F2G: Efficient Discovery of Full-Patterns

Rui Henriques, Sara C. Madeira, and Cláudia Antunes

Dep. of Comp. Science and Eng. *and* KdBIO, Inesc-ID
Instituto Superior Técnico, University of Lisbon, Portugal
{rmch,sara.madeira,claudia.antunes}@ist.utl.pt

Abstract. An increasing number of biomedical tasks, such as pattern-based biclustering, require the disclosure of the transactions (e.g. genes) that support each pattern (e.g. expression profiles). The discovery of patterns with their supporting transactions, referred as *full-pattern mining*, has been solved recurring to extensions over Apriori and vertical-based algorithms for frequent itemset mining. Although pattern-growth alternatives are known to be more efficient across multiple biological datasets, there are not yet adaptations for the efficient delivery of full-patterns. In this paper, we propose a pattern-growth algorithm able to discover full-patterns with heightened efficiency and minimum memory overhead. Results confirm that for dense datasets or low support thresholds, a common requirement in biomedical settings, this method can achieve significant performance improvements against its peers.

1 Introduction

The discovery of frequent patterns is increasingly accomplished in biological settings to identify orchestrations of genes and of their products using exhaustive and efficient searches [17]. However, biological tasks often require the output of both patterns and their supporting transactions. This task is referred as *full-pattern mining*. Illustrating, in gene expression analysis the interest is not so centered on the frequent expression profiles, but mainly on the group of genes that support those expression profiles. Full-pattern mining has been adopted in the past for biclustering [14, 17], gene association analysis [2] and integrative genomic studies [11] using gene expression data and genomic structural variations.

Existing full-pattern miners are simple extensions of frequent itemset mining algorithms that rely on bitset vectors to represent the supporting transactions per pattern [13, 17, 14]. However, bitset vectors offer efficiency problems in terms of memory and time for biological datasets with a medium-to-high number of records. Thus, the goal of this work is to study efficient alternatives for the full-pattern mining task. In particular, we propose the first variant of the FP-growth method, referred as F2G (Frequent Full-pattern Growth), able to deliver full-patterns with heightened efficiency. Experimental results hold evidence for its superior performance. We also show that existing scalability principles and alternative pattern representations can be easily included in F2G.

The paper is structured as follows. In *section 2*, the full-pattern mining task is formalized and its relevance is motivated. In *section 3*, the contributions and limitations from existing research are covered. The proposed F2G algorithm is described in *section 4*. Finally, key implications from the evaluation of F2G against state-of-the-art alternatives are synthesized in *section 5*.

2 Background

Let $\mathcal{L}$ be a finite set of items, and I be an itemset $I \subseteq \mathcal{L}$. A *transaction* t is a pair (t_{id}, I) with $t_{id} \in \mathbb{N}$. An *itemset database* D over $\mathcal{L}$ is a finite set of transactions. A transaction $t = (t_{id}, I)$ contains an itemset A, denoted $A \subseteq t$, if $A \subseteq I$. The *coverage* Φ_I of an itemset I is the set of all transactions in D in which the itemset I is contained: $\Phi_I = \{t \in D \mid I \subseteq t\}$. The *support* of an itemset I in D, denoted sup_I, is its coverage size $|\Phi_I|$.

A *full-pattern* is a pair (I, Φ_I), where I is an itemset and Φ_I the set of all transactions that contain I.

Given an itemset database D and a minimum support threshold θ, the ***full-pattern mining*** task consists of computing the set: $\{(I, \Phi_I) \mid I \subseteq \mathcal{L}, sup_I \geq \theta\}$.

For an illustrative itemset database D=$\{(t_1, \{a, c, e\}), (t_2, \{a, b, d\}), (t_3, \{a, c\})\}$, we have $\Phi_{\{a,c\}}$=$\{t_1, t_3\}$, $sup_{\{a,c\}}$=2. For a minimum support θ=2, the full-pattern mining task over D returns $\{(\{a\}, \{t_1, t_2, t_3\}), (\{a, c\}, \{t_1, t_3\})\}$.

2.1 Applications

Biclustering, the discovery of correlations among transactions (e.g. genes) based on a subset of overall items (e.g. conditions), is an increasingly popular biological task [12]. Fig.1 illustrates how biclustering relies on full-patterns.

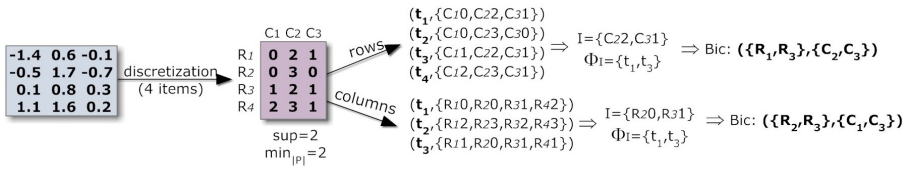

Fig.1: Mining constant biclusters over itemset databases using full-patterns.
The input dataset is discretized. To find biclusters with constant values on rows, the discrete value of each cell is concatenated with the respective column index. Each transaction corresponds to a row with these new values. Full-pattern mining is applied. The columns and rows for each bicluster are, respectively, derived from the items and supporting transactions of each full-pattern. Biclusters with constant values on columns can be mined using the transpose matrix.

Full-pattern mining methods to biclustering combine efficiency with flexibility [17]. This explains the increasing attention towards pattern-based approaches to biclustering such as BiModule [14], DeBi [17], RAP [16] and GenMiner [13].

Full-pattern mining has been also applied to perform association analysis using patient/gene/product-centric views and to study biological networks [2, 3]. These tasks have been applied over gene expression data, mutations and copy number variations [9], and discrete matrices derived from biological networks [3].

2.2 Related work

Separating the disclosure of transactions from the core mining task introduces a significant and unnecessary computational effort. Thus, the existing full-pattern miners rely on frequent itemset mining methods with implementations based on bitset vectors to represent the transaction-sets. Since these structures are maintained during all the search, is trivial to disclose the transactions shared per pattern at the end of the search. There are two approaches with this behavior.

A first approach combines the original Apriori-based strategy with the use of bitset vectors to track patterns coverage [1]. Illustrative implementations include LCM and CLOSE, used respectively by BiModule [14] and GenMiner [13] biclustering methods. They differ regarding the pruning methods applied over the itemset lattice of frequent candidates. Apriori-based methods generally suffer from the costs associated with the generation of a huge number of candidates for low support thresholds, a common requirement in biological tasks [14].

A second approach is the use of vertical-based methods, mostly through the extension of Eclat [19] and Carpenter [15]. Since vertical-based methods rely on intersection operations over transaction-sets to generate candidates, they require structures (such as bitset vectors or diffsets) to maintain the coverage per pattern. When the bitset cardinality becomes large, not only these structures consume a significant amount of memory, but also the intersection gets computationally costly. MAFIA [4] is an illustrative implementation adopted by DeBi [17]. However, these vertical-based methods are not competitive with horizontal-based methods for datasets with a number of transactions that significantly exceeds the average number of items [18].

2.3 The Problem

Existing full-pattern miners suffer from critical drawbacks. First, Apriori variants present efficiency problems for low support thresholds. Second, vertical-based approaches are not prone to deal with databases with a large set of transactions. Although FP-Growth method can overcome these problems [8], this method was not yet adapted to efficiently delivery full-patterns. A straightforward extension would be the annotation of each node on the underlying tree structure (FP-tree) with its supporting transactions, which undesirably leads to an impracticable additional computational complexity. Since most of the biological tasks rely on large and dense datasets and low support thresholds, the study of efficient FP-Growth extensions to discover full-patterns is of critical importance.

3 Solution

Since FP-Growth method relies on compact tree structures, it is positioned as an attractive alternative to tackle the computational overhead of the existing methods that maintain bitset vectors for every frequent candidate. Additionally, it neither requires candidate generation nor multiple database scans. In this section we propose a variant of the original FP-Growth method to deliver full-patterns with heightened efficiency. This algorithm is referred as *F2G*, *F*requent *F*ull-pattern *G*rowth. Similarly to the original FP-Growth method, F2G relies on a compact tree structure (FP-tree), which is recursively mined to enumerate all frequent patterns. Patterns are generated by concatenating the pattern suffixes with the frequent patterns discovered from conditional FP-trees where suffixes are removed. Suffixes are composed according to an ascending frequency order to prune the search space. In F2G, the transactions are optimally stored since they appear at most once in the FP-tree. Thus, unlikely the original method, the transaction-IDs are not lost at the very first scan.

Algorithm 1 describes F2G. The required adaptations, when taking FP-growth algorithm [8] as the basis of comparison, are highlighted by a gray background. To illustrate them consider the new (conditional) FP-trees represented in Fig.2, obtained from the application of Algorithm 1 over the itemset database D=$\{(t_0, \{A, C, D, F\}), (t_1, \{B, D, E, F\}), (t_2, \{B, D, F\}), (t_3, \{A, B, E\}), (t_4, \{A, F\}), (t_5, \{A, B, D, E, F\})\}$.

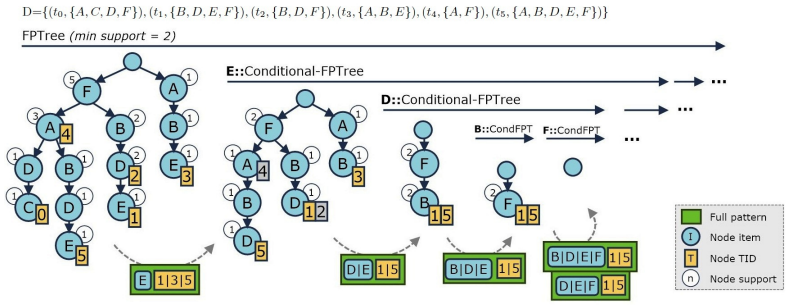

Fig.2: Illustrative behavior of F2G

The adaptations can be synthesized in three steps. *First*, the transaction-IDs are stored in the leaf node of an itemset path in the FP-tree without redundancy to guarantee optimal memory usage. To achieve this, the insertion of transactions in the FP-tree, addTransaction (*line 7*), is adapted so the identifier can be added to the item-node with least frequency. Considering the {F,A,D,C} path from the original FP-Tree in Fig.2, the identifiers t_0 and t_4 are placed in the item-nodes D and A. *Second*, before removing an infrequent node from a path (*line 16*), we need to exchange the transaction-IDs from the infrequent node to its parent (*line 17-19*). In this way the IDs assigned to the target suffixes rise from leafs to the root as long as shorter conditional FP-trees are built. Illustrating, since item C ($|\Phi_C|$=1) in the FP-tree from Fig.2 is infrequent (θ=2), t_0 is propagated upwards. *Third*, we recursively add the transaction-IDs assigned to the parental nodes of a frequent itemset to compute a full-pattern (*line 25*). Also, with the addition of a full-pattern, the identifiers need to be copied to the parental nodes (*line 30*). For the illustrative case, t_1 and t_5 are propagated towards its parent nodes during the construction of the E, D and B conditional FP-trees. The computational time added by these extensions is residual when compared with the construction of conditional trees (*lines 29, 35*).

Algorithm 2 describes the methods related with the (conditional) FP-tree construction. The computational impact of changing the methods addTransaction (*line 11*) and addPrefixPath (*line 19*) is residual. Note, additionally, that the order of items in the header list of the conditional FP-trees should preserve the order observed in the original FP-tree (*line 21*). This is due to the fact that a potential reordering of least-frequent items (different from the original ordering) may no longer preserve transactions at the end of itemset paths. To illustrate this constraint, consider the E conditional FP-tree from Fig.2. If we had opted to not maintain the order of the item nodes, B would appear below the root node, causing transaction t_5 to be relocated. This constraint has a residual impact on the performance. On one hand it has a slight deteriorate effect, since the items in

Algorithm 1: F2G Algorithm

```
   Output: FrequentFullPattern[] fullPatterns
1  Method: runFullPatternGrowthDiscovery
   Input: Transaction[] data, double support
2  Map<Int,Int> mapSup ← getItemsFrequency(data);
3  data ← removeInfrequentItems(data,mapSup);
4  data ← sortItemsets(data); //sort items in desc. freq. order
5  FPTree tree;
6  foreach Transaction trans : data do
7  |  tree.addTransaction(trans.itemset,trans.id) ;
8  tree.createHeaderList(mapSup) ;
9  F2G(tree, ∅, mapSup) ;

10 Method: F2G
   Input: FPTree tree, Itemset α, Map<Int,Int> mapSup
11 pruning(tree, α, mapSup); //FP-BONSAI optimization
12 if tree.hasSinglePath() then addAllCombForPath(tree.path, α) ;
13 else FPGrowthMultiplePaths(tree, α, mapSup) ;

14 Method: FPGrowthMultiplePaths
   Input: FPTree tree, Itemset α, Map<Int,Int> mapSup
15 foreach Int item : tree.headerList /*items in reverse order*/ do
16 |  if mapSup[item] < relativeMinsup then
17 |  |  foreach Node node : tree.getItemNodes(item) do
18 |  |  |  node.parent.trans←node.parent.trans ∪ node.trans ;
19 |  |  |  node.trans = ∅ ;
20 |  |  continue;
21 |  β.values ← α ∪ item;
22 |  β.support ← min(α.support,mapSup[item]);
23 |  foreach Node node : tree.mapItemNodes.get(item) do
24 |  |  node.parent.trans ← node.parent.trans ∪ node.trans ;
25 |  |  β.trans ← β.trans ∪ node.trans ;
26 |  fullPatterns.add(β) ;
27 |  Path[] prefixPaths; //β cond. base (prefixes co-occuring with suffix pattern)
28 |  foreach Node node: tree.getItemNodes(item) do
29 |  |  Path path = node.getParentsUntilRoot();
30 |  |  path.trans ← node.trans ;
31 |  |  prefixPaths.add(path);
32 |  Map<Int,Int> mapβSup ← getItemsSup(prefixPaths);
33 |  FPTree βtree; //β conditional FP-Tree
34 |  foreach Path path : prefixPaths do
35 |  |  βtree.addPrefixPath(path, mapβSup, θ) ;
36 |  βtree.createHeaderList(mapβSup, tree.headerList) ;
37 |  if βtree.hasNodes() then F2G(βtree, β, mapβSup) ;

38 Method: addAllCombForPath //recursively adds path nodes with prefix
   Input: Path path, Itemset α
39 Node node ← path.retrieveFirst();
40 β.items ← α ∪ node.item;
41 β.support ← node.counter;
42 β.trans ← node.trans ;
43 fullPatterns.add(β) ;
44 if path.hasMoreNodes() then
45 |  addAllCombForPath(path, α);
46 |  addAllCombForPath(path, β);
```

Algorithm 2: FP-Tree Construction Methods

```
1  Method: addTransaction
   Input: Itemset itemset, int tid /*transaction ID*/
2  Node node ← root;
3  foreach Int item : itemset.getItems() do
4  |  if node.hasChild(item) then
5  |  |  Node newNode ← createNode(item, node /*parent*/);
6  |  |  node ← newNode;
7  |  else node ← node.getChild(item) ;
8  |  if item==itemset.last() then node.trans ← node.trans ∪ tid ;

9  Method: addPrefixPath
   Input: Path path, Map<Int,Int> mapSup, Int θ /*support*/
10 Node transNode;
11 foreach Node node : path.nodes() /*backward order*/ do
12 |  if mapSup.get(node.item) < θ then continue;
13 |  ... /* code for adding a path to a FP-Tree */
14 |  transNode ← node;
15 transNode.trans ← transNode.trans ∪ path.getTransactions() ;

16 Method: createHeaderList
   Input: Map<Int,Int> mapSup, Int[] headerListSuper
17 headerList ← getItemNodes(). sortByIndexIn(headerListSuper) ;
```

the tree may not be locally order by frequency. On the other hand, this contraint is seized to optimize the efficiency of building conditional FP-trees.

Further Options: F2G method is compliant with further options that promote its scalability. In particular, we briefly motivate how F2G can comply with the adoption of compressed representations and parallelization principles.

A frequent itemset is *maximal* if is frequent and all supersets are infrequent, while is *closed* if is frequent and there exists no superset with the same support. FPMax and FPClose [6] use variants of conditional FP-trees to reduce the running time. These variants, Maximal/Closed Frequent Itemset trees, are optimally constructed to keep track of maximal/closed patterns. Since these trees preserve the item order of the original FP-tree header, the discovery of maximal/closed full-patterns can easily follow the extensions proposed in the F2G method. An alternative method is to rely on hybrid tree-projections [18].

Furthermore, parallelization and distribution principles proposed for FP-growth can be also applied for F2G to improve its scalability [7]. F2G comply with data partitioning principles [10] when relying on two steps. First, F2G is applied for each partition in order to retrieve sets of full-patterns. Second, the globally frequent itemsets are identified and their supporting transactions across partition merged. An alternative direction is to build a global FP-tree and to parallelize the (conditional) FP-tree construction methods [5], which also preserves the soundness of the F2G variant.

4 Results

In this section, we compare the performance of state-of-the-art implementations of Bitset Apriori and Eclat[1] with F2G on synthetic and real datasets. The algorithms are implemented under JVM version 1.6.0-24. The experiments were computed using an Intel Core i5 2.30GHz with 6GB of RAM.

4.1 Synthetic datasets

The properties of the generated datasets are described in Table 1. Patterns with different shapes (varying number of transactions and items) were planted. The number of supporting transactions and items for each pattern follow an Uniform distribution over the ranges presented in Table 1. These settings were developed to mimic commonly observed biclusters in gene expression matrices.

Matrix size (♯rows × ♯columns)	500×50	1000×100	2000×200	4000×400
Nr. of hidden patterns	5	10	15	25
Nr. transactions for the hidden patterns	[10,14]	[14,30]	[30,50]	[50,100]
Nr. items for the hidden patterns	[5,7]	[6,8]	[7,9]	[8,10]
Minimum support assumption	θ=1%	θ=1%	θ=1%	θ=1%

Table 1: Properties of the generated dataset settings

The data density, the average percentage of total items per transaction, can significantly vary from 5% to 50% across biological settings. Therefore, we evaluate our approach in two steps. First, the density is fixed as 20%. The changes

[1] http://www.philippe-fournier-viger.com/spmf/

in performance with varying size are the focus. Second, a specific size is fixed and the density is varied from 5% to 33%.

In Fig.3 we assess the computational overhead of adapting the FP-Growth to perform full-pattern mining. F2G adds only a residual time and memory cost. The impact of sorting items in the header table is minor. Contrasting with F2G, the naive way of attaching and gathering transactions on the FP-tree nodes (extended FP-growth) strongly penalizes the performance.

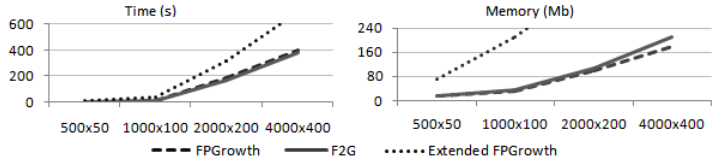

Fig.3: Efficiency of F2G and FPGrowth methods for datasets with varying size

Fig.4 compares the performance of efficient implementations of the major full-pattern miners for different data settings. F2G has significant gains in efficiency against Bitset Apriori and Eclat. The generation of candidate sets is penalized when the number of patterns is considerably high. The high-dimensionality of the datasets penalizes both the time and memory efficiency of Eclat.

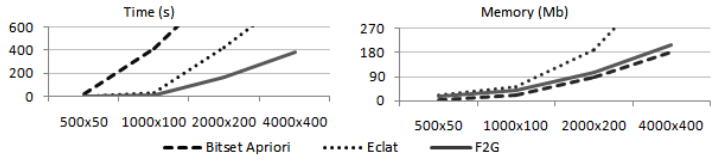

Fig.4: Efficiency of full-pattern mining methods for datasets with varying properties

To further compare the performance of full-pattern mining methods, we fixed the 1000×100 experimental setting and varied the level of sparsity. This analysis is illustrated in Fig.5. First, F2G is the approach more able to deal with dense datasets. Second, the adoption of horizontal approaches for full-pattern mining is the best option in terms of memory for the generated settings.

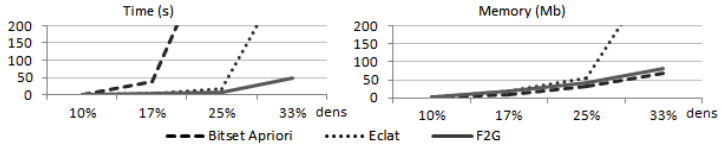

Fig.5: Efficiency of full-pattern mining methods for datasets with varying density

4.2 Real datasets

To assess the performance of the target approaches in real datasets, we adopted alternative gene expression datasets[2]. In particular, Fig.7 illustrates results for the yeast dataset for varying support thresholds. The same relative behavior can

[2] http://www.upo.es/eps/bigs/datasets.html

be observed across the remaining datasets from the selected repository. These datasets were discretized (assuming a Gaussian distribution of values and a target density of 10%) and, finally, column indexes were concatenated with items, as illustrated in Fig.1. Previous observations remain valid. F2G is the most efficient option in terms of time and, along with Apriori, a competitive choice for efficient memory usage.

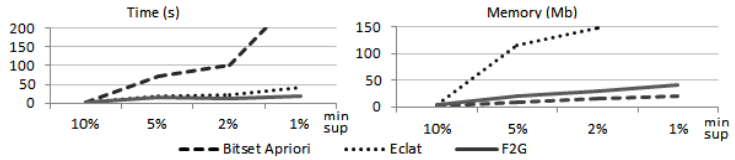

Fig.6: Efficiency of full-pattern mining methods for the yeast (2884×17) microarray

To evaluate the performance of full-pattern miners in different biological settings, we adopted alternative datasets following the procedures described in CP4IM project[3]. Fig.6 presents their performance on the primary-tumor dataset. Similarly, F2G offers the best compromise in terms of efficiency. The same relative behavior can be also observed across other UCI biological datasets, such as lymph and heart-cleveland[3].

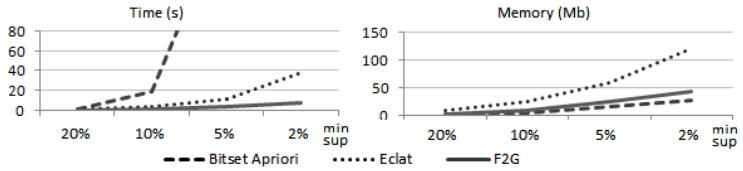

Fig.7: Efficiency of full-pattern mining methods for the primary-tumor dataset

5 Discussion

This work formalizes the full-pattern mining task and motivates its relevance for a critical set of biological applications. The performance of existing approaches is discussed. To tackle their inefficiencies, we propose the F2G algorithm, a pattern growth algorithm that discloses the supporting transactions per pattern with minimum space overhead and heightened efficiency.

F2G relies on FP-tree variants that store transaction-IDs without redundancy and on efficient propagation techniques. We show that F2G can easily accommodate principles from existing research to deliver alternative full-pattern representations or to discover full-patterns in distributed settings.

Results on both synthetic and real datasets show the superior performance of the proposed method. In particular, F2G delivers a distinctive performance both for dense and large datasets and for low support thresholds, common settings in biological tasks. To the best of our knowledge, this is the first attempt to compare full-pattern mining algorithms.

[3] http://dtai.cs.kuleuven.be/CP4IM/datasets/

Acknowledgments

This work was supported by *Fundação para a Ciência e Tecnologia* under the research project D2PM, PTDC/EIA-EIA/110074/2009, and the PhD grant SFRH/BD/75924/2011.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB. pp. 487–499. Morgan Kaufmann, San Francisco, USA (1994)
2. Alves, R., R-Baena, D., Aguilar-Ruiz, J.: Gene association analysis: a survey of frequent pattern mining from gene expression data. B.Bioinf. 11(2), 210–224 (2010)
3. Bebek, G., Yang, J.: Pathfinder: mining signal transduction pathway segments from protein-protein interaction networks. BMC Bioinformatics 8 (2007)
4. Burdick, D., Calimlim, M., Gehrke, J.: Mafia: A maximal frequent itemset algorithm for transactional databases. In: ICDE. pp. 443–452. IEEE CS (2001)
5. Chen, D., Lai, C., Hu, W., Chen, W., Zhang, Y., Zheng, W.: Tree partition based parallel frequent pattern mining on shared memory systems. In: IPDPS. p. 8 (2006)
6. Grahne, G., Zhu, J.: Efficiently using prefix-trees in mining frequent itemsets. In: FIMI. CEUR W.Proc., vol. 90. CEUR-WS (2003)
7. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15(1), 55–86 (2007)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. 29(2), 1–12 (2000)
9. Hochreiter, S., Bodenhofer, U., Heusel, M., Mayr, A., Mitterecker, A., Kasim, A., Khamiakova, T., Van Sanden, S., Lin, D., Talloen, W., Bijnens, L., Göhlmann, H., Shkedy, Z., Clevert, D.A.: FABIA: factor analysis for bicluster acquisition. Bioinformatics 26(12), 1520–1527 (2010)
10. Javed, A., Khokhar, A.: Frequent pattern mining on message passing multiprocessor systems. Distributed and Parallel Databases 16(3), 321–334 (2004)
11. Lopez, F.J., Blanco, A., Garcia-Alcalde, F., Cano, C., Marin, A.: Fuzzy association rules for biological data analysis: A case study on yeast. BMC Bioinf. 9 (2008)
12. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Trans. Comput. Biol. Bioinformatics 1(1), 24–45 (2004)
13. Martinez, R., Pasquier, C., Pasquier, N.: Genminer: Mining informative association rules from genomic data. In: BIBM. pp. 15–22. IEEE CS (2007)
14. Okada, Y., Fujibuchi, W., Horton, P.: A biclustering method for gene expression module discovery using closed itemset enumeration algorithm. IPSJ Transactions on Bioinformatics 48(SIG5), 39–48 (2007)
15. Pan, F., Cong, G., Tung, A.K.H., Yang, J., Zaki, M.J.: Carpenter: finding closed patterns in long biological datasets. In: SIGKDD. pp. 637–642 (2003)
16. Pandey, G., Atluri, G., Steinbach, M., Myers, C.L., Kumar, V.: An association analysis approach to biclustering. In: SIGKDD. pp. 677–686. ACM (2009)
17. Serin, A., Vingron, M.: Debi: Discovering differentially expressed biclusters using a frequent itemset approach. Algorithms for Molecular Biology 6, 1–12 (2011)
18. Wang, J., Han, J., Pei, J.: Closet+: searching for the best strategies for mining frequent closed itemsets. In: SIGKDD. pp. 236–245. ACM (2003)
19. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: SIGKDD. pp. 326–335. ACM, New York, NY, USA (2003)