Discovering Significant OPSM Subspace Clusters in Massive Gene Expression Data

Byron J. Gao[†], Obi L.Griffith[‡], Martin Ester[†], and Steven J.M. Jones[‡]
[†] School of Computing Science, Simon Fraser University, Canada
[‡] Genome Sciences Centre, British Columbia Cancer Agency, Canada
bgao@cs.sfu.ca, obig@bcgsc.ca, ester@cs.sfu.ca, sjones@bcgsc.ca

ABSTRACT

Order-preserving submatrixes (OPSMs) have been accepted as a biologically meaningful subspace cluster model, capturing the general tendency of gene expressions across a subset of conditions. In an OPSM, the expression levels of all genes induce the same linear ordering of the conditions. OPSM mining is reducible to a special case of the sequential pattern mining problem, in which a pattern and its supporting sequences uniquely specify an OPSM cluster. Those small *twig clusters*, specified by long patterns with naturally low support, incur explosive computational costs and would be completely pruned off by most existing methods for massive datasets containing thousands of conditions and hundreds of thousands of genes, which are common in today's gene expression analysis. However, it is in particular interest of biologists to reveal such small groups of genes that are tightly coregulated under many conditions, and some pathways or processes might require only two genes to act in concert. In this paper, we introduce the *KiWi* mining framework for massive datasets, that exploits two parameters k and w to provide a biased testing on a bounded number of candidates, substantially reducing the search space and problem scale, targeting on highly promising seeds that lead to significant clusters and twig clusters. Extensive biological and computational evaluations on real datasets demonstrate that *KiWi* can effectively mine biologically meaningful OPSM subspace clusters with good efficiency and scalability.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Data Mining; J.3 [Life and Medical Sciences]: Biology and Genetics

General Terms: algorithms, performance

Keywords: twig cluster, order-preserving submatrix, subspace clustering, gene expression data, scalability

1. INTRODUCTION

As an incomparable breakthrough in experimental molecular biology, DNA microarrays, serial analysis of gene expression (SAGE) and similar technologies have enabled a wave of extraordinary genomewide investigations of gene expression, allowing the monitoring of activities of many genes over many different conditions. The resulting expression data can be viewed as an $n \times m$ matrix with n genes (rows) and m conditions (columns), in which each entry gives the expression level of a given gene under a given condition.

Clustering is a major tool for gene expression analysis. Coexpression of genes in a cluster can be used to infer functional associations between genes and identify coregulation. Since most genes are expected to be tightly coregulated only under certain conditions, subspace clustering has gained popularity in recent years. Pattern-based subspace clustering, where clustering is performed by pattern similarity rather than distance, is particularly meaningful due to the fact that coregulated genes are not necessarily expressed at the same (or even similar) absolute expression levels. For example, a transcription factor may be able to perform its function at a very different concentration from its target genes. Recently, *order-preserving submatrixes* (OPSMs) [6] have been introduced and accepted as a biologically meaningful pattern-based subspace cluster model. An OPSM, essentially a 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 Figure 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. Discovery of significant OPSMs can play an essential role in inferring gene regulatory networks.

The OPSM cluster model focuses on the relative order of columns rather than the uniformity of actual values in data matrixes. 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 unique 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 cluster, with all the supporting sequences as the cluster contents. The number of supporting sequences is the *support* for the pattern.

As costs of gene expression analysis continue to decrease, the numbers and sizes of expression datasets have been growing at an ever-increasing rate. To give just two examples, the Gene Expression Omnibus (GEO) currently records over 70,000 conditions for over 100 different organisms [5] and the Stanford Microarray Database (SMD) contains over 10,000

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.
Copyright 2006 ACM 1-59593-339-5/06/0008 ...$5.00.

public experiments for 21 organisms [15]. With expression datasets of potentially tens or hundreds of thousands of both rows and columns, there is a need for algorithms that can handle not only "large datasets" but "massive datasets".

Protein-protein interactions, biological pathway membership and coregulation demonstrate "power law" relationships [3]. That is to say, we tend to observe a small number of very large gene groups and a large number of very small groups. For gene coregulation, the smaller groups, roughly under 50 genes, are of particular interest since in many cases biologists are looking for tissue or condition specific gene regulation related to some highly specialized process or disease. Some pathways or processes may require only two genes to act in concert. In the context of OPSM discovery, we use the term *twig clusters* to denote those small clusters specified by long patterns with naturally low support, corresponding to the "twigs" in a pattern search tree. Discovery of twig clusters is of essential biological importance.

Most existing sequential pattern mining methods, breadth-first or depth-first, aim at finding a complete set of patterns and rely on some minimum support threshold, *min_sup*, to prune the search space. However, the discovery of twig clusters requires the smallest possible *min_sup* of 2, making the pruning futile. The challenge becomes even more onerous in OPSM mining due to the fact that the transformed sequence database is extremely dense. Therefore, existing methods do not scale to massive gene expression datasets for small *min_sup* values, failing to produce any twig clusters.

To address these challenges, we propose the mining framework *KiWi* that exploits two parameters k and w to perform a biased testing on a bounded number of candidate patterns, targeting significant OPSMs with a focus on twig clusters. *KiWi* performs a beam search, at each level maintaining only the k most promising patterns as seeds for extension in further levels. When testing candidate patterns, *KiWi* considers only a vertical slice of width w of the supporting sequences, instead of the entire suffixes, ensuring the k spots are reserved for seeds that are likely to be extended into longer patterns. For ranking statistic, *KiWi* uses weighted support, taking the distribution of data into consideration.

Our main contributions are as follows:

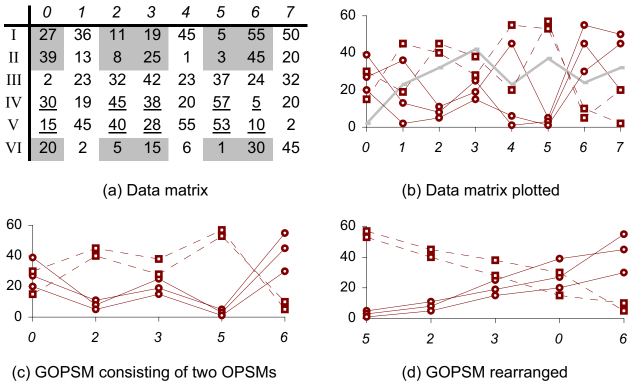
1. We introduce the *KiWi* framework for mining significant OPSM subspace clusters in massive gene expression datasets. The framework is also applicable to mining sequential patterns from dense, massive datasets in general.

2. We present a corresponding algorithm that implements the *KiWi* framework, tailored to the OPSM problem, providing candidate testing optimizations, memory management and seed management.

3. As an add-on, *KiWi* can also discover generalized OPSMs where the expression levels of genes induce either the same or opposite linear ordering of the conditions, capturing anti-correlations among genes as well.

4. Our comprehensive experimental study on real datasets demonstrates that *KiWi* can find numerous biologically meaningful twig clusters that existing methods fail to; in addition, *KiWi* scales nicely to massive datasets.

The rest of the paper is organized as follows. Section 2 introduces the objective of the paper and reviews related work. Section 3 proposes the *KiWi* mining framework and presents the main algorithm. Section 4 reports the experimental results, and Section 5 concludes the paper.



	0	1	2	3	4	5	6	7
I	27	36	11	19	45	5	55	50
II	39	13	8	25	1	3	45	20
III	2	23	32	42	23	37	24	32
IV	30	19	45	38	20	57	5	20
V	15	45	40	28	55	53	10	2
VI	20	2	5	15	6	1	30	45

(a) Data matrix

(b) Data matrix plotted

(c) GOPSM consisting of two OPSMs

(d) GOPSM rearranged

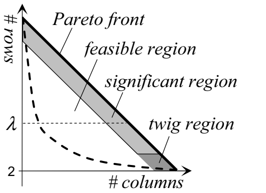
Figure 1: OPSM and GOPSM.

The raw 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.

2. OBJECTIVE AND RELATED WORK

Gene expression data can be represented as an $n \times m$ matrix D containing expression levels for n genes (rows) under m conditions (columns). A submatrix $S = (R \times C)$, where R is a subset of rows and C a subset of columns in D, is called an *order-preserving submatrix* (OPSM) [6] 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. If the constraint is relaxed such that the sequence can be either strictly ascending or descending, S becomes a *generalized order-preserving submatrix* (GOPSM). GOPSMs are able to capture anti-correlations among genes, which can imply common process/pathway membership or negative regulation. One gene may repress the expression of other genes (negative regulators) and anti-correlated genes might represent members of opposing pathways. Figure 1 illustrates the concepts of OPSM and GOPSM.

As motivated in the introduction, we want to find significant OPSMs in massive gene expression data. While various significance measures can be defined, it is consensus that for fixed # columns (# rows), larger # rows (# columns) leads to more significance. Rather than combining them into a single measure, we consider the competing goals of maximizing # columns and # rows in a bicriteria optimization problem. The feasible region of such a bicriteria problem, as shown in Figure 2, contains all feasible solutions, each being a (# rows, # columns) pair such that there exists such an OPSM in the given data matrix. A feasible pair can map to multiple OPSMs. The *significant region* contains partially ordered feasible solutions that are close to the Pareto optimums. While the closeness can be defined in various ways, one straightforward approach is to threshold # rows for each # columns, such that there are no more than t number of OPSMs with their # rows $\geq$ the threshold. The *twig region* is part of the significant region that is deepest and hardest to reach, containing the longest terminals of the search tree (space). Note that the search space for pattern mining can generally be organized in a tree structure, e.g., Rymon's generic set enumeration tree [14] for frequent



Pareto front: the monotonically descending line containing all non-dominated solutions.

feasible region: the region under (including) *Pareto front* containing all feasible solutions.

significant region: the shaded region under (including) *Pareto front* in *feasible region*.

twig region: the dark shaded region at the bottom of *significant region*.

Figure 2: Significant region and twig region.

pattern mining. In this paper, we use *significant clusters* and *twig clusters* to denote OPSMs having their (# rows, # columns) pairs falling in the corresponding regions.

In Figure 2, λ is an imaginary lower bound of *min_sup*, reflecting the minimum *min_sup* for which the complete set of frequent patterns can be efficiently mined using existing algorithms. It may move down with the growth of computational power, and move up due to the ever-increasing complexity of data. For dense and massive datasets, a large part of the significant region is normally pruned off, and the complete twig region is always in the prunings. Note that the Pareto front in Figure 2 serves only for the purpose of explanation. The dashed curve is a more realistic illustration of the shape of the Pareto front, showing more clearly how severe the pruning damage can be. In our experiment, *PrefixSpan* [13], one of the fastest sequential mining algorithms, did not return after 48 hours on real dataset human-SMD-cDNA (12671 genes and 2852 conditions) [15] for *min_sup* = 2535. It returned after 36 hours on Affymetrix (12332 genes and 1640 conditions) [5] for *min_sup* = 1234 with the maximum pattern length of 7. If we consider $\lambda = 1234$, then it would cross the Pareto front at # *columns* = 7, while the longest pattern with support ≥ 2 has length of 161.

The objective of the paper, is to discover significant OPSM subspace clusters from massive gene expression data, with a focus on twig clusters with size as small as 2.

2.1 Relationship to Sequential Pattern Mining

Conventional sequential pattern mining was motivated and introduced [2] in the context of transaction databases, where a sequence is an ordered list of itemsets. A common subsequence with support beyond the minimum support threshold, *min_sup*, is called a sequential pattern. The sequential pattern mining problem is to find the complete set of 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 Figure 3 (b). Each sequential pattern uniquely specifies an OPSM cluster, with all the *supporting sequences* (sequences containing the pattern) as the cluster contents. The number of supporting sequences is the *support* for the pattern, i.e., the cluster size.

Comparing to conventional sequential pattern mining, the OPSM problem 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 the sequences; or, exactly once in the absence of missing values. Third, even if with a small alphabet, sequences can be long, easily containing thousands of items (conditions). These properties imply that the transformed sequence database in the OPSM problem is extremely dense and high-dimensional.

In principle, with an additional scan to obtain the supporting sequences for each pattern, sequential pattern mining methods are applicable to OPSM discovery. However, the dense and high dimensional nature of the transformed sequence database renders *min_sup*-based methods infeasible for massive datasets with low threshold settings.

2.2 Related Work

Subspace clustering (e.g., [1]) aims at discovering clusters embedded in subspaces. Measuring similarity by pattern rather than distance, pattern-based subspace clustering has proved to be particularly meaningful in the application of gene expression analysis. Biclustering [7] discovers local coherence of genes and conditions in a submatrix of a DNA array. In the biological sense, genes in such clusters have the same amount of response to the conditions. δ-pCluster [18] models 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 conditions.

The δ-pCluster model can be too restrictive in many applications. As a relaxation, *order-preserving submatrixes* (OPSMs) [6] are essentially subspace clusters in which the expression levels of all genes induce the same linear ordering of the conditions. As the OPSM problem is NP-hard, a model-based algorithm is given in [6] to find the "best" OPSM according to a hard-coded statistical measure. Scalability issues are not considered. The proposed algorithm requires excessive computational resources if applied to large gene expression matrixes. OP-clustering [12] generalizes the OPSM model by grouping attributes into equivalent classes. Their proposed method, OPC-tree, performs exhaustive enumeration as in conventional sequential pattern mining, and fails for the same reason to obtain most significant subspace clusters from massive datasets.

Sequential pattern mining was introduced in [2]. Existing methods are essentially either breath-first such as *GSP* [16] or depth-first such as *PrefixSpan* [13]. *PrefixSpan* is known as one of the most efficient algorithms so far.

3. THE *KiWi* MINING FRAMEWORK

With the objective to discover significant OPSM subspace clusters including twig clusters in massive datasets, *KiWi* utilizes two parameters k and w to provide a biased testing on a bounded number of candidates, substantially reducing the search space and problem scale, targeting on highly promising seeds that are likely to lead to long patterns.

3.1 Principles

We use *pattern* to denote an ordered subset of column labels (pattern elements), e.g., $p^i = [p_1, p_2, ..., p_i]$ 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, and the k candidates with the highest values of this statistic are retained as the seeds of level i. The number of candidates is thus upper-bounded by $k \times m$ and the search space is selectively restricted. Based on the observation that a long pattern segments its supporting sequences into small sections, 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.

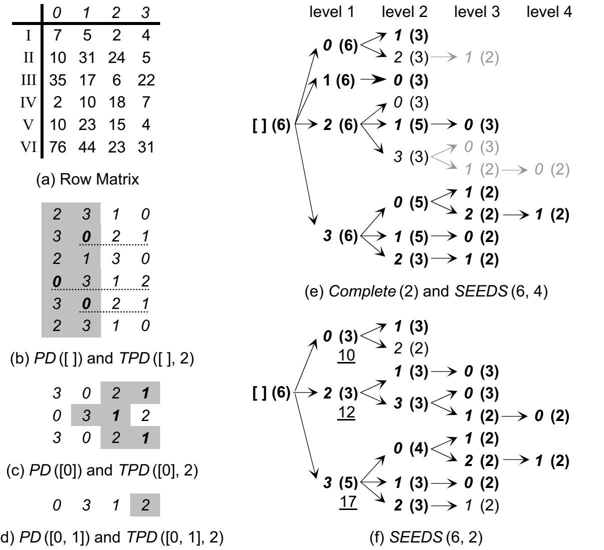
For each seed p^i, the *projected database* for p^i, $PD(p^i)$, is maintained containing the sequences supporting p^i under the *w-constraint*, that is, a sequence in $PD(p^i)$ is segmented by the elements of p^i into sections of length no more than w except for the last section. Rapidly shrinking, the 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 *truncated projected database* for p^i with respect to w, $TPD(p^i, w)$, contains some truncated regions of width at most w, each for a sequence in $PD(p^i)$ called $window(p^i, w)$ that immediately follows the appearance of p_i.

In testing candidate p^{i+1}, only $TPD(p^i, w)$ is considered, which does not only substantially improve efficiency, but is also necessary for long pattern discovery. Should we consider the entire suffixes of the sequences, some unpromising candidates that appear very late in the sequences would gain high ranks to take many of the k spots that are reserved for promising seeds, and the iteration would soon terminate and fail to grow long patterns.

To rank candidates, support seems to be an apparent and natural choice as the statistic. Experiments also show 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 thus cannot be well distinguished. We introduce the novel 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, $|suffix(p_i, r)|$, 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 Figure 3 (b). 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 constitute the output pattern set $SEEDS(k, w) \subseteq Complete(2)$, where $Complete(2)$ is the complete set of patterns with support of at least 2. Figure 3 (e) shows $Complete(2)$ in a tree structure. Once k and w are determined, $SEEDS(k, w)$ is determined. In $KiWi$, $p^l \in SEEDS(k, w) \Rightarrow p^i \in SEEDS(k, w)$ for $1 \leq i \leq l$, that is, for p^l to be discovered, each of its prefixes p^i has to appear among the top k seeds in the ith iteration.

Running example. In Figure 3, the raw data matrix with 4 conditions and 6 genes in (a) is transformed into a 4×6 sequence database as shown in (b). (b) $\sim$ (d) exemplify the *PDs* and *TPDs* (the shaded areas) along the growing process of pattern [0, 1] with respect to w = 2, from which we can see how the problem scale is greatly reduced level by level. (e) shows $Complete(2)$ and the actual support for each pattern. (e) 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 0 to 4, the non-gray-colored patterns are candidates and the dark-colored ones are eventually chosen as seeds. The results in (e) are reasonably good, but one longest pattern, $[2, 3, 1, 0]$ is missing. Level 2 of (e) also shows the ranking problem that, some candidates with support 3 are chosen as seeds, some not. (f) shows the mining process when w is adjusted to 2. The results in (f) are very good with both the longest patterns discovered, together with most of the more frequent patterns at each level. Comparing to (e), the full length 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.



(a) Row Matrix

(b) *PD* ([ ]) and *TPD* ([ ], 2)

(c) *PD* ([0]) and *TPD* ([0], 2)

(d) *PD* ([0, 1]) and *TPD* ([0, 1], 2)

(e) *Complete* (2) and *SEEDS* (6, 4)

(f) *SEEDS* (6, 2)

Figure 3: Running example.

When weighted support is used, w is insensitive. For w = 4, 3, or 2, exactly the same results are returned as in (f). The underlined numbers in (f) for level 1 show the weighted support values for the candidates, e.g., the weighted support for [0] is 10, which can be verified in (b). In (b), the three 0-*suffixes* for sequences supporting [0] are underlined. Note that only $TPD([\,], 2)$ is considered in testing candidate [0]. From (f) we can also see that, if weighted support is used, even for $k = 2$, the two longest patterns will still be returned.

More discussions and insights on the ranking statistics and choices of k and w are detailed in [8]. Existing depth-first methods such as *PrefixSpan* [13] cannot perform within-level comparisons to keep the most promising patterns to work on; thus most computations are wasted on recursively mining patterns that are eventually shown to be insignificant, despite their advantage of parsimonious memory usage. Most existing breath-first methods such as *GSP* [16] are inferior to *PrefixSpan* due to various reasons [13], the large memory consumption is a serious inherent problem. *KiWi*, however, can well-bound the memory usage and significantly reduce the search space and problem scale due to the introductions of k and w. Note that if k is big enough and w is set to m, *KiWi* returns the complete set of patterns.

3.2 Algorithm and Techniques

Algorithm 1 gives the pseudocode for the *KiWi* pattern discovery algorithm with certain low level details.

Overview. 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 $\{0, 1, ..., m-1\}$. The output pattern set *SEEDS* contains *seeds* from different levels, at most k for each level and organized as linked lists in order to save space. *seeds* is a list of seeds, each being a pattern and a list of elements conceptually (physically, an element in some linked list). [ ] denotes the empty list. *projections* is a list of projected databases, each for a seed in *seeds* with the same index, represented as a list of supporting sequence indexes. *mSeeds* is a list of integers used for statistic counting, which always has size of $m \times |seeds|$ since we preserve m positions for each seed. The data structures *seeds*, *mSeeds*, and *projections* are reused from iteration to iteration.

Initially, *seeds* contains one empty seed [ ] (line 1), and $PD([\,])$ contains all the n sequence indexes (line 2). As long as *seeds* is not empty (line 3), we continue to grow the seeds. In each iteration, we first perform candidate testing (line 4, 5, 6, 7, 8), then, *seeds* is updated with the (at most) k candidates having top values in *mSeeds* (line 9). The projected databases are updated as well for the seeds in *seeds* (line 10). Finally, *seeds* gets rid of the seeds with support less than 2 (line 11) and is written to disk (line 12). For efficiency, instead of recording the supporting sequence indexes for each candidate during testing, we update *projections* after the update of *seeds*, then only at most k seeds need to be processed and the update takes negligible time.

Candidate testing. Generally, support counting is the major time consumer in frequent (sequential) pattern mining. *KiWi* achieves great efficiency in statistic counting due to several reasons. First, since only the top k seeds are allowed to grow, the number of candidates is well bounded and the search space is selectively restricted. Second, the horizontal (line 6) and vertical (line 7) slicing of the sequence database dramatically reduce the problem scale from the entire database to a particular truncated projected database.

While these principles guarantee an efficient runtime complexity of candidate testing, technically, *KiWi* is also carefully designed to achieve the best efficiency within the complexity. m is not big comparing to conventional sequential pattern mining; thus, we preserve m positions in *mSeeds* for each seed since it has at most m ways to grow one more element. This fixed 1-to-m correspondence allows us to avoid explicit candidate generation. Note that the candidate generation and pruning time is not trivial in *GSP* like methods when the database is dense. In more details, let $seeds[s] + e$ be a candidate with e as the newly appended element, then, $s' = m \times s + e$, meaning that the statistic incrementing for the candidate will be done in $mSeeds[s']$ (line 8). Conversely, knowing s', an index in *mSeeds*, we have $e = s' \ mod \ m$ and $s = s' \ div \ m$, meaning that the statistic stored in $mSeeds[s']$ is for candidate $seeds[s]+e$. This converse derivation is used in line 9 to update *seeds*, growing the linked lists properly.

In Algorithm 1, lines 4 ~ 8 perform statistic counting. First, *mSeeds* is initialized with $m \times |seeds|$ number of 0's (line 4). Then, for each seed in *seeds* (line 5), for each element in the truncated projected database for the seed, (lines 6, 7), weighted support is incremented for the corresponding candidate (line 8), precisely, $seeds[s] + e$.

Algorithm 1 *KiWi* pattern discovery

```
Input: $n \times m$ sequence database $SB$, $k$, $w$
Output: $SEEDS(k, w)$
 1: $seeds$ = [[ ]];
 2: $projections$ = [[0, 1, ... n - 1]];
 3: while (|$seeds$| != 0) {
 4:    $mSeeds$.initialize($m \times |seeds|$, 0);
 5:    for each seed in $seeds$ with index $s$
 6:       for each sequence $r$ in $projections[s]$
 7:          for each element $e$ in $window(seeds[s], w)$ of $r$
 8:             $mSeeds[m \times s + e]$ += $|suffix(e, r)|$;
 9:    $seeds$.update($k$, $mSeeds$);
10:    $projections$.update($seeds$);
11:    $seeds$.cleanup();
12:    $seeds$.writeup(); }
```

Other techniques used by *KiWi* in pattern discovery, such as the use of inverse *SB* (*ISB*), memory management and seeds management are detailed in [8].

Complexities. The runtime of *KiWi* is dominated by candidate testing in lines 4 ~ 8 of Algorithm 1. By upper-bounding the number of visits of the *TPDs*, 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. For the average case analysis we assume a random data matrix. In such a matrix, each pattern of length i is on average supported by $n \times \frac{1}{i!}$ number of sequences. Knowing that $i! \geq 2^i$ for $i \geq 1$, the average case runtime is $O(kwn \times (\frac{1}{1!} + \frac{1}{2!} + ... + \frac{1}{l!})) = O(kwn \times (\frac{1}{2^0} + \frac{1}{2^1} + ... + \frac{1}{2^l})) = O(kwn \times 2) = O(kwn)$. 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 *mSeeds*, and kn for *projections* since each pattern can be supported by at most n sequences. The memory taken by *seeds* is $O(k)$ and negligible. It is reasonable to assume *SB* can be well accommodated in memory, and normally $n \gg m$, thus *projections* 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 $\min(k, m)$. Some memory management techniques are also employed to avoid memory spill caused by *projections*. Overall, the memory consumption in *KiWi* is well bounded.

OPSM formation and GOPSM mining. To actually form OPSM clusters, we need to retrieve the supporting sequences for the output patterns, which were not stored for space efficiency. We provide a pattern querying phase following pattern discovery, allowing users to specify interesting patterns, and only for which *SB* is scanned to form the corresponding OPSMs. A pattern extension heuristic is also applied for the patterns after obtaining their supporting sequences. Surprisingly, only some simple adjustment is sufficient to allow *KiWi* to mine GOPSMs, i.e., doubling *SB* by appending a list of reversely ordered sequences, or more efficiently, mining *SB* forward and backward simultaneously. [8] has more details regarding OPSM formation, pattern extension, and GOPSM mining.

4. EXPERIMENTAL EVALUATIONS

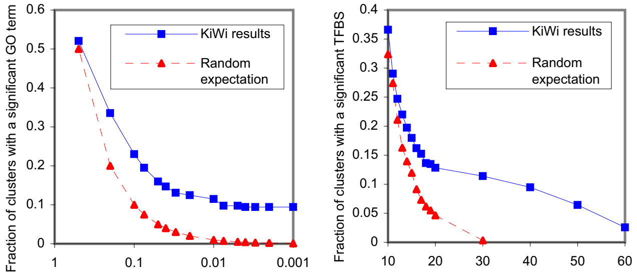
Comprehensive experiments were performed on massive real and synthetic datasets for biological and computational evaluations of *KiWi*. The real datasets included Affymetrix (HG-U133A) experiments from GEO (GPL96) [5], SAGE libraries also from GEO (GPL4), and cDNA experiments from SMD [15]. Affymetrix probes, cDNA clones and SAGE tags were normalized and mapped to gene/protein identifiers as previously reported [9]. For comparison with existing algorithms, two small datasets, SU [17], a smaller dataset of Affymatrix (HG-U95A) experiments, and breast cancer [10], were also used. Table 1 lists the sizes of these datasets. The experiments were run on a 2.8GHz/2GB Windows machine.

4.1 Biological Evaluation

Biologically meaningful groups of coregulated genes should tend to have common (statistically over-represented) biological processes and transcription factor binding sites (TFBS). In this series of experiments, we use two methods to demonstrate that *KiWi* is able to identify clusters from gene expression datasets of very large size, that are consistent with biological expectation for coregulated genes.

Methods. The first method uses the Gene Ontology (GO), a set of structured, controlled vocabularies to identify functional associations between gene products [4]. Current GO annotation and external reference files were downloaded from the Gene Ontology Annotation resource at EBI (www.ebi.ac.uk/GOA). Using these files, and the Ensembl Perl API (www.ensembl.org) the gene identifiers in the cluster data were mapped to protein identifiers compatible with GO. Each cluster of protein IDs was then submitted to the High-Throughput GoMiner command-line interface [19]. Statistically over-represented GO terms were defined using a Fisher's exact test and corrected for multiple testing by false discovery rate detection (100 permutations). The second method uses the oPOSSUM tool to identify statistically over-represented transcription factor binding sites (TFBS) [11]. The oPOSSUM API was downloaded and installed (www.cisreg.ca/cgi-bin/oPOSSUM/opossum). Each cluster of genes was submitted to the software and statistically over-represented TFBSs were defined using the Z-score option.

Results. Biological validation was performed on the affymetrix dataset with 12332 genes and 1640 conditions. For the validation, a set of representative clusters were chosen with size from 5 to 10 and minimum dimensionality of 15. A total of 634 clusters met these criteria, with 22 clusters containing anti-correlated genes. The GO analysis shows that clusters identified by *KiWi* are significantly more likely to share a common biological process than random expectation (Figure 4 (a)). For example, if we consider a P-value threshold of 0.01, more than 10% of clusters have at least one significant GO term compared to the random expectation of close to zero. Similarly, the TFBS analysis shows that clusters identified by *KiWi* are significantly more likely than random expectation to share sequences bound by the same transcription factor (Figure 4 (b)). For example, if we consider a Z-score of 30, more than 10% of clusters have at least one TFBS over-represented in the regulatory regions for these genes. Random expectation for this same Z-score threshold is close to zero. To summarize, *KiWi* successfully identifies small groups of correlated and anti-correlated genes that belong to a common function or process and/or share common transcription factor binding sites.



(a) P-value (FDR corrected, 100 permutations)

(b) Z-score for TFBS over-representation

Figure 4: Biological evaluation.

4.2 Computational Evaluation

In this series of experiments, we show that *KiWi* is able to effectively discover significant OPSM clusters including twig clusters using real datasets; we also demonstrate the efficiency and scalability of *KiWi* using synthetic data.

Discovery of twig clusters. To demonstrate *KiWi* can well reach the twig region, we compared *KiWi* with *LoPad*, a dynamic programming based algorithm that finds the (real) longest pattern with respect to *min_sup* of 2. Table 1 reports the longest pattern length and 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 dataset and the program was terminated.

Table 1: *KiWi* vs. *LoPad*: longest pattern discovery.

Dataset	# of genes	# of conditions	*LoPad*	Time (s)	*KiWi*	Time (s)
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
Affymetix	12332	1640	161	341476	160	1458
cDNA	12671	2852	NA	NA	228	1735

Discovery of significant clusters. To assess the statistical significance of an OPSM, [6] computes an upper bound on the probability that the corresponding model would appear in a random matrix, and their algorithm (referred to as *Alg*) tries to find the most significant OPSM one at a time. They reported several significant OPSMs with different pattern lengths on the breast cancer dataset. While they did not give the runtime, *KiWi*, in just 1.47 seconds, found much better results for every case they reported as shown in Table 2. In addition, since this is a very small dataset, we were able to set $min\_sup = 2$ and obtain the results from *PrefixSpan* in 2085 seconds (with output file size of 12 GB), from which we can see that our results are actually optimal.

We have defined significant clusters as OPSMs having their (# rows, # columns) pairs falling in the significant region. We also gave a quantitative definition of the significant region, i.e., to threshold # rows (support) for each # columns (pattern length), such that there are no more than t number of OPSMs with their # rows $\geq$ the threshold. To evaluate the coverage of significant clusters by *KiWi*, we used the small breast cancer dataset since it was the only one we obtained output from *PrefixSpan* for $min\_sup = 2$. For

t = 100 and 1000 (values inside the brackets) respectively, Table 3 shows the support range above the threshold, the number of significant clusters returned by *PrefixSpan* and *KiWi*, together with the coverage for each pattern length of 1 – 16. The *KiWi* results were obtained in 24 seconds, about 1.2% of *PrefixSpan*, but they cover on average 92.6% (t = 100) and 80.7% (t = 1000) of the significant clusters.

Table 2: *KiWi* vs. *Alg*: significant OPSM discovery.

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

Table 3: *KiWi* vs. *PrefixSpan*: coverage.

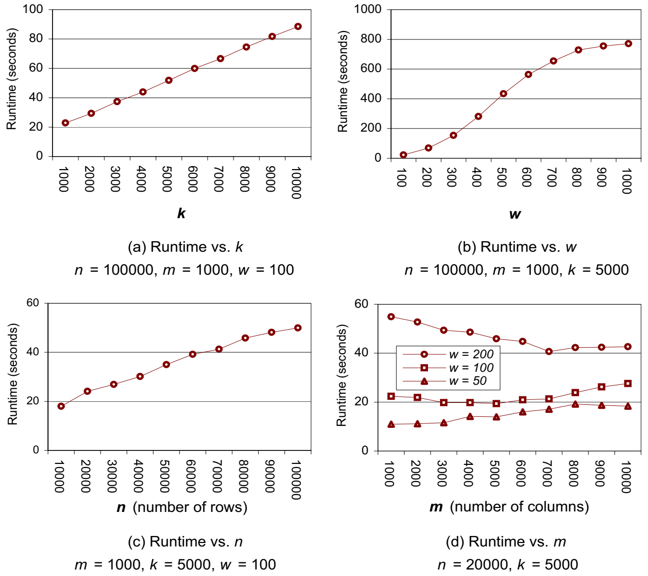
Pat. len.	Support range	Prefix-Span	*KiWi*	Coverage (%)
1	3226 – 3226 (3226)	22 (22)	22 (22)	100 (100)
2	2506 – 1909 (720)	100 (462)	100 (462)	100 (100)
3	1613 – 1136 (800)	98 (996)	98 (996)	100 (100)
4	795 – 571 (414)	99 (994)	99 (994)	100 (100)
5	337 – 244 (184)	99 (978)	99 (978)	100 (100)
6	129 – 94 (73)	100 (958)	100 (958)	100 (100)
7	50 – 35 (27)	77 (870)	77 (870)	100 (100)
8	19 – 14 (11)	63 (860)	63 (594)	100 (69.1)
9	9 – 7 (6)	75 (398)	54 (183)	72 (46)
10	6 – 5 (4)	12 (381)	12 (255)	100 (66.9)
11	4 – 4 (3)	3 (971)	3 (574)	100 (59.1)
12	3 – 3 (3)	38 (38)	25 (25)	65.8 (65.8)
13	3 – 3 (3)	1 (1)	1 (1)	100 (100)
14	2 – NA (2)	NA (723)	NA (243)	NA (33.6)
15	2 – 2 (2)	35 (35)	18 (18)	51.4 (51.4)
16	2 – 2 (2)	1 (1)	1 (1)	100 (100)

Efficiency and scalability. We used synthetic data (random matrixes of different sizes) for the runtime analysis of *KiWi*. The results shown in Figure 5 (a) – (c) confirm the average case complexity of $O(kwn)$. In (d), the runtime does not show a clear trend when varying m, which is consistent with our complexity analysis. (c) and (d) together show that *KiWi* scales nicely with respect to n and m.

5. CONCLUSIONS

Accepted as a biologically meaningful subspace cluster model, order-preserving submatrixes (OPSMs) capture the general tendency of gene expressions across a subset of conditions. Biologists are particularly interested in OPSMs consisting of a small number of genes sharing expression patterns over many conditions, which we call twig clusters. Most existing methods fail to discover those twig clusters in massive datasets due to the explosive computational costs. In this paper, we introduced the *KiWi* framework for mining significant OPSM subspace clusters in massive gene expression data, focusing on twig clusters. *KiWi* exploits two parameters k and w to perform a biased testing on a bounded number of candidates, keeping only highly promising seeds that will likely lead to significant clusters and twig clusters. Our extensive experiments on real and synthetic data demonstrated that *KiWi* scales well to massive datasets and can discover numerous biologically meaningful OPSM subspace clusters that cannot be mined using existing methods.

The concept of twig clusters and the *KiWi* framework are not limited to OPSM mining, but applicable to sequential pattern mining in general for dense and massive datasets.



(a) Runtime vs. k
n = 100000, m = 1000, w = 100

(b) Runtime vs. w
n = 100000, m = 1000, k = 5000

(c) Runtime vs. n
m = 1000, k = 5000, w = 100

(d) Runtime vs. m
n = 20000, k = 5000

Figure 5: Runtime.

6. REFERENCES

[1] R. Agrawal, et al., Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD*, 1998.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE*, 1995.

[3] R. Albert. Scale-free networks in cell biology. *Journal of Cell Science*, 118:4947–4957, 2005.

[4] M. Ashburner, et al., Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, 2000.

[5] T. Barrett, et al., NCBI GEO: mining millions of expression profiles–database and tools. *Nucleic Acids Res.*, 33:D562–D566, 2005.

[6] A. Ben-Dor, et al., Discovering local structure in gene expression data: The order-preserving submatrix problem. *Journal of Computational Biology*, 10(3-4):373–384, 2003.

[7] Y. Cheng and G. Church. Biclustering of expression data. In *ISMB*, 2000.

[8] B. Gao, et al., Discovering significant OPSM subspace clusters in massive gene expression data. *Technical Report, TR 2006–18, Simon Fraser University*, 2006.

[9] O. Griffith, et al., Assessment and integration of publicly available SAGE, cDNA microarray, and oligonucleotide microarray expression data for global coexpression analyses. *Genomics*, 86:476–488, 2005.

[10] I. Hedenfalk, et al., Gene-expression profiles in hereditary breast cancer. *NEJM*, 344:539–548, 2001.

[11] S. Ho Sui, et al., oPOSSUM: identification of over-represented transcription factor binding sites in coexpressed genes. *Nucleic Acids Res.*, 33(10):3154–64, 2005.

[12] J. Liu and W. Wang. OP-cluster: clustering by tendency in high dimensional space. *ICDM*, 2003.

[13] J. Pei, et al., Mining sequential patterns by pattern-growth: the PrefixSpan approach. *TKDE*, 16(10):1424–1440, 2004.

[14] R. Rymon. Search through systematic set enumeration. *KR*, pages 539–550, 1992.

[15] G. Sherlock, et al., The stanford microarray database. *Nucleic Acids Res.*, 29(1):152–155, 2001.

[16] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. *EDBT*, 1996.

[17] A. Su, et al., Large-scale analysis of the human and mouse transcriptomes. *PNAS*, 99(7):4465–4470, 2002.

[18] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets. *SIGMOD*, 2002.

[19] 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*, 6:168, 2005.