Several Remarks on Mining Frequent Trajectories in Graphs

Henry Z. Lo[1], Dan A. Simovici[1], and Wei Ding[1]

Univ. of Massachusetts Boston, Dept. of Computer Science, Boston, Massachusetts
{henryzlo,dsim,ding}@cs.umb.edu

Abstract. We apply techniques that originate in the analysis of market basket data sets to the study of frequent trajectories in graphs. Trajectories are defined as simple paths through a directed graph, and we put forth some definitions and observations about the calculation of supports of paths in this context. A simple algorithm for calculating path supports is introduced and analyzed, but we explore an algorithm which takes advantage of traditional frequent item set mining techniques, as well as constraints placed on supports by the graph structure, for optimizing the calculation of relevant supports. To this end, the notion of the path tree is introduced, as well as an algorithm for producing such path trees.

1 Introduction

Determining frequent item sets in market basket data sets is an unsupervised data mining activity that has received a great deal of attention beginning with the seminal paper [2] and continuing with several fundamental references [6, 9, 5] A monograph dedicated to this task is [1].

Finding frequent item sets is a necessary step in computing association rules. An association rule stipulates that with a certain probability customers who buy an item set K will buy an item set H. Such rules provide actionable information for marketeers who will place items from $K \cup H$ in physical proximity in order to stimulate sales.

The purpose of this paper is to develop a study of frequent trajectories in graphs inspired by the ideas used in the analysis of market basket data sets. The study of trajectory data has been explored intensively in the literature [3, 8, 4] motivated by the large amount of spatio-temporal data allowed by location acquisition technologies. Our model is simpler than the model used in the previous references, in that, it does not include explicitly the temporal aspect. In exchange, our approach extends ideas that originate in market-basket analysis and allows us to build simple and efficient algorithms that will allow, at a later stage, the integration of the temporal aspect.

We present some preliminary results and formulate a number of open problems that we intend to approach in our future research. In the second section we set forth definitions and preliminary information relevant to our work. Theorems and observations about the implications of the graph for calculating path supports are discussed in the third section. Section 4 proposes a simple algorithm for calculating path supports. Section 5 introduces the notion of path trees, which provide insight into possible trajectories in a graph, and may be used in future work to reduce the amount of computation required for path support. The paper concludes with a discussion of further avenues of research.

2 Trajectories in Directed Graphs

Unless stated otherwise, vectors in $\mathbb{R}^l$ are row vectors, except for vectors of the form $\mathbf{e}_i$ which are column vectors; the components of $\mathbf{e}_i$ are 0 with the exception of the i^{th} component that equals 1, for $1 \leq i \leq l$.

Let $\mathcal{G} = (V, E)$ be a finite directed graph without loops having the set of vertices V and the set of edges $E \subseteq V \times V$. We assume that $|V| = n$ and $|E| = m$. If $e_k = (v_i, v_j) \in E$, we refer to v_i as the *source* of e_k and to v_j as the *target* of e_k. This defined the mappings $\mathsf{source} : E \longrightarrow V$ and $\mathsf{target} : E \longrightarrow V$ given by $\mathsf{source}(e_k) = v_i$ and $\mathsf{target}(e_k) = v_j$ for $1 \leq k \leq |E|$.

The *set of outgoing edges of a vertex* v_i is $\mathsf{out}(v_i) = \{e \in E \mid \mathsf{source}(e) = v_i\}$, while the set of *incoming edges of* v_i is $\mathsf{inc}(v_i) = \{e \in E \mid \mathsf{target}(e) = v_i\}$. The *out-degree of a vertex* v_i is the number $\mathsf{d}_+(v_i) = |\mathsf{out}(v_i)|$; *in-degree of* v_i is the number $\mathsf{d}_-(v_i) = |\mathsf{inc}(v_i)|$.

If D is a set of vertices in $\mathcal{G} = (V, E)$, denote by $\mathcal{G}_D$ the subgraph of $\mathcal{G}$ determined by the set D, $\mathcal{G}_D = (D, (D \times D) \cap E$. The previous notations are extended for D by defining the *set of outgoing edges of D* as

$$\mathsf{out}(D) = \{e \in E \mid \mathsf{source}(e) \in D, \mathsf{target}(e) \notin D\},$$

while the set of *incoming edges of* D is

$$\mathsf{inc}(D) = \{e \in E \mid \mathsf{target}(e) \in D, \mathsf{source} \notin D\}.$$

A *trajectory* in the graph $\mathcal{G}$ is a sequence of edges $(e_1, \ldots, e_p)$ such that $\mathsf{target}(e_i) = \mathsf{source}(e_{i+1})$ for $1 \leq i \leq p - 1$ and no vertex occurs twice in the sequence

$$(\mathsf{source}(e_1), \ldots, \mathsf{source}(e_p), \mathsf{target}(e_p)).$$

The directed graph $\mathcal{G}$ is represented by its incidence matrix $C_{\mathcal{G}} \in \{-1, 0, 1\}^{n \times m}$ defined as

$$(C_{\mathcal{G}})_{pk} = \begin{cases} -1 & \text{if } \mathsf{source}(e_k) = v_p, \\ 1 & \text{if } \mathsf{target}(e_k) = v_p, \\ 0 & \text{otherwise.} \end{cases}$$

If the graph is clear from context, the subscript $\mathcal{G}$ will be omitted.

Note that each column corresponds to an edge e_k and contains exactly two non-zero numbers that correspond to the source and the target of e_k. Each row corresponds to a node of the graph and contains a -1 for each edge that exits the node and an 1 for each edge that enters the node.

A trajectory is represented by a sequence $\mathbf{t} = (t_1, \ldots, t_m) \in \{0, 1\}^m$, where $m = |E|$, given by

$$t_k = \begin{cases} 1 & \text{if } e_k \text{ occurs in the trajectory,} \\ 0 & \text{otherwise.} \end{cases}$$

for $1 \leq k \leq m$.

Example 1. Consider the directed graph given in Figure 1 which has seven vertices and 14 edges. The incidence matrix $C \in \mathbb{R}^{7\times 14}$ is:

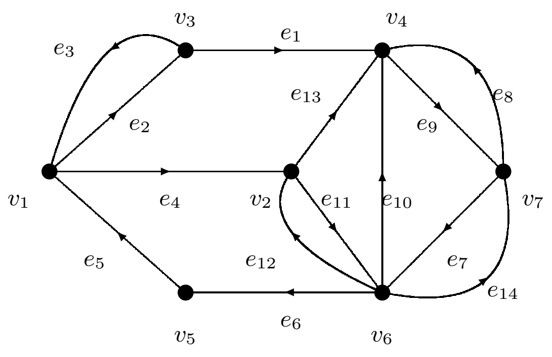

Fig. 1. Directed Graph

$$C = \begin{pmatrix} 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 \\ -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Theorem 1. *Let $\mathcal{G} = (V, E)$ be a directed finite graph with $|V| = n$ and $|E| = m$. If $\boldsymbol{t} \in \{0, 1\}^m$ represents a trajectory in the graph $\mathcal{G}$ that departs from the vertex v_i and ends in vertex v_j then*

$$C\boldsymbol{t}' = -\boldsymbol{e}_i + \boldsymbol{e}_j.$$

A trajectory table for a directed graph $\mathcal{G} = (V, E)$ is a table whose attributes are the edges of a directed graph and whose rows are trajectories. For instance, the following matrix T is a table of trajectories in the graph $\mathcal{G}$:

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}
$\mathbf{t}_1$	1	1	0	0	0	0	1	0	1	0	0	0	0	0
$\mathbf{t}_2$	0	0	0	1	1	1	0	1	1	0	0	0	1	0
$\mathbf{t}_3$	0	0	0	1	0	0	0	1	1	1	1	0	0	0
$\mathbf{t}_4$	0	0	0	1	0	0	0	0	1	0	0	0	1	0
$\mathbf{t}_5$	0	0	0	0	0	0	1	0	0	1	0	0	0	0
$\mathbf{t}_6$	0	0	0	0	0	0	1	0	0	0	0	1	1	0

By Theorem 1, the matrix CT' gives the extremities of the paths specified above

$$CT' = \begin{pmatrix} -1 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 \end{pmatrix}$$

The first, third and fourth columns refer to paths that start in v_1 and end in v_6, v_4 and v_7, respectively.

3 Support for Edge Sets

If D is a set of vertices in $\mathcal{G}$ and no trajectory begins or ends in D, then

$$\sum_{e \in \mathsf{inc}(D)} \mathsf{supp}(e) = \sum_{e \in \mathsf{out}(D)} \mathsf{supp}(e).$$

Let $E_{\mathbf{t}}$ be the set of edges that occur on a trajectory **t** in a directed graph $\mathcal{G} = (V, E)$. A set of edges K of a directed graph $\mathcal{G} = (V, E)$ occurs on a trajectory $\mathbf{t} = (e_1, \ldots, e_p)$ if $K \subseteq E_{\mathbf{t}}$.

The T-*support* of K is

$$\mathsf{supp}_T(K) = |\{\mathbf{t} \text{ in } T \mid K \subseteq E_{\mathbf{t}}\}|.$$

It is immediate that the support function $\mathsf{supp}_T : \mathcal{P}(E) \longrightarrow \mathbb{N}$ is anti-monotonic, that is, $E_1 \subseteq E_2$ implies $\mathsf{supp}_T(E_2) \leq \mathsf{supp}_T(E_1)$ for $E_1, E_2 \subseteq E$.

Unlike, the similar problem involving market basket, there exists certain interesting connections between the supports of edge sets motivated by the underlying graph structure.

Theorem 2. *Let $\mathcal{G} = (V, E)$ be a directed tree having the root v_0 and the set of leaves $\{u_1, \ldots, u_\ell\}$. The support of any path that joins v_0 to a leaf u_p equals $\min\{\mathsf{supp}(e) \mid e$ occurs on the path joining v_0 to $u_p\}$.*

Note that any directed graph has a cover that consists of directed trees because the edges of the graph yield such a cover.

4 A Simple Algorithm for Support Computation

Path supports are recorded by the object `supports` that consists of a hash map h such that $h(\wp) = \mathsf{supp}(\wp)$ for any path $\wp$, and a method `update` which sets the support of the paths.

Let Ξ be a set of pairs of the form $(\wp, s)$, where $\wp$ is a path and $s \in \mathbb{N}$. The call `supports.update`(Ξ) sets the supports of the paths that occur in the first components of the pairs of Ξ to the values specified by the second components of these pairs, respectively. When this method is called as `supports.update`$(\wp, s)$ we assume that $\Xi = \{(\wp, s)\}$. The function `recursive-traversal` takes as arguments a set of paths T, a vertex v, a path $\wp$ that ends in v, a minimal level of support θ and performs the computation shown in Algorithm 1.

```
Data: $T, v, \wp, \theta$
Result: The supports mapping
initialize supports;
foreach edge $e \in out(v)$ do
    $\wp = (\wp, e)$ ;
    $s = supp(\wp)$ ;
    supports.update($\wp, s$);
    if $s > \theta$ then
        $\Xi$ = recursive-traversal($T$, target($e$), $\wp, \theta$);
        supports.update($\Xi$);
    end
    $\wp = \wp - \{e\}$;
end
return supports;
```

Algorithm 1: The recursive function `recursive-traversal` computes the supports of all θ-frequent paths that extend a given path $\wp$

The function `recursive-traversal` is used in the function `traverse` which starts with a set of paths T, a vertex v and a minimal support θ and computes the supports of the θ-frequent paths that emerge from v. The pseudocode of this function is shown in Algorithm 2.

For 1000 trajectories and a minimal support of 0.3 the algorithm applied to trajectories that originate in v_1 generates the following results:

Path	Support
(v_1, v_2)	502
$(v_1, v_3), (v_3, v_4), (v_4, v_7), (v_7, v_6)$	354
$(v_1, v_3), (v_3, v_4), (v_4, v_7)$	498
$(v_1, v_3), (v_3, v_4)$	498
(v_1, v_3)	498

The algorithms in this paper were implemented in Python 2.7 and run on a computer with an Intel $i7 \times 980$ @ 3.33 GHz processor running Ubuntu 11.10. An experiment was run on the traverse algorithm on the graph in Figure 1, with trajectories generated as randomly terminated walks starting at vertex $v1$. Results are shown for 100, 1,000, 10,000, 100,000, and 1,000,000 trajectories. The support threshold for these experiments is 0.2.

The dependency of the average time is shown in Figure 2.

```
**Data**: Initial vertex $v$ and minimal support threshold $\theta$
**Result**: The supports mapping
initialize supports;
$T' = \emptyset$;
$C = incidence\_matrix$;
foreach path $p \in T$ do
    if $(Ct)_v == -1$ then
        $T' = T' \cup \{p\}$ ;
    end
end
$\Xi$ = recursive-traversal($T', v, \emptyset, \theta$);
supports.update($\Xi$);
return supports;
```

Algorithm 2: The function `traverse` computes the supports of all θ-frequent paths from an initial vertex v.

Table 1. Average running time of `traverse` vs. number of trajectories for minimum support 0.2

Size of data set	100	1,000	10,000	100,000	1,000,000
Time (ms)	9.645	95.331	972.251	9697.326	106538.961

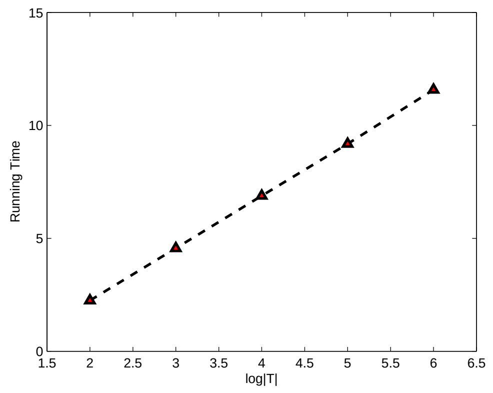

Fig. 2. Dependency of the Average Running Time on the Size of the Set of Trajectories

Table 2. Average running time (ms)/number of maximal frequent paths of `traverse` vs. θ for $|T|$ trajectories

Data	Minimal Support θ				
size	0.01	0.05	0.1	0.2	0.4
1000	102/6	102/6	98/4	95/3	87/2
10000	1026/6	1035/6	990/4	960/2	875/2
100000	10747/6	10853/6	10318/4	9981/3	8975/2

Theorem 3. *Let $\mathcal{G} = (V, E)$ be a directed tree having the root v_0 and the set of leaves $\{u_1, \ldots, u_\ell\}$. The support of any path that joins v_0 to a leaf u_p equals $\min\{\textsf{supp}(e) \mid e$ occurs on the path joining v_0 to $u_p\}$.*

5 The Path Tree of a Graph

Market basket data studies seek arbitrary frequent item sets. In contrast, we are interested here in supports of sequences of edges that form paths in the traffic graph. Thus, we need to develop an adequate counterpart to Rymon trees that are used in formulating the standard Apriori algorithm [7].

Let P_{v_i} to be the set of all simple paths which originate from vertex v_i. We can visualize P_{v_i} graphically using a tree rooted at v_i. The children of each vertex v_i in this tree are vertices which are direct successors of v_i in the graph and are not ancestors of v_i in the tree.

The *path tree* for paths that start from v_1 in the directed graph given in Figure 1 is shown in Figure 3.

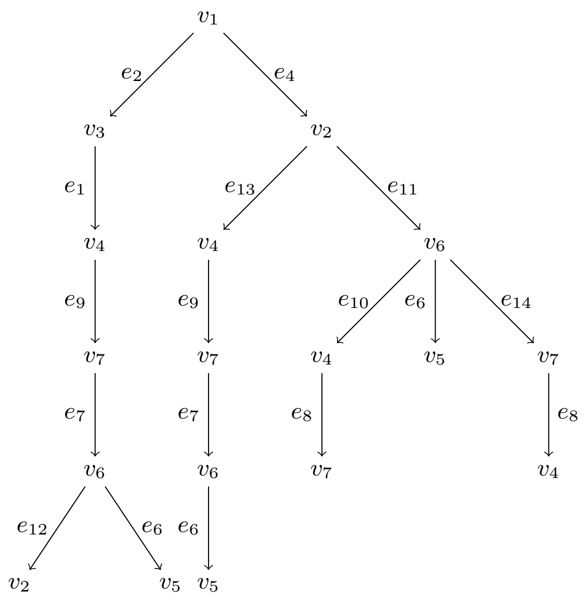

Fig. 3. Path tree for the graph in Figure 1

Note that in the path tree we could have multiple occurrences on an edge. For example, in the tree shown in Figure 3, the edge e_7 occurs twice, on the paths $e_2e_1e_9e_7$ and $e_4e_{13}e_9e_7$.

Theorem 4. *Let $\gamma_{v_i}(e_j) = \{\wp \in P_{v_i} \mid e_j$ is the last edge in $\wp\}$. That is, $\gamma_{v_i}(e_j)$ is the set of all paths which begin at vertex v_i and end with edge e_j. Then, for trajectories beginning at v_i, $\mathsf{supp}(e_j) = \sum_{\wp \in \gamma_{v_i}(e_j)} \mathsf{supp}(\wp)$.*

Note that when $\mid \gamma_{v_i}(e_j) \mid = 1$, then $\mathsf{supp}(e_j) = \mathsf{supp}(\wp)$ for $\wp \in \gamma_{v_i}(e_j)$. We can use this fact to extrapolate supports for paths which end in unique edges without actually calculating support for such a path.

The following algorithm decreases the number of computations requiring passes through all trajectories, as is required during the computation of support. It does so by using the case in the previous theorem when $\mid \gamma_{v_i}(e_j) \mid = 1$.

Note that using this method requires pre-computation of edge supports, which can be done in one pass.

Using the theorem requires the construction of a path tree rooted at some vertex v_i. However, the path tree can become intractably large. We can limit our attention to the relevant parts of the path tree by halting tree growth before edges which are not θ-frequent are added.

The following algorithm computes the set of maximal paths M_{x_i}.

```
Data: $T, v, \theta$
Result: $M_v$
initialize supports;
$T' = \emptyset$;
$C = incidence\_matrix$;
foreach path $p \in T$ do
    if $(Ct)_v == -1$ then
        $T' = T' \cup \{p\}$ ;
    end
end
$s = \sum_{t \in T'} t$;
$\xi = \{(e_i, s_i) \mid s_i \in s\}$;
supports.update($\xi$);
$M_v = \emptyset$;
foreach edge $e \in \mathsf{out}(v)$ do
    if $\mathsf{supp}(e) > \theta$ then
        $M_v = M_v \cup$ path-traverse(supports, $e, \theta$);
    end
end
return $M_v$;
```

Algorithm 3: The function `max-path` computes the set of all maximal θ-frequent paths M_v that originate from vertex v.

```
Data: supports, $\wp, \theta$
Result: set of paths $M$
$M = \emptyset$;
foreach edge $e \in$ out(target($lastedge(\wp)$)) do
    if supports$(e) > \theta$ and target$(e) \notin \wp$ then
        $M = M \cup$ path-traverse(supports, $(\wp, e), \theta$);
    end
end
if $M = \emptyset$ then
    return $\{\wp\}$;
else
    return $M$;
end
```

Algorithm 4: The recursive function `path-traverse` traverses all θ-frequent paths starting with $\wp$, and returns a set of paths M which can not be extended with θ-frequent, non-repeated edges.

For 10,000 trajectories and a minimal support of 0.1 the `max-path` function returns the following table containing the maximal paths that start from v_1:

Maximal Path
$(v_1, v_2), (v_2, v_4), (v_4, v_7), (v_7, v_6), (v_6, v_5)$
$(v_1, v_2), (v_2, v_6), (v_6, v_5)$
$(v_1, v_3), (v_3, v_4), (v_4, v_7), (v_7, v_6), (v_6, v_2)$
$(v_1, v_3), (v_3, v_4), (v_4, v_7), (v_7, v_6), (v_6, v_5)$

The dependency of the average running time versus the size of the data set for a minimal support level of 0.2 is shown in Table 3.

Table 3. Average running time of `max-path` vs. number of trajectories for minimum support 0.2

Size of data set	100	1,000	10,000	100,000	1,000,000
Time (ms)	0.173378	1.099992	9.958768	100.506067	996.609592

The dependency of the average running time and the number of maximal paths on the size of the data set and the minimal support is presented in Table 4.

6 Conclusion and Further Work

There are many issues left to investigate. Support may be defined for a variety of types of sets of edges and connections between supports for various sets of edges out to be analyzed and used to simplify algorithms for computing supports.

Table 4. Average running time (ms)/number of maximal paths of `max-path` vs. θ for $|T|$ trajectories

Data	Minimal Support θ				
size	0.01	0.05	0.1	0.2	0.4
1000	1.14/6	1.14/6	1.11/4.1	1.11/3	1.06/2
10000	10.10/6	10.03/6	10.03/4	10.25/3	10.57/2
100000	102.38/6	104.12/6	107.64/4	102.07/3	102.01/2
1000000	1002.63/6	1002.18/6	1002.06/4	1002.22/3	1002.33/2

Association rule need to be explored in this context. Connections between the confidence of rules of the form $\wp \rightarrow e_1, \ldots, \wp \rightarrow e_h$, where $e_1, \ldots, e_h$ are edges that continue the path $\wp$ can be used to simplify the computation of confidence of such rules.

A measure of "attractiveness" can be introduced for path that join two vertices v_1 and v_2. The tradeoff between the length of the path and the support of the path (which shows the number of drivers that take the path) can be used for defining such a measure.

References

1. J.-M. Adamo. *Data Mining for Association Rules and Sequential Patterns*. Springer-Verlag, New York, 2001.
2. Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993. ACM, New York.
3. Fosca Giannotti. Mobility, data mining and privacy: Mining human movement patterns from trajectory data. In *Extraction et gestion des connaissances (EGC'2011), Actes, 25 au 29 janvier 2011, Brest, France*, volume RNTI-E-20 of *Revue des Nouvelles Technologies de l'Information*, pages 5–6. Hermann-Éditions, 2011.
4. Fosca Giannotti, Mirco Nanni, Dino Pedreschi, Fabio Pinelli, Chiara Renso, Salvatore Rinzivillo, and Roberto Trasarti. Unveiling the complexity of human mobility by querying and mining massive trajectory data. *VLDB Journal*, 20(5):695–719, 2011.
5. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Dallas, TX,* pages 1–12. ACM, New York, 2000.
6. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. Technical Report C-1997-8, University of Helsinki, 1997.
7. D. Simovici and C. Djeraba. *Mathematical Tools for Data Mining*. Springer-Verlag, New York, 2008.
8. Roberto Trasarti, Fabio Pinelli, Mirco Nanni, and Fosca Giannotti. Mining mobility user profiles for car pooling. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011,* pages 1190–1198. ACM, 2011.
9. M. J. Zaki and C.J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17:462–478, 2005.