranks to take many of the k seats that are reserved for promising patterns, and the iteration would soon terminate and fail to grow long patterns.

Best effort approach. This is a *best effort* approach. Users can use the two parameters k and w to bound the computational resources, i.e., space and time. Then, KiWi does what it can to find as many as possible deep OPSMs based on what is available. Most existing sequential pattern mining methods are exact. The breadth-first methods normally suffer from the inherent large memory consumption problem. The depth-first methods have parsimonious memory usage; however, they cannot perform within-level comparisons to keep the most promising patterns to work on, and a lot of computations are wasted on recursively mining patterns that are eventually shown to be insignificant. With k and w, KiWi can well bound the computational resources. If k is sufficiently large and w is set to m, KiWi becomes exact returning the complete set of patterns.

This best effort approach is particularly suitable for gene expression analysis for the following reason. In gene expression analysis, the more valuable information in a cluster is the genes, not the subspace. The subspace is just a set of experiments that provide evidence or explanation for why the genes form a cluster. The best effort approach used in KiWi may result in some losses of deep OPSMs; however, it may not lose as many clusters because *many* OPSMs would map to the same cluster.

4.2 Overview

We use *pattern* to denote an ordered subset of column labels (pattern elements), e.g., $p^i = [p_1, p_2, \ldots, p_i]$ is a pattern of length i and $p^{i+1} = p^i + p_{i+1}$. p^i can be used to denote either a *candidate* or a *seed* (qualified candidate) at level (iteration) i as they are generated in a level-wise manner, starting with level 1. Conceptually, all seeds from level $i-1$ are extended by each possible single element, creating candidates for level i. Note that this candidate generation step is avoided in the actual implementation. These candidates are assessed using a certain statistic. Based on Observation 1, the k candidates with the highest values of this statistic are retained as the seeds of level i. We use S_i to denote this set

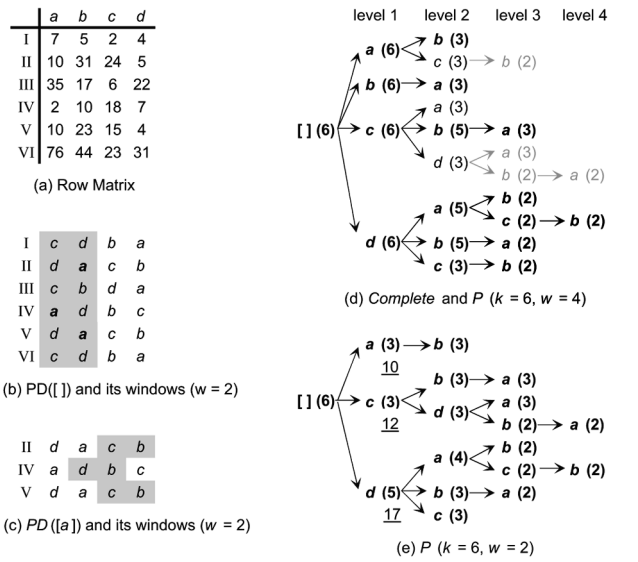


Fig. 3. Running example. (a) Row matrix. (b) PD ([ ]) and its windows ($w = 2$). (c) PD ([a]) and its windows ($w = 2$). (d) *Complete* and P ($k = 6, w = 4$). (e) P ($k = 6, w = 2$).

of seeds. The number of candidates is thus upper bounded by $k \times m$ and the search space is selectively restricted. Based on Observation 2, the elements of a long pattern can be expected to appear early, say, in the next w positions of the supporting sequences. Thus, to test candidates, KiWi considers a vertical slice of width w of the supporting sequences, instead of the entire suffixes.

For each seed $p^i \in S_i$, the *projected database* for p^i, $PD(p^i)$, is maintained containing the sequences supporting p^i under the *w-constraint*, that is, a sequence in $PD(p^i)$ is segmented by the elements of p^i into sections of length no more than w except for the last section. Rapidly shrinking, the projected databases provide a horizontal slicing of the sequence database and reduce the problem scale effectively from iteration to iteration. The usage of w further performs a vertical slicing on the projected databases, dramatically reducing the problem scale since w is usually very small comparing to m. The *window* for each sequence r in $PD(p^i)$ is a region of width at most w in r that immediately follows p_i. Projected databases (*PDs*) and windows are illustrated in Figs. 3b and 3c.

In testing candidate p^{i+1}, only the windows of $PD(p^i)$ are considered. As we have explained, this does not only substantially improve efficiency, but is also necessary for long pattern discovery, avoiding the waste of valuable seats by unpromising candidates.

To rank candidates, support seems to be an apparent and natural choice as the statistic. Experiments also show that this choice can produce reasonably good results. Nevertheless, support does not consider the distribution of column labels in the sequence database. Moreover, many candidates may have tied support values and thus cannot be well distinguished. We introduce the concept of *weighted support*. While each supporting sequence contributes 1 to support, its contribution to weighted support depends on "how it supports," i.e., on the position of the supported pattern in the sequence.

Specifically, let r be a sequence supporting a candidate p^i. Suffix (p_i, r) is the suffix of r starting at element p_i, whose

length, $|\text{suffix}(p_i, r)|$, signals the potential of p^i to continue to grow in r. The weighted support for p^i is defined as the summation of all such suffix lengths over the supporting sequences of p^i, as exemplified in Fig. 3b. The usage of weighted support significantly improves the quality of seeds, leading to the discovery of longer patterns with even small k values. Also, it makes w an insensitive parameter.

All the seeds from different levels, i.e., S_i for different is, constitute the output pattern set $P = \bigcup S_i$. From this set, the deep (long) patterns that correspond to deep OPSMs can be retrieved automatically by comparing against a preset λ threshold or interactively through a user query interface.

Obviously, $P \subseteq Complete$, where *Complete* is the complete set of patterns with support of at least 2. Fig. 3d shows *Complete* in a tree structure. In `KiWi` once w is chosen, $PD(p)$ for each pattern $p \in Complete$ is determined, as well as the statistic value for p. Then, it depends on k whether p would appear in P or not. At a given length, some top k patterns in *Complete* may not survive if their prefixes failed; the extra seats will be taken by some patterns not among the top k in *Complete*, but having their prefixes survived in the previous level. In other words, $p^l \in S_l \Rightarrow p^i \in S_i$ for $1 \leq i \leq l$, meaning, for p^l to be discovered, each of its prefixes p^i has to appear among S_i.

4.3 Running Example

In Fig. 3, the raw data matrix with four experiments and six genes in Fig. 3a is transformed into a 6×4 sequence database as shown in Fig. 3b. Figs. 3b and 3c exemplify the *PDs* and their windows (the shaded areas) in growing pattern $[a]$ with respect to $w = 2$, from which we can see how the problem scale is reduced level by level. Fig. 3d shows *Complete* and the actual support for each pattern and also shows the pattern discovery process for $k = 6$ and $w = 4$, in which support is used as the statistic for simplicity of illustration. For each level of 1 to 4, the nongray-colored patterns are candidates and the dark-colored ones are eventually chosen as seeds.

The results in Fig. 3d are reasonably good, but one longest pattern, $[c, d, b, a]$, is missing. Level 2 of Fig. 3d also shows the ranking problem that some candidates with support 3 are chosen as seeds, some not. Fig. 3e shows the mining process when w is adjusted to 2. The results in Fig. 3e are very good with both the longest patterns discovered, together with most of the more frequent patterns at each level. Comparing to Fig. 3d, the $w = m$ case, the support values are suppressed but in a biased fashion. Some candidates appearing early have their support values less suppressed; thus, they are favored and have increased probabilities to emerge as seeds.

When weighted support is used, w is less sensitive. For $w = 4$, 3, or 2, exactly the same results are returned as in Fig. 3e. The underlined numbers in Fig. 3e for level 1 show the weighted support values for the candidates, e.g., the weighted support for $[a]$ is 10, which can be verified in Fig. 3b. In Fig. 3b, the three a-suffixes for sequences supporting $[a]$ (II, IV, and V) have the total length of 10. Note that only windows for $PD([\,])$ are considered in testing candidate $[a]$. From Fig. 3e, we can also see that if weighted support is used, even for $k = 2$, the two longest patterns will still be returned.

4.4 Algorithm

Algorithm 1 gives the pseudocode for the `KiWi` pattern discovery algorithm. Since candidate testing dominates the runtime, the pseudocode provides certain low-level details toward this respect. Column labels are renamed with integers 0 to $m-1$ before the raw data matrix is transformed into the sequence database SB; thus, each sequence in SB is a permutation of a subset of $\{0, 1, \ldots, m-1\}$. The output pattern set P contains S from different levels, at most k for each level and organized as linked lists in order to save space. S is a list of seeds, each being a pattern and a list of elements. [ ] denotes the empty list. PD is a list of projected databases, each for a seed in S with the same index. For example, $PD[s]$ stores a list of indexes of sequences supporting the seed $S[s]$. $Stats$ is a list of integers used for statistic counting, which always has the size of $m \times |S|$ since we preserve m positions for each seed. The data structures S, $Stats$, and PD are reused from iteration to iteration.

Algorithm 1 `KiWi` pattern discovery

Input: $n \times m$ sequence database SB, k, w
Output: P

```
1:  P ← ∅;
2:  S = [[]];
3:  PD = [[0, 1, ... n − 1]];
4:  while (true)
5:    Stats.initialize(m × |S|, 0);
6:    for each seed in S with index s
7:      for each sequence r in PD[s]
8:        for each element e in the window of r
9:          Stats[m × s + e] += |suffix(e, r)|;
10:       end for
11:     end for
12:   end for
13:   S.update(k, Stats);
14:   S.prune(2);
15:   if |S| = 0 then
16:     break;
17:   end if
18:   P ← P ∪ S;
19:   PD.update(S);
20: end while
```

Lines 1-3 perform initialization. P starts with an empty set. S initially contains one empty pattern [ ]. $PD([\,])$ contains all the n sequence indexes. Lines 4-20 mine all levels of S from iteration to iteration. During each iteration, lines 5-12 perform candidate testing. Then in line 13, S is updated with the (at most) k candidates having top values in $Stats$. Then in line 14, S gets rid of the seeds with support less than 2. Lines 15-17 check the loop termination condition. If there is no seeds in S, pattern growth cannot continue. Otherwise, S is added to P (line 19), PD is updated for seeds in S (line 20), and the next iteration starts. When the loop ends, P contains the computed S from all levels.

Note that it is possible to record the supporting sequence indexes for each candidate during candidate testing. But that way, the memory and CPU consumptions are significant. We choose to update PD after the update of S, when we only need to process at most k seeds. The runtime of this update is dominated by candidate testing and negligible.

Candidate testing. Generally, support counting is the major time consumer in frequent (sequential) pattern

mining. In principle, KiWi achieves efficiency in statistic counting by upper bounding the number of candidates, as well as horizontal (line 7) and vertical (line 8) slicing of the sequence database.

Other than that, KiWi avoids explicit candidate generation. Comparing to conventional sequential pattern mining, m is not big in OPSM mining. Thus, we preserve m positions in *Stats* for each seed since it has at most m ways to grow one more element. In particular, let $S[s]+e$ be a candidate with e being the newly appended element. Then, $s' = m \times s + e$, meaning that the statistic incrementing for the candidate will be done in $Stats[s']$ (line 9). Conversely, knowing s', an index in $Stats$, we have $e = s' \bmod m$ and $s = s' \operatorname{div} m$, meaning that the statistic stored in $Stats[s']$ is for candidate $S[s]+e$. This converse derivation is used in line 13 to update S, growing the linked lists properly.

In Algorithm 1, lines 5-12 perform statistic counting. First, $Stats$ is initialized with $m \times |S|$ 0s (line 5). Then, for each seed in S with index s (line 6), for each sequence r in the projected database of $PD[s]$ (line 7), and for each element e in the window of r (line 8), weighted support is incremented for candidate $S[s]+e$ (line 9).

Use of inverse sequence database. Locating windows of projected databases and incrementing weighted support values require intensive accesses to column indexes. Since each sequence in the sequence database SB contains unique elements, for a given row index, SB can be considered as a one-to-one function mapping column indexes to sequence elements. Therefore, the inverse function, denoted as ISB and essentially a matrix of the same size as SB, maps sequence elements to column indexes in a one-to-one manner.

ISB can be used in any situation requiring the retrieval of the column index for a given element. In line 8 of Algorithm 1, window of r just corresponds to some column index c with $c - ISB[r][last] \leq w$ where $last$ is the last element of $S[s]$. Without ISB, we have to either perform a linear search for, or record, $ISB[r][last]$ for each sequence r supporting each seed in S. In line 9, $|\text{suffix}(e, r)|$ can be quickly calculated as $|SB[r]| - ISB[r][e]$. In line 19, to update PD, for each new seed $S+e$, we only need to check each sequence r in $PD[s]$, and see if $ISB[r][e] - ISB[r][last] \leq w$. If it is, r also supports the new seed. ISB is also used in pattern extension as to be discussed.

The SPADE [52] algorithm maintains a list of sid (sequence id) and eid (element id) pairs for each pattern. The lists of sid correspond to the PDs in KiWi. The lists of eid are condensed into a single static data structure ISB in KiWi, making use of the fact that in OPSM mining, each column label (item) in the alphabet appears exactly once in a sequence.

Computational analysis. The runtime of KiWi is dominated by candidate testing in lines 5-12 of Algorithm 1. By upper bounding the number of visits of elements in the windows, we can obtain the worst case complexity as $O(lkwn)$, where l is the length of the longest pattern. Note that for each pattern, there are at most n supporting sequences.

Now we provide the average case analysis assuming a random data matrix. In such a matrix, each pattern of length i is on average supported by $n \times \frac{1}{i!}$ sequences. Knowing that $i! \geq 2^i$ for $i \geq 1$, the average case runtime is

$$\begin{aligned} &O\left(kwn \times \left(\frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{l!}\right)\right) \\ &= O\left(kwn \times \left(\frac{1}{2^0} + \frac{1}{2^1} + \cdots + \frac{1}{2^l}\right)\right) \\ &= O(kwn \times 2) = O(kwn). \end{aligned}$$

Note that the average case runtime is linear in n for fixed min_sup of 2. Some other algorithms also show similar empirical results, but their support is defined as a fraction of n, i.e., min_sup increases proportionally with the increase of n.

The worst case space complexity is $O(mn + km + kn)$, where mn is for SB and ISB, km for $Stats$, and kn for PD since each pattern can be supported by at most n sequences. The memory taken by S is $O(k)$ and negligible. In general, it is reasonable to assume that SB can be well accommodated in memory, and normally $n \gg m$; thus, PD would incur the largest memory consumption. In practice, the support for each seed reduces dramatically along the growing of the seed. It can be very big for the first iteration, but in that case the number of patterns is upper bounded by $\min(k, m)$. Overall, by adjusting k, the total memory consumption in KiWi can be well controlled.

5 KiWi Extensions

In this section, we discuss technical extensions to the KiWi framework for improved efficiency, effectiveness, and usability.

5.1 Parameter Tuning

KiWi is a best effort approach that strives to optimize its performance based on what is available. In particular, k and w are the tunable parameters used to bound the computational resources. With unlimited k and w, KiWi returns the complete set of patterns. Now we discuss how to automatically determine the parameters based on the available computational resources.

Determining k. Parameter k can be determined based on the available memory. Recall that the worst case space complexity is $O(mn + km + kn)$. By adjusting k, we want to make sure KiWi would not incur memory spills. It is possible to do so because the amount of memory consumed by the data structures in KiWi can be precisely calculated based on k, m, and n, where m and n are given.

As shown in Algorithm 1, PD is the largest memory consumer. Other data structures, such as SB, ISB, and S, take relatively small amount of memory constantly from iteration to iteration. The memory consumption by PD is unstable; however, within each iteration, it can be calculated (line 13 of Algorithm 1) before it is actually used (line 19).

Based on this observation, we can determine k dynamically from iteration to iteration so that the total memory consumption is always within the memory constraint. In more detail, in lines 6-12 of Algorithm 1, the statistics for the candidate patterns are collected and stored in $Stats$, from which we know the size of projected database (support) for each candidate. Then, in updating S (line 13), we choose the

top k from $Stats$ such that the calculated PD is within the given constraint. In line 19, PD is actually filled but it will not cause a memory spill.

One drawback of the above solution is that there would be a few iterations where PD is very large and a relatively small k has to be used. Then these iterations would become the performance bottleneck of KiWi. A remedy is to maintain a sufficiently large k for such iterations at a reasonable efficiency cost.

Observe that the projected database of a superpattern is a subset of the projected database of a subpattern. The implication is that at any level, we can make use of PD of any previous level if we do not have an updated one. The earlier the level we use, the more efficiency we lose. If we have to use the initial PD as in line 3, all the efficiency that PD intends to offer is lost for this level, since the problem scale does not shrink from the whole sequence database SB. While we could use the entire PD from some previous level, for best efficiency, a better strategy is to update PD as much as possible from iteration to iteration. In this case, PD can be mixed containing both unupdated entries and updated entries.

Determining w. Let l_u be the minimum pattern length that the user is interested in. If the user does not specify l_u, a default value will be used, which is the minimum pattern length that would make OPSMs of size $2 \times l_u$ statistically significant in an $n \times m$ data matrix.

Then, we run the R program (described in Appendix A) with l_u as the input pattern length. The program will return the cdf of X, where X is the maximum length of sections segmented by a pattern of length l_u.

Then, we choose w such that $P(X \leq w) \approx s$, where s can be set to, e.g., 0.5. This guarantees that, on average, the biased sampling will sample $n_s = s \times n$ of the sequences supporting each prefix of a pattern of length $\geq l_u$. Thus, the choice of w is actually the choice of sample size.

Parameter $s = \frac{n_s}{n}$ is the ratio of sample size over population size. In practice, when n is large, we tend to choose a smaller s, leading to a smaller w and reduced runtime. In statistics, suppose we have determined that sample size n_0 is needed (in estimating mean or proportion) to achieve a desired precision and confidence level; for finite population, a correction is made as follows:

$$n_s = n_0 \times \frac{n}{n_0 + n - 1},$$

where n_s is the adjusted sample size [11]. Then, $\frac{n_s}{n} = \frac{n_0}{(n_0-1)+n}$, from which we can see that $s = \frac{n_s}{n}$ decreases when n increases.

Example 4. Suppose $m = 90$ and $l_u = 20$. Use them as input parameters to run the R program. With $s = 0.5$, since $P(X \leq 12) \approx 0.5$, we choose $w = 12$. This guarantees that, on average, for each candidate pattern p of length $\leq l_u$, about 50 percent of the sequences supporting p will be used in support counting.

5.2 Backward Mining

We extend KiWi to perform backward mining in different ways for two purposes, right-leaning pattern mining and GOPSM mining.

Right-leaning pattern mining. In KiWi, a pattern p^l must have all its prefixes survived (among top k) in 1 to l levels. If one prefix fails, p^l will not be discovered.

Depending on where the elements of a pattern appear in its supporting sequences, the pattern can be *evenly distributed*, *left-leaning*, or *right-leaning*. For an evenly distributed pattern, the sections in the supporting sequences segmented by the pattern tend to be even in size on average. KiWi favors such patterns.

For a left-leaning pattern, the sections tend to be small in size on the left-hand side and large on the right-hand side. In other words, the elements of the pattern appear dense on the left and sparse on the right. Such patterns are relatively easy to be discovered for the following reasons. Since a random sequence supports a pattern with length l with the probability of $\frac{1}{l!}$, the support of a pattern contributes to its survivability (chance of being selected as seed) increasingly from round to round. Then a planted pattern with a certain support value would receive less and less competition from round to round. Thus, if a pattern survives in the early rounds, it is likely that it will eventually survive. Then, left-leaning patterns are favored because KiWi works from left to right.

For a right-leaning pattern, the sections tend to be large in size on the left-hand side and small on the right-hand side. Based on the above argument, such patterns are disfavored by KiWi. To solve this problem, we extend KiWi to perform backward mining, where the mining process goes from right to left from iteration to iteration. This requires an additional run of KiWi. Since the two runs are independent, a simple way of implementing such backward mining is to produce a reversely ordered sequence database from the original data matrix and then run KiWi.

GOPSM discovery. Backward mining can also be used for mining generalized OPSMs. Recall that in a GOPSM, the sequence of expression values can be either strictly ascending or descending. GOPSMs are able to capture anticorrelations among genes.

The most straightforward approach to mine GOPSMs is to double the size of SB by appending a list of reversely ordered sequences, and then run KiWi as in normal OPSM mining. While this approach works fine, it is not memory efficient, since both SB and ISB would need to be doubled in size.

We propose to use backward mining. Different from the way used for right-leaning pattern mining, here forward mining and backward mining need to be performed simultaneously. Specifically, the projected database $PD(p)$ for each pattern p can be divided into two groups: one normally supports p and the other reversely supports it. Then in candidate testing, we know in which direction we can locate the testing bed, i.e., the windows (line 8 of Algorithm 1), to perform statistic counting.

5.3 OPSM Formation and Pattern Extension

OPSM formation. To form OPSM clusters, we need to scan the sequential database SB to retrieve the supporting sequences for the discovered patterns. These supporting sequences are not stored in the pattern discovery phase for several reasons. First, such storage costs significant amount of memory. Second, due to the biased sampling technique

used in candidate testing, sequences in $PD(p)$ constitute an incomplete portion of the entire set of supporting sequences for pattern p. Third, not all the patterns are interesting, e.g., some intermediate short patterns.

Before the scanning, we want to remove those uninteresting patterns first from all the discovered patterns. For this purpose, we provide an interactive query interface that allows users to select interesting patterns based on length and support. Then, the sequence database SB is scanned once to obtain the supporting sequences for the selected patterns. During the scanning, we need to check if a selected pattern p is a subsequence of a sequence r. We can again use ISB to achieve $O(|p|)$ runtime for this operation, instead of $O(m)$. Note that usually $|p| \ll m$. We use an example to illustrate the main idea. Let $p = [a, b, c]$. If p is supported by r, then $ISB[r][a] < ISB[r][b] < ISB[r][c]$.

The query interface is also used after OPSM formation, when users want to retrieve and examine OPSMs of different sizes. Some convenient features are added to the interface for this purpose.

Pattern extension. A pattern p may not grow into its maximal superpattern p^l that has the same supporting sequences. Such p^l is also called a *closed* pattern. Some intermediate elements of p^l may be missing in p because the corresponding candidates did not get high enough ranks to emerge as seeds; however, some succeeding elements may have survived and eventually reached p_l.

Example 5. Suppose $[a, b, c, d, e, f]$ and $[a, b, d, e, f]$ have the same support in SB. $[a, b, c]$ may have a low support (statistic) and not get selected. However, $[a, b, d]$ may have a higher support and get selected, eventually growing into $[a, b, d, e, f]$.

For the purpose of finding closed patterns, we provide a pattern extension heuristic that is executed during OPSM formation, where each pattern p is extended after obtaining its supporting sequence set $R(p) \supseteq PD(p)$.

Now we give the main ideas of the heuristic. Pattern p segments each sequence in $R(p)$ into $|p| + 1$ sections of length ≥ 0. The same-positioned sections from all sequences in $R(p)$ form a *chunk*. Within each chunk, if there is any element e that appears $|R(p)|$ times, it can be inserted into p as a new element.

It is rare, but possible that there are two or more such common elements within each chunk. We can use a bottom-up approach to find a longest ordered subset of them that has the support of $R(p)$. After the extension, all patterns are closed.

5.4 Statistical Significance of OPSMs

The number of discovered OPSMs may be very large. Although we provide a query interface allowing users to select interesting OPSMs of certain size, it is highly desirable to reveal the statistical significance of OPSMs and to rank them according to the significance.

However, since the underlying distribution of the test statistic is complex and unknown, it is hard to analytically obtain p-values of OPSMs. To get around the problem, we provide two approaches for the estimation of p-values. The first approach uses upper bounds. The second approach uses simulation with bootstrap sampling.

Recall that the p-value is the probability of obtaining a value of the test statistic at least as extreme as the one observed by chance. We use $p(R, C)$ to denote the p-value for OPSM (R, C). Intuitively, an OPSM with a larger size is more significant. In other words, $p(R', C') \leq p(R, C)$ if $|R'| \geq |R|$ and $|C'| \geq |C|$. Specifically, if two OPSMs have the same size, then they have the same p-values. Thus, we use (r, c) to denote a *model* representing all the OPSMs with r rows and c columns. We use $p(r, c)$ to denote the p-value for all the OPSMS that (r, c) represents.

Upper bounds. The probability that a random row supports a pattern with length c is $\frac{1}{c!}$. Assuming independence among the rows, the probability of having at least r rows supporting the pattern is the r-tail of the $(n, (\frac{1}{c!}))$ binomial distribution

$$\sum_{i=r}^{n} \binom{n}{i} \left(\frac{1}{c!}\right)^i \left(1 - \frac{1}{c!}\right)^{(n-i)}.$$

In total, there are $\frac{m!}{(m-c)!}$ possible patterns; thus, $U_0(r, c)$ is an upper bound of $p(r, c)$, where

$$U_0(r, c) = \frac{m!}{(m-c)!} \sum_{i=r}^{n} \binom{n}{i} \left(\frac{1}{c!}\right)^i \left(1 - \frac{1}{c!}\right)^{(n-i)}.$$

Here, $U_0(r, c)$ is used in [8] to estimate the significance of model (r, c). However, this approach can be problematic because these upper bounds do not necessarily preserve the ordering of the corresponding p-values.

Example 6. Let $n = 10{,}000$ and $m = 100$. Consider two models $(1, 80)$ and $(5, 120)$. $U_0(1, 80) = 100 \times 1 = 100$. $U_0(5, 120) = \frac{100!}{95!} \times 0.00008773 = 792{,}597$. From the results, $(1, 80)$ is more significant than $(5, 120)$, which is counterintuitive.

To solve the problem, we use $U(r, c)$ to upper bound $p(r, c)$, taking into consideration all the more extreme models, i.e., (r', c') with $r' \geq r$ and $c' \geq c$:

$$U(r, c) = \sum_{r'=r}^{n} \sum_{c'=c}^{m} \frac{m!}{(m-c')!} \sum_{i=r'}^{n} \binom{n}{i} \left(\frac{1}{c'!}\right)^i \left(1 - \frac{1}{c'!}\right)^{(n-i)}.$$

It is trivial to prove that for $r' \geq r$ and $c' \geq c$, $U(r', c') \leq U(r, c)$, indicating that (r', c') is a more significant model than (r, c).

Upper bounds can be used to order OPSMs. However, they do not provide accurate estimation of their p-values. In the following, we use bootstrap-based simulation to estimate p-values of OPSMs.

Bootstrap-based simulation. In cases where the test statistics are complex, the p-values based on traditional analytic approximations are often inaccurate, and the only reliable way to calculate p-values is by simulation. We sketch a bootstrap-based simulation approach for estimating p-values of OPSMs in an $n \times m$ data matrix as follows:

Let A be an $n \times m$ random matrix, where $a_1, a_2, \ldots, a_n$, are the n rows, each being a random permutation of m distinctive symbols. We use a binary random variable $Y(r, c)$ for each model (r, c). Let $Y(r, c) = 0$ if (r', c'), $r' \geq r$

and $c' \geq c$, is not observed in A; $Y(r,c) = 1$ otherwise. Obviously, $p(r,c) = E(Y(r,c))$. To estimate $p(r,c)$ using bootstrap resampling, we choose an integer B, and for $1 \leq b \leq B$, do the following:

1. Generate a new $n \times m$ matrix $A^{(b)}$ with rows $a_1^{(b)}, a_2^{(b)}, \ldots, a_n^{(b)}$, where $a_i^{(b)}$s are randomly selected from the rows of A with replacement. Note that it is possible that $a_1^{(b)} = a_2^{(b)}$.
2. For each matrix $A^{(b)}$, mine the OPSMs.
3. Let $Y^{(b)}(r,c) = 0$ if (r',c'), $r' \geq r$ and $c' \geq c$, is not observed in $A^{(b)}$; 1 otherwise.

Then, $p(r,c)$ can be approximated by

$$p^*(r,c) = \frac{1}{B}\sum_{b=1}^{B} Y^{(b)}(r,c).$$

The bootstrap method [14] has been extensively used in estimating a feature of a distribution such as p, especially when analytic solutions are impossible. It has been shown that the bootstrap estimator, p^*, is a consistent estimator for p [44], i.e., for every $\epsilon > 0$,

$$lim_{n\to\infty} P(|p_n^* - p| < \epsilon) = 1.$$

It is almost certain that p^* will be arbitrarily close to p when n is large enough. In addition, the estimator becomes more accurate as B increases. It is suggested that $B = 10n$ be used to make the variability due to the bootstrap simulation small [12]. A smaller B may be used if little change in p^* is observed as the number of iterations increases.

Note that for Step 2, it suffices to mine the closed patterns only. If we can observe a closed pattern (corresponding to a model), then all the nonclosed patterns derivable from the closed pattern (corresponding to submodels) can be observed. To further improve efficiency, we could consider mining *closed models*, where each closed model corresponds to many closed patterns. We could also use other frequency estimation techniques [36] in the mining process.

6 Experimental Evaluations

We performed extensive biological experiments on real data sets to demonstrate that `KiWi` can discover deep OPSM clusters that are highly meaningful. We also performed comprehensive computational experiments on real and synthetic data sets to further validate the effectiveness and efficiency of `KiWi`. For the purpose of the experiments, we implemented `KiWi` in `C++`, together with a dynamic programming-based algorithm that finds the longest pattern (will be referred to as `LoPad`), and a synthetic data generator. PrefixSpan [40] was obtained from the authors.

6.1 Biological Evaluation

6.1.1 Introduction

The objective of this series of experiments is to demonstrate that `KiWi` can effectively discover deep OPSM clusters that are biologically meaningful from massive data sets. Recall that an OPSM is deep if it falls in the deep region, which refers to part of the significant region that is below λ. As discussed in Section 2, we conservatively set $\lambda = 0.002 \times total\ number\ of\ rows$.

Deep OPSM clusters are small in size and large in dimensionality. Such clusters are of particular interest to biologists as these may represent small groups of genes that are tightly coregulated under many conditions. Previous studies have tended to focus on larger clusters by design or necessity. While these large clusters have been shown to be of interest or value, there is no reason to expect that all or even most biological processes or disease mechanisms would involve tight coregulation of large groups of genes. In fact, we expect that many important processes will involve relatively small numbers of genes. Some pathways or processes might require only two genes to act in concert.

Indeed, our own biological assessments discussed below showed that many of these small OPSM clusters are of biological interest. Specifically, we demonstrated that these clusters correctly group related probe sequences, tend to share common biological processes and common regulatory sequences, and share common experimental annotation terms such as tissue source, gender, ethnicity, etc.

6.1.2 Methods

Data sets. Several data sets and methods were utilized to test for biological coherence of deep OPSM clusters predicted by `KiWi`. Expression data sets utilized include: 1) GPL96—a set of 1,640 Affymetrix (HG-U133A) experiments from the Gene Expression Omnibus (GEO, GPL96) [7] covering a broad range of experimental conditions; 2) expO—a set of 1,026 Affymetrix (HG-U133 Plus 2.0) experiments from 123 different cancer tissue types from the expO (Expression Project for Oncology) project (GEO, GSE2109). These data sets are summarized in Table 1.

Data set processing. The Affymetrix HG-U133A (GPL96) probes were normalized and mapped to gene/protein identifiers as previously described [21]. The expO data were normalized from CEL files using the Bioconductor "just.gcrma" function in the "gcrma" library (version 2.4.1). Probes were mapped to Uniprot and Ensembl identifiers using the Bioconductor "biomart" package [18]. Tab-delimited data matrices for each data set were loaded into the `KiWi` software (v. 1.0) and OPSM clusters identified using the parameters outlined in Table 1. Parameters were chosen by experimentation to identify values of k and w that would produce the largest number of clusters and longest patterns but still run to completion in 24 to 48 hours.

Gene ontology analysis. The first validation method used the GO, a set of structured, controlled vocabularies to identify functional associations between gene products [6]. Current GO annotations and external references file were downloaded from the Gene Ontology Annotation resource at EBI (http://www.ebi.ac.uk/GOA/). Each cluster of protein ids was submitted to the High-Throughput GoMiner command-line interface [53]. Statistically overrepresented GO terms were defined using Fisher's exact test and corrected for multiple testing by false discovery rate detection (100 permutations).

oPOSSUM TFBS analysis. The second validation method uses the oPOSSUM tool to identify statistically overrepresented transcription factor binding sites (TFBS) [25]. The oPOSSUM API and MySQL database were

TABLE 1
Data Sets and Parameters Used in KiWi Assessment

Dataset	# of genes	# of exps.	λ	k	w	Min. genes	Min. exps.
GPL96	12,332	1,640	25	30,000	45	2	10
expO	20,113	1,026	40	100,000	18	2	10

TABLE 2
KiWi Results

Dataset	# clusters found	Mean genes per cluster (range)	Mean exps. per cluster (range)	# of deep clusters (%)
GPL96	13,412	5.11 (2 to 162)	24.04 (11 to 108)	13,195 (98.4%)
expO	23,555	3.89 (2 to 23)	42.48 (10 to 120)	23,555 (100%)

Note: the minimum values seen for the ranges of numbers of genes and experiments were set in KiWi as parameters (see Table 1).

downloaded and installed locally (http://www.cisreg.ca/cgi-bin/oPOSSUM/oPOSSUM). Each cluster of genes was submitted to the software and statistically overrepresented TFBSs were defined using the Z-score option.

Grouping of probes to common gene identifier. Affymetrix gene expression chips (such as the HG-U133 Plus 2.0 platform, used for expO) contain numerous probe sets derived from the same gene either as redundant probe sets or probe sets for different transcripts of the same gene. It is expected that such probe sets will display correlated expression given that they measure the same or related transcripts. Therefore, we expect that probe sets mapped to a common gene identifier will be grouped together in the same subspace cluster more often than expected by chance. This represents a kind of positive control experiment. The number of probe pairs mapped to the same gene was determined for all clusters. Significance was assessed by random permutation analysis. That is, 10,000 sets of clusters were randomly generated (with the same sizes and dimensions as produced by KiWi). The mean number of redundant probe pairs for all clusters was then compared between KiWi and the distribution of random results.

Experimental annotation analysis. KiWi OPSM clusters consist of a set of two or more genes found to have correlated expression patterns for some subset of the available experiments. Most validation methods look for biologically consistent grouping of genes. To determine if the experiments were also grouped in a meaningful way, an overrepresentation analysis was applied to experimental annotations. The expO data set was chosen as this data set is accompanied by carefully annotated clinical details. For example, each experiment is annotated as one of 123 different tumour tissue types. Clusters with a minimum of two genes and 50 experiments were selected for analysis. Statistically overrepresented experimental annotation terms were defined using Fisher Exact statistics and corrected for multiple testing by a Benjamini and Hochberg (BH) correction with the Bioconductor "multtest" package. Overall significance of the KiWi clusters was determined by comparing to randomly generated clusters using a Kolmogorov-Smirnov test.

6.1.3 Results

OPSM discovery results. For both data sets analyzed, KiWi was able to run to completion (after parameter optimization) and produce a large number of OPSM clusters. The results are summarized in Table 2 and density distributions for cluster size (number of genes) versus pattern length (number of experiments) plotted in Fig. 4. The density plots were produced using the Bioconductor "hexbin" library (version 2.3.0). A large number of clusters (13,412 and 23,555) were identified for the two data sets, with a range of sizes and pattern lengths. In general, KiWi appears well suited to identifying smaller clusters with long patterns. For example, in the expO data (Fig. 4B) 80 percent of clusters had five genes or fewer, 97 percent had 10 genes or fewer, and 100 percent were under the λ threshold and thus deep. These clusters, while small in gene number, are in many cases coexpressed across a large number of experiments. For clusters with 5 genes or fewer, the number of experiments over which coexpression was observed ranged from 10 to 120 with a mean of 42.48. Similar trends were seen for the GPL96 data (Fig. 4A). For the GPL96 and expO data sets, the average cluster size was 5.11 and 3.89 and the average pattern length was 24.04 and 42.48, respectively.

GO and oPOSSUM analysis. GO and oPOSSUM analysis results are shown for the GPL96 data set (Figs. 5 and 6). For the GO validation, a set of representative clusters were chosen with minimum size 5 and minimum dimensions 15. A total of 634 clusters met these criteria, with 22 clusters containing anticorrelated genes. The GO analysis shows that clusters identified by KiWi are significantly more likely to share a common biological process than random expectation (Fig. 5). The fraction of clusters with at least one significantly overrepresented GO biological process term at each level of significance are shown. For example, if we consider a p-value threshold of 0.01, more than 10 percent of clusters have at least one significant GO term compared to the random expectation of close to zero.

Similarly, the oPOSSUM TFBS analysis shows that clusters identified by the KiWi algorithm are significantly more likely than random expectation to share sequences bound by the same transcription factor (Fig. 6). The fraction of clusters with at least one significantly overrepresented transcription factor binding site (TFBS) at each level of significance is shown. For example, if we consider a Z-score of 30, more than 10 percent of clusters have at least one TFBS overrepresented in the regulatory regions for these genes. Random expectation for this same Z-score threshold is close to zero. To summarize, the KiWi algorithm

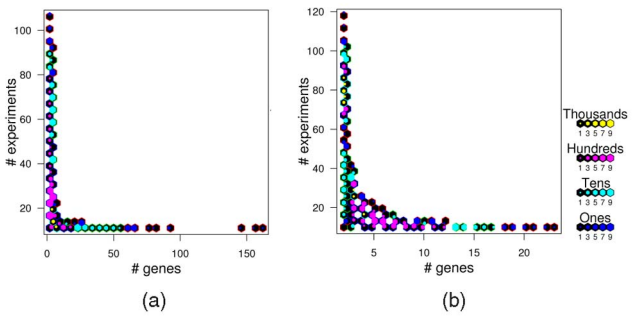


Fig. 4. KiWi results for GPL96 and expO data sets. (a) GPL96, (b) expO.

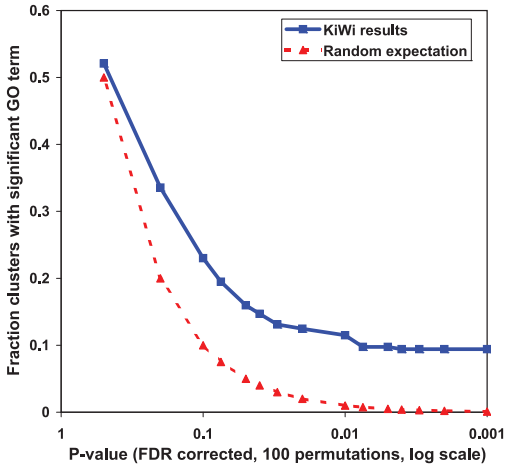


Fig. 5. Gene Ontology analysis for GPL96 data set.

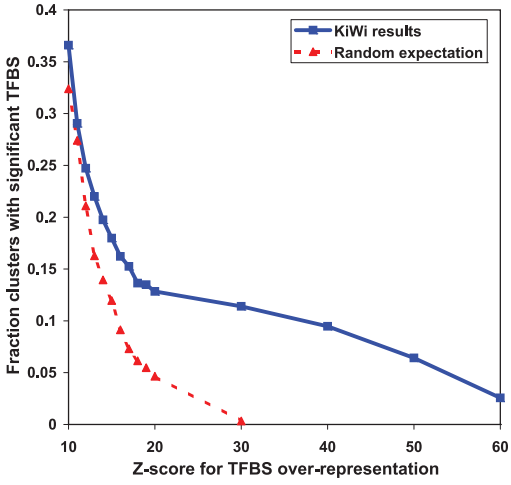


Fig. 6. oPOSSUM TFBS analysis for GPL96 data set.

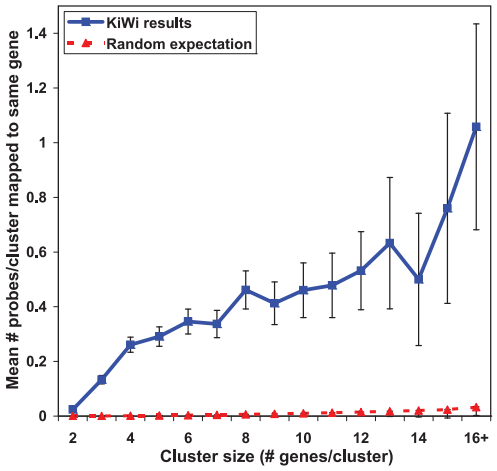


Fig. 7. Probe to common gene analysis for expO data set.

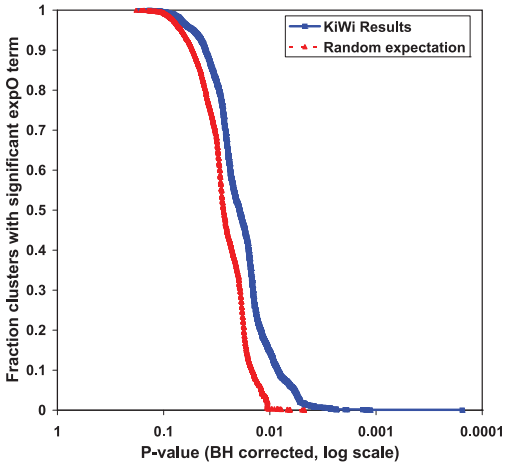


Fig. 8. Experimental annotation analysis for expO data set (all annotations).

successfully identifies groups of genes that belong to a common function or process and/or share common transcription factor binding sites.

Grouping of probes to common gene identifier. Fig. 7 shows the results of the "probe to common gene" analysis. The mean number of probe pairs in a cluster that are mapped to the same gene (redundant probes) is shown for each cluster size (number of genes per cluster) for the expO data set. (Error bars indicate 95 percent confidence limits.) Using the expO data set (because it is based on the more current Affymetrix platform), we found that of the 23,555 clusters identified by `KiWi`, 1,880 (7.98 percent) contained at least one pair of redundant probes (i.e., different probes corresponding to the same gene) and on average, KiWi clusters contained 0.177 redundant probe pairs per cluster. This was significantly more than 24.5 (0.10 percent) clusters with at least one redundant pair ($p < 0.0001$, 10,000 permutations) and the 0.002 average number of redundant probes per cluster ($p < 0.0001$, 10,000 permutations) identified in our random simulations.

Experimental annotation analysis. Fig. 8 shows the results of the "experimental annotation analysis." The fraction of clusters with at least one significantly over-represented experimental annotation term at each level of significance are shown. This used the expO data because unlike most gene expression data sets, the expO data are accompanied by careful and comprehensive experimental annotations. The graph shows a significant tendency by `KiWi` to group experimental dimensions with common experimental annotation terms such as tissue source, histology, gender, ethnicity, smoking, or alcohol consumption status ($p = 0.009$).

6.1.4 Discussion

GO and oPOSSUM performance were evaluated using a large Affymetrix data set (referred to here as GPL96). We showed that `KiWi` can group genes with overrepresentation of GO biological processes and oPOSSUM transcription factor binding sites. Using another large Affymetrix data set (expO), we show that `KiWi` is also very good at grouping probes for the same gene. In fact, a significant proportion of the total clusters contain "redundant" probes. In itself, this is not a surprising result, but is an important positive control that shows `KiWi` correctly identifies logically related genes. The expO data set was also useful for its evaluation of experiment-level clustering by `KiWi`. Part of the promise of subspace clustering methods is that they will identify not only coexpressed genes but also the subset of experimental tissues or conditions under which genes are coexpressed. Such clusters could be of particular value for

TABLE 3
KiWi versus LoPad: Longest Pattern Discovery

Dataset	# of genes	# of exps.	LoPad	Time (second)	KiWi	Time (second)
breast cancer	3226	22	16	87	16	1
SU-gene	6990	85	51	2807	51	13
SU-prot	8727	85	51	3529	51	15
SAGE-gene	20283	243	59	28995	59	701
SAGE-tag	153204	243	55	32585	54	858
GPL96	12332	1640	161	341476	160	1458
cDNA	12671	2852	NA	NA	228	1735

identifying tissue- or stage-specific coregulation. But, they also allow identification of coregulation for previously unconsidered sample groups. For example, we were able to identify gene clusters specific to gender, smoking and alcohol consumption status.

6.2 Computational Evaluation

The objective of this series of experiments is to validate the effectiveness and efficiency of KiWi in discovering deep OPSM clusters from massive data sets.

6.2.1 Effectiveness

Data sets. These experiments used real data sets including Affymetrix (HG-U133A) experiments from the Gene Expression Omnibus (GEO, GPL96) [7], SAGE libraries also from GEO (GPL4), and cDNA experiments from the Standford Microarray Database (SMD) [26]. Affymetrix probes, cDNA clones, and SAGE tags were normalized and mapped to gene/protein identifiers as previously reported [21]. For comparison with existing algorithms, two small data sets, SU [46], a smaller data set of Affymatrix (HG-U95A) experiments, and breast cancer [24], were also used. Table 3 lists the sizes of these data sets.

Results. Deep OPSMs correspond to long patterns. To demonstrate that KiWi is effective in finding deep OPSMs, we compare KiWi with LoPad, which can find the longest pattern with respect to min_sup of 2. Table 3 reports the longest pattern length and corresponding running time (in seconds) for LoPad and KiWi, respectively. We observe that in all cases KiWi was able to find the optimal results or very close, in significantly shorter time. Note that LoPad, with runtime complexity of $O(m^2n^2)$, did not return after one week for the cDNA data set and the program was terminated.

More frequent subpatterns are likely to grow into more frequent superpatterns. The support of a pattern reflects its potential to continue to grow. Thus, an algorithm that is capable of mining patterns with large support values should be also capable in mining long patterns. Ben-Dor et al. [8] proposed an algorithm (will be referred to as Alg) that strives to find the most significant OPSM one at a time. Note that for a pattern of given length, more support means more significance. They reported several significant OPSMs with different pattern lengths on the breast cancer data set. While they did not give the runtime, KiWi in just 1.47 seconds found much better results (larger support values) for every case they reported as shown in Table 4. In addition, since this is a very small data set, we were able to set $min\_sup = 2$ and obtain the results from PrefixSpan in 2,085 seconds (with output file size of 12 GB), from which we can see that our results are actually optimal.

TABLE 4
KiWi versus Alg: Significant OPSM Discovery

Pattern length	Alg	KiWi	PrefixSpan
4	347	795	795
6	42	129	129
8	7	19	19

6.2.2 Efficiency

We used synthetic data to demonstrate the efficiency and scalability of KiWi. Random matrixes of different sizes were generated by the synthetic data generator. The results shown in Fig. 9 confirm the average case complexity of $O(kwn)$. In particular, Fig. 9c shows that the runtime of KiWi is linear in n. Note that in KiWi , min_sup is always fixed at 2. As previously mentioned, some existing algorithms also show a similar trend for some fixed "min_sup," but their support is defined as a fraction of n, i.e., with the increase of n, min_sup proportionally increases. Also note that in Fig. 9d, the runtime does not show a clear trend when varying m, which is consistent with our complexity analysis. From Figs. 9c and 9d, we can see that KiWi scales nicely with respect to n and m.

7 Conclusion

OPSMs have been accepted as a biologically meaningful pattern-based subspace cluster model, capturing the general tendency of gene expression across a subset of experiments. Biologists are particularly interested in OPSMs consisting of a small number of genes sharing expression patterns over many experiments. However, such OPSMs are often *deep* in massive data sets in the sense that existing exact methods fail to discover them due to the explosive computational costs.

In this paper, we have proposed a novel best effort mining framework KiWi for the deep OPSM discovery problem. KiWi exploits two parameters k and w to bound the available computational resources and search a selected promising search space, and does what it can to find as many as possible deep OPSMs. We have performed extensive biological and computational experiments on real data sets that demonstrated the meaningfulness of deep OPSMs, and the effectiveness and efficiency of KiWi in discovering deep OPSMs.

The deep OPSM problem we have identified and studied can play an important role in inferring gene regulatory networks and is of essential biological significance. Beyond that, our study can have a broader influence in pattern mining in general. The deep OPSM problem, in a broader sense, is a deep pattern problem, where the principles and ideas in our solution are directly applicable. Our attempt to estimate statistical significance of OPSMs has made initial contributions to estimation of statistical significance of patterns, a fundamental problem in pattern mining [22] that has been largely overlooked.

Appendix

Maximum Length of Sections

Here, we derive the cdf of the maximum length of sections of sequences segmented by a pattern. The cdf theoretically

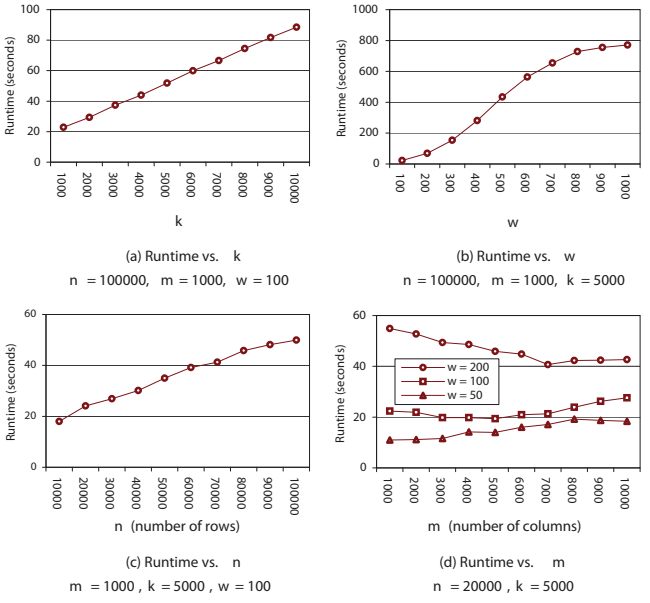


Fig. 9. Runtime.

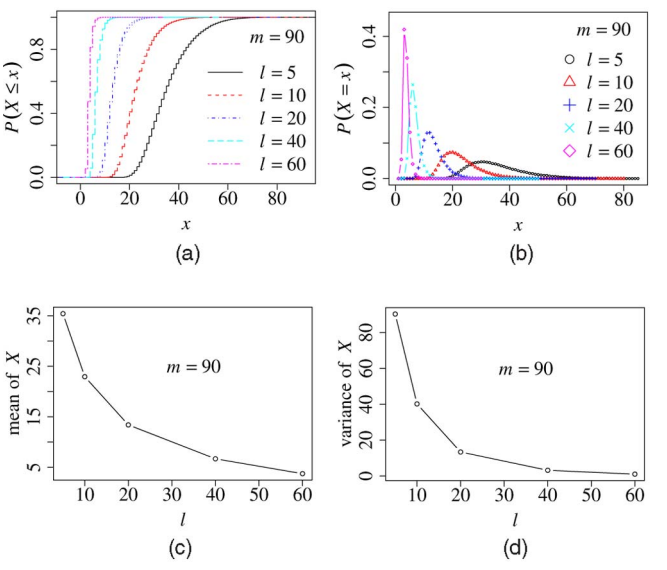


Fig. 10. Distributions and properties of X: (a) cdf, (b) pmf, (c) mean, and (d) variance.

justifies Observation 2 in Section 4.1. It is also used to determine the parameter w in Section 5.1.

Suppose there is a sequence of length m consisting of distinct elements. Then, any pattern of length $l, 1 \leq l \leq m$, separates the sequence into $P = l + 1$ sections with length $r_1, r_2, \ldots, r_P$, respectively. Define the total length $N = \sum_{i=1}^{P} r_i$. Apparently, $N = m - l$. Let X be the maximum length of all sections produced by a random pattern of length l, i.e., $X = max\{r_1, r_2, \ldots, r_P\}$.

Example 7. Pattern bc segments sequence $abcdef$ into three sections: a, '', and def with lengths 1, 0, and 3. In this case, $N = 4, P = 3$, and $X = max\{1, 0, 3\} = 3$. Another pattern of length 2 may result in a different set of section lengths such as $\{0, 1, 3\}$ (for pattern ac) or $\{1, 2, 1\}$ (for pattern be).

There are $\binom{6}{2} = 15$ different sets of section lengths in total for all patterns of length 2 and $\binom{m}{l}$ for all patterns of length l in general, where $\binom{m}{l} = \frac{m!}{l!(m-l)!}$. In fact, these distinguished sets of section lengths are formed by permuting the components of each of the partitions of N that have at most P elements. If the number of elements of a partition is less than P, zero(s) will be added before permuting it to form compositions.

Example 8. Integer $N = 4$ has five partitions: $\{4\}, \{3, 1\}, \{2, 2\}, \{2, 1, 1\}, \{1, 1, 1, 1\}$. When $P = 3$, only the first four partitions will be utilized, and $\binom{3+4-1}{4} = 15$ different compositions can be formed by permuting partitions $\{4, 0, 0\}, \{3, 1, 0\}, \{2, 2, 0\}$, and $\{2, 1, 1\}$.

Obviously, the maximum section length, X, takes distinct elements formed by the largest component of each partition of N utilized as it values. The probability mass function of X is given by

$$P(X = x) = \frac{\sum_x \binom{P}{m_1^{(x)} \cdots m_d^{(x)}}}{\binom{m}{l}},$$

where $\sum_x$ sums over partitions with the maximum component equal to x, d represents the number of distinct components of such a partition including zero, $m_1^{(x)}, \ldots, m_d^{(x)}$ denote the counts of the corresponding distinct components, and $\binom{P}{m_1 \cdots m_d} = \frac{P!}{m_1! \cdots m_d!}$.

Example 9. For the above example, X takes values 2, 3, and 4 based on the four partitions used, $P(X = 4) = \binom{3}{1\,2}/\binom{6}{2} = 3/15$ corresponding to permutations of $\{4, 0, 0\}$, $P(X = 3) = \binom{3}{1\,1\,1}/\binom{6}{2} = 6/15$ corresponding to permutations of $\{3, 1, 0\}$, and $P(X = 2) = [\binom{3}{2\,1} + \binom{3}{1\,2}]/\binom{6}{2} = 6/15$ corresponding to permutations of $\{2, 2, 0\}$ and $\{2, 1, 1\}$.

Thus, the cdf for the random variable X is

$$P(X \leq x) = \sum_{t \leq x} P(X = t).$$

An R program was written to compute the cdf of X for given m and l values. Using the program, the distributions of X were obtained for $m = 90$ and $l =$ 5, 10, 20, 40, and 60. The means and variances of X were also computed. The results are summarized in Fig. 10.

Figs. 10a and 10c reveal the relationship between X and l that conforms to Observation 2. In Fig. 10c, the mean of X decreases as l increases. Fig. 10b shows that X takes values $1, \ldots, m - l$ in general. The distributions of X are right skewed but become more symmetric as l decreases. According to Fig. 10d, the variance of X decreases too as l increases.

ACKNOWLEDGMENTS

A preliminary version of this paper was published in the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) [17].

REFERENCES

[1] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J.S. Park, "Fast Algorithms for Projected Clustering," *SIGMOD Record,* vol. 28, no. 2, pp. 61-72, 1999.

[2] C.C. Aggarwal and P.S. Yu, "Finding Generalized Projected Clusters in High Dimensional Spaces," *SIGMOD Record,* vol. 29, no. 2, pp. 70-81, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," *SIGMOD Record,* vol. 27, no. 2, pp. 94-105, 1998.

[4] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int'l Conf. Data Eng. (ICDE),* 1995.

[5] R. Albert, "Scale-Free Networks in Cell Biology," *J. Cell Science,* vol. 118, pp. 4947-4957, 2005.

[6] M. Ashburner et al., "Gene Ontology: Tool for the Unification of Biology, the Gene Ontology Consortium," *Nature Genetics,* vol. 25, no. 1, pp. 25-29, 2000.

[7] T. Barrett et al., "NCBI GEO: Archive for High-Throughput Functional Genomic Data," *Nucleic Acids Research,* vol. 37, pp. D885-D890, 2009.

[8] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, "Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem," *J. Computational Biology,* vol. 10, nos. 3/4, pp. 373-384, 2003.

[9] Y. Cheng and G.M. Church, "Biclustering of Expression Data," *Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology (ISMB),* 2000.

[10] L. Cheung, D.W. Cheung, B. Kao, K.Y. Yip, and M.K. Ng, "On Mining Micro-Array Data by Order-Preserving Submatrix," *Int'l J. Bioinformatics Research and Applications,* vol. 3, no. 1, pp. 42-64, 2007.

[11] W.G. Cochran, *Sampling Techniques,* third ed. John Wiley and Sons, 1977.

[12] A.C. Davison and D.V. Hinkley, *Bootstrap Methods and Their Application.* Cambridge Univ. Press, 1997.

[13] I. Dhillon, E. Marcotte, and U. Roshan, "Diametrical Clustering for Identifying Anti-Correlated Gene Clusters," *Bioinformatics,* vol. 13, no. 19, pp. 1612-1619, 2003.

[14] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *Annals of Statistics,* vol. 7, pp. 1-26, 1979.

[15] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD),* 1996.

[16] J.H. Friedman and J.J. Meulman, "Clustering Objects on Subsets of Attributes," *J. Royal Statistical Soc.,* vol. 66, no. 4, pp. 815-849, 2004.

[17] B.J. Gao, O.L. Griffith, M. Ester, and S.J.M. Jones, "Discovering Significant OPSM Subspace Clusters in Massive Gene Expression Data," *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD),* 2006.

[18] R. Gentleman et al., "Bioconductor: Open Software Development for Computational Biology and Bioinformatics," *Genome Biology,* vol. 5, no. 10, p. R80, 2004.

[19] S. Goil, H. Nagesh, and A. Choudhary, "MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets," Technical Report CPDC-TR-9906-010, Northwestern Univ., 1999.

[20] A. Goldstrohm, A. Greenleaf, and M. Garcia-Blanco, "Co-Transcriptional Splicing of Pre-Messenger RNAs: Considerations for the Mechanism of Alternative Splicing," *Gene,* vol. 277, nos. 1/2, pp. 31-47, 2001.

[21] O. Griffith et al., "Assessment and Integration of Publicly Available SAGE, cDNA Microarray, and Oligonucleotide Microarray Expression Data for Global Coexpression Analyses," *Genomics,* vol. 86, pp. 476-488, 2005.

[22] S. Hanhijärvi, M. Ojala, N. Vuokko, K. Puolamäki, N. Tatti, and H. Mannila, "Tell Me Something I Don't Know: Randomization Strategies for Iterative Data Mining," *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD),* 2009.

[23] J. Hartigan, "Direct Clustering of a Data Matrix," *J. Am. Statistical Assoc.,* vol. 67, no. 337 pp. 123-129, 1972.

[24] I. Hedenfalk et al., "Gene-Expression Profiles in Hereditary Breast Cancer," *New England J. Medicine,* vol. 344, no. 8, pp. 539-548, 2001.

[25] S. Ho Sui et al., "oPOSSUM: Identification of Over-Represented Transcription Factor Binding Sites in Co-Expressed Genes," *Nucleic Acids Research,* vol. 33, no. 10, pp. 3154-3164, 2005.

[26] J. Hubble et al., "Implementation of Genepattern within the Stanford Microarray Database," *Nucleic Acids Research,* vol. 37, pp. D898-D901, 2009.

[27] L. Jensen et al., "Arrayprospector: A Web Resource of Functional Associations Inferred from Microarray Expression Data," *Nucleic Acids Research,* vol. 32, pp. W445-W448, 2004.

[28] L. Jing, M.K. Ng, and J.Z. Huang, "An Entropy Weighting K-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 19, no. 8, pp. 1026-1041, Aug. 2007.

[29] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analyis.* John Wiley and Sons, 1990.

[30] H.-P. Kriegel, P. Kroger, M. Renz, and S. Wurst, "A Generic Framework for Efficient Subspace Clustering of High-Dimensional Data," *Proc. IEEE Fifth Int'l Conf. Data Mining (ICDM),* 2005.

[31] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering," *ACM Trans. Knowledge Discovery Data,* vol. 3, no. 1, pp. 1-58, 2009.

[32] J. Liu and W. Wang, "OP-Cluster: Clustering by Tendency in High Dimensional Space," *Proc. IEEE Third Int'l Conf. Data Mining (ICDM),* 2003.

[33] J. Liu, J. Yang, and W. Wang, "Biclustering in Gene Expression Data by Tendency," *Proc. IEEE Computational Systems Bioinformatics Conf. (CSB),* 2004.

[34] J.B. MacQueen, "Some Methods for Classifiation and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp. Math. Statistics and Probability,* 1967.

[35] S. Madeira and A. Oliveira, "Biclustering Algorithms for Biological Data Analysis: A Survey," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 1, no. 1, pp. 24-45, Jan.-Mar. 2004.

[36] T. Mielikainen and H. Mannila, "The Pattern Ordering Problem," *Proc. Seventh European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD),* 2003.

[37] B. Mirkin, *Mathematical Classification and Clustering.* Kluwer Academic Publishers, 1996.

[38] B. Modrek and C. Lee, "A Genomic View of Alternative Splicing," *Nature Genetics,* vol. 30, no. 1, pp. 13-19, 2002.

[39] R.T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB),* 1994.

[40] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The Prefixspan Approach," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 11, pp. 1424-1440, Nov. 2004.

[41] A. Prelic et al., "A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data," *Bioinformatics,* vol. 22, no. 9, pp. 1122-1129, 2006.

[42] C.M. Procopiuc, M. Jones, P.K. Agarwal, and T.M. Murali, "A Monte Carlo Algorithm for Fast Projective Clustering," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2002.

[43] J. Qian et al., "Beyond Synexpression Relationships: Local Clustering of Time-Shifted and Inverted Gene Expression Profiles Identifies New, Biologically Relevant Interactions," *J. Molecular Biology,* vol. 314, no. 5, pp. 1053-1066, 2001.

[44] C.P. Robert and G. Casella, *Monte Carlo Statistical Methods.* Springer, 2004.

[45] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT),* 1996.

[46] A. Su et al., "Large-Scale Analysis of the Human and Mouse Transcriptomes," *Proc. Nat'l Academy of Sciences USA,* vol. 99, no. 7, pp. 4465-4470, 2002.

[47] H. Wang, W. Wang, J. Yang, and P.S. Yu, "Clustering by Pattern Similarity in Large Data Sets," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2002.

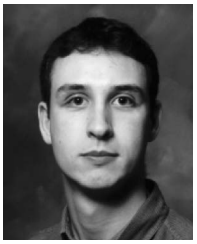
[48] K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee, "Findit: A Fast and Intelligent Subspace Clustering Algorithm Using Dimension Voting," *Information and Software Technology,* vol. 46, no. 4, pp. 255-271, 2004.

[49] J. Yang, W. Wang, H. Wang, and P.S. Yu, "δ-Clusters: Capturing Subspace Correlation in a Large Data Set," *Proc. 18th Int'l Conf. Data Eng. (ICDE),* 2002.

[50] K.Y. Yip, D.W. Cheung, and M.K. Ng, "Harp: A Practical Projected Clustering Algorithm," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 11 pp. 1387-1397, Nov. 2004.

[51] M.L. Yiu and N. Mamoulis, "Iterative Projected Clustering by Subspace Mining," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 2, pp. 176-189, Feb. 2005.
[52] M.J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning*, vol. 42, nos. 1/2, pp. 31-60, 2001.
[53] B. Zeeberg et al., "High-Throughput GoMiner, an 'Industrial-Strength' Integrative Gene Ontology Tool for Interpretation of Multiple-Microarray Experiments, with Application to Studies of Common Variable Immune Deficiency (CVID)," *BMC Bioinformatics*, vol. 6, article 168, 2005.
[54] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An Efficient Data Clustering Method for Very Large Databases," *SIGMOD Record*, vol. 25, no. 2, pp. 103-114, 1996.
[55] Y. Zhao, J. Xu Yu, G. Wang, L. Chen, B. Wang, and G. Yu, "Maximal Subspace Coregulated Gene Clustering," *IEEE Trans. Knowledge and Data Eng.*, vol. 20, no. 1, pp. 83-98, Jan. 2008.

Byron J. Gao received the PhD and BSc degrees in computer science from Simon Fraser University, Canada, in 2007 and 2003, respectively. He spent one and a half years as a postdoctoral fellow at the Database Lab of University of Wisconsin-Madison. In 2008, he joined Texas State University, San Marcos, as an assistant professor. His research spans several related fields including data mining, databases, information retrieval, and bioinformatics.



Obi L. Griffith received the PhD degree in medical genetics from the University of British Columbia, Canada, in 2008. He was a postdoctoral fellow at the Michael Smith Genome Sciences Center, BC Cancer Research Center until December 2009. Currently, he is working as a postdoctoral fellow at the Lawrence Berkeley National Laboratory. His research interests include bioinformatics, cancer informatics, gene expression and regulation, biomarkers, biostatistics, next-generation sequencing analysis, community-driven genome annotation, clustering, and cancer therapeutics.

Martin Ester received the PhD degree in computer science from the Swiss Federal Institute of Technology in 1989. In 1993, he joined the University of Munich as an assistant professor and cofounded their Data Mining lab. He joined the School of Computing Science of Simon Fraser University as an associate professor in 2001 and became a full professor in 2008. He has published extensively in leading conferences and journals of his field. His publications have been very well cited, and his H-number is currently 29. His most well-known paper on the density-based clustering algorithm DBSCAN has been cited more than 2,300 times. His current research interests include clustering, multi-relational data mining, graph mining, social network analysis, bioinformatics, and information extraction.

Hui Xiong received the PhD degree from the University of Minnesota. Currently, he is working as an associate professor in the Management Science and Information Systems Department at Rutgers University. His general area of research is data and knowledge engineering, with a focus on developing effective and efficient data analysis techniques for emerging data intensive applications. He has published more than 70 technical papers in peer-reviewed journals and conference proceedings. He is a coeditor of *Clustering and Information Retrieval* (Kluwer Academic Publishers, 2003) and a coeditor-in-chief of *Encyclopedia of GIS* (Springer, 2008). He is an associate editor of the *Knowledge and Information Systems journal*. He is a senior member of the IEEE, and a member of the ACM.

Qiang Zhao received the PhD degree in statistics from the University of Missouri, Columbia, in 2004. He worked at the University of Texas-Pan American as an assistant professor from 2004 to 2006 before joining the Department of Mathematics at Texas State University, San Marcos. His research interests include survival analysis, nonparametric statistics, computational statistics, and bioinformatics.

Steven J. M. Jones received the PhD degree at the Sanger Institute where he was involved in the *C. elegans* genome project. Currently, he is workinga as an associate director and head of bioinformatics at Canada's Michael Smith Genome Sciences Center, Vancouver, BC. He has established himself as one of Canada's brightest young scientists, being honored in 2006 with the Top 40 Under 40 Award by Caldwell Partners International. He also received in 2006 the Senior Early Career Scholar Award by Peter Wall Institute for Advanced Studies, the Spencer Award for IT Innovation from the University of British Columbia as well as the President's 40th Anniversary Award from Simon Fraser University. He has been an author on more than 150 peer reviewed publications. He has been Principal Investigator and coapplicant on numerous awarded grants totaling over $20 million.