

Computational Aspects of Mining Maximal Frequent Patterns ¹

Guizhen Yang²

Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA

Abstract

In this paper we study the complexity-theoretic aspects of mining maximal frequent patterns, from the perspective of counting the number of all distinct solutions. We present the first formal proof that the problem of counting the number of maximal frequent itemsets in a database of transactions, given an arbitrary support threshold, is #P-complete, thereby providing theoretical evidence that the problem of mining maximal frequent itemsets is NP-hard. We also extend our complexity analysis to other similar data mining problems that deal with complex data structures, such as sequences, trees, and graphs. We investigate several variants of these mining problems in which the patterns of interest are subsequences, subtrees, or subgraphs, and show that the associated problems of counting the number of maximal frequent patterns are all either #P-complete or #P-hard.

Key words: data mining, complexity, maximal frequent patterns, #P-complete

1 Introduction

Since the invention of the Apriori algorithm about a decade ago [1], the field of data mining has flourished into a research area of significant technological and social importance, with applications ranging from business intelligence [2–5] to security [6–8] to bioinformatics [9–11]. However, in spite of the multitude of data mining algorithms developed, not much effort has been made on the theoretical front end to study the inherent complexity nature of data mining

¹ The extended abstract of this paper appeared in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2004.

² This work was done while the author was a faculty member in Department of Computer Science and Engineering, University at Buffalo, The State University of New York.

problems themselves. A thorough investigation of these fundamental problems is greatly needed since it will not only provide invaluable insights into many data mining problems but will also shed new light on the characteristics of different data mining algorithms and benchmark datasets.

In this paper we seek to provide a theoretical account of the computational difficulty of a genre of data mining problems that deal with maximal frequent patterns. These problems can be viewed as instances of the *theory extraction* problem [12] — they are mainly concerned with *enumerating* all frequent patterns (described using some language) which satisfy some property and are present in a sufficiently large number of transactions (records) in a database. Examples of this sort include frequent itemsets, association rules, induced subgraphs, etc. Normally, a partial order, \preceq , can be defined among all frequent patterns in such a way as to preserve the *downward closure* property, i.e., given any patterns p_1 and p_2 , if $p_1 \preceq p_2$ and p_2 is frequent, so is p_1 . Hence *maximal* frequent patterns are those frequent patterns that do not have any successor with respect to this partial order. Mining maximal frequent patterns is an important problem because the set of maximal frequent patterns not only uniquely defines a theory given an interestingness criterion, but the number of maximal frequent patterns *can* be significantly smaller than the number of frequent patterns as well [12].

We study the complexity of data mining problems from the perspective of *counting* the number of solutions. It is natural to assume that any algorithm which can enumerate all the solutions would be able to count them as efficiently as well. This counting aspect reveals the inherent complexity nature of data mining problems — the expected output is merely a number instead of a presentation of all the solutions; even an exponential number only requires a polynomial number of bits of storage space in binary notation. Therefore, given an enumeration problem, its associated counting problem may have *lower* time and/or space complexity (see Section 3.1 for such an example).³

We use the notion of #P-completeness as a theoretical analysis tool to study the computational complexity of a given counting problem. The class #P was first introduced in [13,14] to include all counting problems for which any single solution can be computed by a nondeterministic Turing machine in polynomial time [15]. The notion of #P-completeness is therefore used to capture the “hardest” problems in #P (see Section 3.2 for details). These #P-complete problems provide natural candidates for the type of problems that may still remain *intractable* even if P=NP [15], since under this computation model, an “efficient” algorithm for solving a #P-complete problem would behave as if it could, by magic, guess the exact number of correct solutions and simultaneously validate all of them in polynomial time.

³ The complexity results herein are only concerned with *worst-case* complexity.

Our theoretical investigation begins with the problem of mining maximal frequent itemsets — one of the most fundamental problems studied in data mining [16,12,17–21]. We present the first formal proof (see Section 4) that the problem of counting the number of maximal frequent itemsets in a database of transactions, given an arbitrary support threshold, is $\#P$ -complete, thereby providing theoretical evidence that the problem of mining maximal frequent itemsets is NP-hard, i.e., intractable in the worst case. Since the existence of a maximal frequent itemset can be checked in polynomial time, this result identifies the problem of counting the number of maximal frequent itemsets as one of the few known counting problems whose associated decision problems are “easy”, i.e., belong to P.

It is worth pointing out that the NP-hardness of mining maximal frequent itemsets was also recently established in [22] by proving the following claim: given a set of maximal frequent itemsets, it is NP-complete to decide whether this set can be grown with new maximal frequent itemsets. In contrast, our result asserts a much stronger claim about the hardness of mining maximal frequent itemsets — it is $\#P$ -complete to decide the exact number of all maximal frequent itemsets — there is no clear clue that this problem would belong to NP.

Note that number of maximal frequent itemsets can be exponentially smaller than the number of frequent itemsets [16,12]. In contrast to mining frequent itemsets, several algorithms have been shown to be able to gain computational efficiency substantially for mining maximal frequent itemsets [16,12,17–21]. Given that the problem of counting the number of frequent itemsets has also been shown to be $\#P$ -complete [12], our new complexity result implies a rather unexpected analogy: the problem of counting maximal frequent itemsets is of as great worst-case computational complexity as the problem of counting frequent itemsets.

Having established the $\#P$ -completeness of the counting problem for mining maximal frequent itemsets, we extend our complexity analysis to other similar problems that deal with complex data structures, such as sequences [23–29], trees [30–32], and graphs [33–36], which have attracted intensive research interests in recent years. We investigate several variants of these mining problems in which the patterns of interest are subsequences, subtrees, or subgraphs, and show that their associated problems of counting the number of maximal frequent patterns are all either $\#P$ -complete or $\#P$ -hard (our complexity results are summarized in Table 2).

The rest of this paper is organized as follows. Section 2 introduces the basic concepts and notations to be used throughout this paper. In Section 3 we introduce the theory of $\#P$ -completeness. Section 4 presents our formal proof that the problem of counting the number of maximal frequent item-

sets is #P-complete. The complexity results concerning other maximal frequent patterns, including subsequences, subtrees, and subgraphs, are presented Section 5. Finally, we discuss related work in Section 6 and conclude this paper in Section 7.

2 Preliminaries

Here we introduce the important concepts and notations that will be used throughout the paper. First we describe how to represent databases using bipartite graphs and binary matrices (see [37] for a more detailed survey). Then we will formalize several notions of itemsets with different support characteristics. Table 1 summarizes the notations that will be used in this paper and their meanings.

\mathcal{D}	a database of transactions
I	an itemset
$ S $	the cardinality of a set S
$\mathcal{D}(I)$	the set of transactions in database \mathcal{D} that contain I
$f_{\mathcal{D}}(I)$	the support of I in database \mathcal{D}
$\mathcal{G}_{\mathcal{D}}$	the bipartite graph corresponding to database \mathcal{D}
$\mathcal{A}_{\mathcal{D}}$	the binary matrix corresponding to database \mathcal{D}
$\Pi \propto_T \Pi'$	problem Π Turing reduces to problem Π'
$\mathcal{F}_{\sigma}(\mathcal{D})$	the set of all σ -frequent itemsets in database \mathcal{D}
$\mathcal{M}_{\sigma}(\mathcal{D})$	the set of all maximal σ -frequent itemsets in database \mathcal{D}
$\mathcal{C}_{\delta}(\mathcal{D})$	the set of all maximal δ -occurrent itemsets in database \mathcal{D}

Table 1. Summary of Notations and Their Meanings

2.1 Databases and Itemsets

A database comprises a set of transactions (records). Each transaction has a unique transaction identifier (tid) and contains a set of items. For simplicity we will normally omit the tid of a transaction and just list the set of items that it contains. A set of items is often called an *itemset*. Let I be an itemset and t a transaction. We will use the notation, $I \subseteq t$, to denote that I is a

subset of the set of items that t contains. When the context is clear, we will often directly refer to a transaction as the set of items that it contains.

Given a set S , we will use the notation, $|S|$, to denote the *cardinality* of S , i.e., the number of elements in S . Let I be an itemset and \mathcal{D} a database of transactions. We will use the notation, $\mathcal{D}(I)$, to represent the set of transactions of \mathcal{D} that are a superset of I , i.e., $\mathcal{D}(I) \stackrel{\text{def}}{=} \{t \mid I \subseteq t, t \in \mathcal{D}\}$.

2.2 Bipartite Graphs and Binary Matrices

A *bipartite graph*, \mathcal{G} , can be represented as a triple, $\mathcal{G} = (U, V, E)$, where U and V are *disjoint* sets of vertices and E is the set of edges between vertices in U and V such that $E \subseteq U \times V$.

A bipartite graph $\mathcal{G} = (U, V, E)$ is called a *bipartite clique* if there is an edge between *every* pair of vertices in U and V , i.e., $E = U \times V$. Usually we will omit the set of edges when we write down a bipartite clique. Given a bipartite clique, $\mathcal{G} = (U, V)$, where $|U| = m$ and $|V| = n$, we will call it a bipartite (m, n) -clique, or a bipartite $(m, *)$ -clique (if the cardinality of V is of no importance), or a bipartite $(*, n)$ -clique (if the cardinality of U is of no importance).

We will say that a bipartite graph, $\mathcal{G}' = (U', V', E')$, *appears in* another bipartite graph, $\mathcal{G} = (U, V, E)$, if $U' \subseteq U, V' \subseteq V, E' \subseteq E$. Of particular interest are those bipartite cliques that appear in a given bipartite graph. We will say that a bipartite clique, $\mathcal{G}' = (U', V')$, is a *maximal* bipartite clique in a given bipartite graph \mathcal{G} , if \mathcal{G}' appears in \mathcal{G} and there exists no other bipartite clique, $\mathcal{G}'' = (U'', V'')$, in \mathcal{G} such that $U' \subseteq U''$ and $V' \subseteq V''$.

One can easily establish a one-to-one correspondence between bipartite graphs and databases of transactions. Given a database \mathcal{D} , its corresponding bipartite graph, denoted $\mathcal{G}_{\mathcal{D}} = (U, V, E)$, can be constructed as follows: U comprises all database transactions in \mathcal{D} ; V comprises all items appearing in \mathcal{D} ; for all $u \in U, v \in V, (u, v) \in E$ iff $v \in u$, i.e., transaction u contains item v . On the other hand, given a bipartite graph, $\mathcal{G} = (U, V, E)$, its corresponding database of transactions, denoted $\mathcal{D}_{\mathcal{G}}$, can be constructed as follows: let U represent the set of database transactions and V the set of items; transaction u contains item v iff $(u, v) \in E$.

A *binary matrix* is a matrix in which each entry has value either 0 or 1. A one-to-one correspondence between binary matrices and databases of transactions can also be established rather straightforwardly. Given a database \mathcal{D} , we number its transactions as t_1, t_2, \dots, t_m (corresponding to rows 1 to m of a matrix) and all the items as x_1, x_2, \dots, x_n (corresponding to columns 1 to n

of a matrix). Then \mathcal{D} corresponds to an $m \times n$ matrix, denoted $\mathcal{A}_{\mathcal{D}}$, in which its entry a_{ij} has value 1 iff transaction t_i contains item x_j ; otherwise, its value is 0. Given a binary matrix \mathcal{A} , we will use $\mathcal{D}_{\mathcal{A}}$ to denote its corresponding database of transactions.

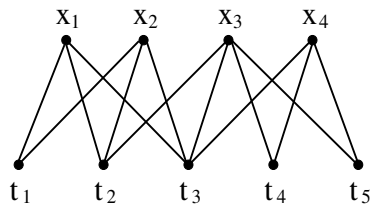


Fig. 1. Bipartite Graph Representation of the Database in Example 1

Example 1 Consider a database \mathcal{D} that consists of the following transactions, t_1, t_2, t_3, t_4, t_5 , where

$$\begin{aligned} t_1 &= \{x_1, x_2\} \\ t_2 &= \{x_1, x_2, x_3\} \\ t_3 &= \{x_1, x_2, x_3, x_4\} \\ t_4 &= \{x_3, x_4\} \\ t_5 &= \{x_3, x_4\} \end{aligned}$$

Here x_1, x_2, x_3, x_4 are the items in \mathcal{D} . The corresponding bipartite graph, $\mathcal{G}_{\mathcal{D}}$, and binary matrix, $\mathcal{A}_{\mathcal{D}}$, are illustrated in Figures 1 and 2, respectively. \square

	x_1	x_2	x_3	x_4
t_1	1	1	0	0
t_2	1	1	1	0
t_3	1	1	1	1
t_4	0	0	1	1
t_5	0	0	1	1

Fig. 2. Binary Matrix Representation of the Database in Example 1

In the sequel, we will use either binary matrices or bipartite graphs to represent databases of transactions. Whenever we speak of the size of a database of transactions, normally we mean the size of its corresponding bipartite graph under some commonsense economic representation. We will not explicitly specify a particular data structure for storage.

2.3 Support and Maximality

Frequent itemset mining is mainly concerned with those itemsets that are present in a sufficiently large number of database transactions. The number of

occurrences of an itemset in a database is commonly referred to as the *support* of this itemset, formalized as follows.⁴

Definition 1 (Support) Let I be an itemset and \mathcal{D} a database of transactions. The support of I in \mathcal{D} , denoted $f_{\mathcal{D}}(I)$, is the number of transactions of \mathcal{D} in which I occurs as a subset, i.e., $f_{\mathcal{D}}(I) \stackrel{\text{def}}{=} |\mathcal{D}(I)|$.

Lemma 1 Let \mathcal{D} be a database of transactions, I and J two itemsets. If $I \subseteq J$, then $\mathcal{D}(I) \supseteq \mathcal{D}(J)$ and $f_{\mathcal{D}}(I) \geq f_{\mathcal{D}}(J)$.

Note that even if two database transactions contain the same set of items they are still different from each other, since each transaction has its own unique tid. Therefore, they will each contribute one count towards the support of an itemset that they contain.

Definition 2 (δ -Occurrent Itemset) An itemset I is called δ -occurent in a database \mathcal{D} , where $1 \leq \delta \leq |\mathcal{D}|$, if the support of I in \mathcal{D} is δ , i.e., $f_{\mathcal{D}}(I) = \delta$.

Definition 3 (σ -Frequent Itemset) An itemset I is called σ -frequent in a database \mathcal{D} , where $1 \leq \sigma \leq |\mathcal{D}|$, if $f_{\mathcal{D}}(I) \geq \sigma$, i.e., the support of I in \mathcal{D} is at least σ .

Lemma 2 Let \mathcal{D} be a database of transactions, I and J two itemsets. If $I \subseteq J$ and both I and J are δ -occurent in \mathcal{D} , then $\mathcal{D}(I) = \mathcal{D}(J)$.

PROOF. Since J is a δ -occurent itemset in \mathcal{D} , let $\mathcal{D}(J) = \{t_1, t_2, \dots, t_\delta\}$. It follows that $I \subseteq J \subseteq t_i$ for all $1 \leq i \leq \delta$. Because I is also δ -occurent in \mathcal{D} , so $\mathcal{D}(I) = \{t_1, t_2, \dots, t_\delta\} = \mathcal{D}(J)$. \square

In the sequel, when we discuss properties of itemsets with respect to a database \mathcal{D} , for brevity we will usually omit the database \mathcal{D} , especially when \mathcal{D} is fixed or its existence is clear from the context.

Having defined the notion of σ -frequent itemsets, now we can formally state the problem of mining frequent itemsets as follows:

Given a database of transactions \mathcal{D} and an arbitrary integer value σ , where $1 \leq \sigma \leq |\mathcal{D}|$, enumerate all σ -frequent itemsets in \mathcal{D} .

We should point out the subtle difference between δ -occurent and σ -frequent itemsets. If an itemset is δ -occurent, then its support must be *exactly* δ . The

⁴ The support of an itemset can be also defined as the *percentage* of transactions in which this itemset occur. For convenience here we use an integer value to define the support of an itemset, since it can always be computed by multiplying the percentage number by the total number of database transactions.

support of a σ -frequent itemset, however, can be any value greater than or equal to σ . Clearly, if an itemset is σ -frequent, then it must be δ -occurrent for some $\delta \geq \sigma$.

Example 2 Consider again the database \mathcal{D} in Example 1. Its bipartite graph and binary matrix representations are illustrated in Figures 1 and 2, respectively. One can easily validate the following: $\{x_2, x_3\}$ is a 2-occurrent itemset ($\mathcal{D}(\{x_2, x_3\}) = \{t_2, t_3\}$); $\{x_1, x_2\}$ is a 3-occurrent and 2-frequent itemset ($\mathcal{D}(\{x_1, x_2\}) = \{t_1, t_2, t_3\}$). \square

If we consider subset inclusion as defining a partial order for itemsets, then we can introduce the notions of maximal δ -occurrent and maximal σ -frequent itemsets, as follows.

Definition 4 (Maximal δ -Occurrent Itemset) Suppose I is a δ -occurrent itemset in a database of transactions \mathcal{D} . We say that I is a maximal δ -occurrent itemset in \mathcal{D} , if there exists no itemset J such that $J \supset I$ and J is δ -occurrent in \mathcal{D} .⁵

Definition 5 (Maximal σ -Frequent Itemset) Suppose I is a σ -frequent itemset in a database of transactions \mathcal{D} . We say that I is a maximal σ -frequent itemset in \mathcal{D} , if there exists no itemset J such that $J \supset I$ and J is σ -frequent in \mathcal{D} .

Now that we have introduced maximal σ -frequent itemsets, we can formally state the problem of mining maximal frequent itemsets as follows:

Given a database of transactions \mathcal{D} and an arbitrary integer value σ , where $1 \leq \sigma \leq |\mathcal{D}|$, enumerate all maximal σ -frequent itemsets in \mathcal{D} .

One can easily see that if an itemset I is σ -frequent, then any (nonempty) subset $J \subset I$ is also σ -frequent. On the other hand, if $J \subset I$ is *not* σ -frequent, then I cannot be σ -frequent either. Note that once all the maximal σ -frequent itemsets have been computed, then all the σ -frequent itemsets can be directly enumerated from them without having to read from the database any more. Conceptually the information about σ -frequent itemsets can be “summarized” using maximal σ -frequent itemsets — the number of maximal σ -frequent itemsets *can* be significantly smaller than the number of σ -frequent itemsets.

Note that if I is a δ -occurrent itemset, it does not necessarily mean that any subset $J \subset I$ is also δ -occurrent. It must be true, however, that J is λ -occurrent for some $\lambda \geq \delta$.

⁵ A maximal δ -occurrent itemset is essentially a frequent closed itemset with support δ [38], where δ is greater than or equal to the support threshold. This explicit notation will be handy for our complexity analysis.

The notion of maximal occurrent itemsets plays an important role in our complexity analysis of mining maximal frequent itemsets. In Section 4 we will develop lemmas to establish several connections between maximal occurrent and maximal frequent itemsets. The following example illustrates these concepts.

Example 3 Continue with the database in Example 1. One can easily validate the following: $\{x_2, x_3\}$ is a 2-occurrent itemset but not maximal, since $\{x_1, x_2, x_3\}$ is also 2-occurrent; $\{x_1, x_2, x_3\}$ is a maximal 2-frequent itemset; $\{x_1, x_2\}$ is a maximal 3-occurrent itemset but not a maximal 2-frequent itemset; $\{x_3, x_4\}$ is a maximal 3-occurrent and maximal 2-frequent itemset. \square

Let \mathcal{D} be a database and $\mathcal{G}_{\mathcal{D}}$ its corresponding bipartite graph. In Section 2.2 we show that there is a one-to-one correspondence between bipartite graphs and databases of transactions. In fact, there is also a one-to-one correspondence between maximal occurrent itemsets in \mathcal{D} and maximal bipartite cliques in $\mathcal{G}_{\mathcal{D}}$. Their relationship is formally stated in the following lemma.

Lemma 3 Let \mathcal{D} be a database of transactions and $\mathcal{G}_{\mathcal{D}}$ the bipartite graph corresponding to \mathcal{D} . Then every maximal δ -occurrent itemset in \mathcal{D} corresponds to a unique maximal bipartite $(\delta, *)$ -clique in $\mathcal{G}_{\mathcal{D}}$.

Example 4 Consider the bipartite graph in Figure 1 which corresponds to the database in Example 1. Note that $\{x_1, x_2\}$ is a maximal 3-occurrent itemset and corresponds to the unique bipartite $(3, 2)$ -clique, $(\{t_1, t_2, t_3\}, \{x_1, x_2\})$, in Figure 1. \square

3 Theoretical Foundations

Most data mining problems espouse two different but closely related aspects: enumeration of all solutions and counting the number of all solutions. In this section we will first discuss these two aspects for the problem of mining maximal frequent itemsets. Then we introduce the notion of #P-completeness as a complexity analysis tool for the class #P of counting problems.

3.1 Enumeration vs. Counting

The problem of mining maximal frequent itemsets, as formally defined in Section 2.3, is to *enumerate* all maximal frequent itemsets whose support is no less than a preset threshold. A natural question that one may ask is:

What is the (worst-case) computational complexity of enumerating all the maximal frequent itemsets?

Since all the maximal frequent itemsets must be enumerated, an immediate observation is that the computational cost must be *at least* proportional to the *number* of maximal frequent itemsets. Conceptually, maximal frequent itemsets constitute a “summary” of all frequent itemsets given the same support threshold — the number of all maximal frequent itemsets can be substantially smaller than the number of frequent itemsets. But the question still remains:

Is the number of maximal frequent itemsets always polynomial in the size of the database?

Unfortunately the answer to the question above turns out to be “No”. In the following example we will show a database of transactions with an exponential number of maximal frequent itemsets at a certain support threshold.

	x_1	x_2	\cdots	\cdots	\cdots	x_{2n-1}	x_{2n}
t_1	0	1	1	\cdots	1	1	1
t_2	1	0	1	\cdots	1	1	1
\vdots	1	1	0	\cdots	1	1	1
\vdots				\ddots			
\vdots	1	1	1	\cdots	0	1	1
t_{2n-1}	1	1	1	\cdots	1	0	1
t_{2n}	1	1	1	\cdots	1	1	0

Fig. 3. A Database with an Exponential Number of Maximal Frequent Itemsets

Example 5 Let $X = \{x_1, x_2, \dots, x_{2n-1}, x_{2n}\}$ denote a set of $2n$ items. We will construct a database \mathcal{D} with $2n$ transactions, $t_1, t_2, \dots, t_{2n-1}, t_{2n}$, as follows: $t_i = X - \{x_i\}$ for all $1 \leq i \leq 2n$, i.e., transaction t_i comprises all the items in X except item x_i . The database \mathcal{D} is illustrated schematically in Figure 3.

Now we claim that the number of maximal n -frequent itemsets in \mathcal{D} is $\binom{2n}{n}$. To see why, note that for any itemset $I \subseteq X$, $\mathcal{D}(I) = \{t_i \mid x_i \notin I, x_i \in X\}$. It follows that if $|I| = k$ then $f_{\mathcal{D}}(I)$ (the support of I in \mathcal{D}) is exactly $2n - k$. Thus any itemset I is maximal n -frequent iff $|I| = n$, since for any itemset J , if $J \supset I$ then it must be true that $f_{\mathcal{D}}(J) < n$. Clearly, there are $\binom{2n}{n}$ number of different itemsets of size n . So the number of maximal n -frequent itemsets is $\binom{2n}{n}$. The size of \mathcal{D} is $O(n^2)$. Moreover, $\binom{2n}{n} = \frac{(2n)!}{n! \cdot n!} \geq 2^n$. Hence the number of maximal n -frequent itemsets in \mathcal{D} is exponential in the size of \mathcal{D} . \square

Example 5 above provides a strong indication that no algorithm can efficiently enumerate all maximal frequent itemsets in the worst case, given an arbitrary support threshold. The reason is just downright straightforward — the number of maximal frequent itemsets may be exponential in the worst case — one

would just need to spend at least an exponential amount of time to “print” them out, let alone the additional cost to “compute” them!

However, the argument above is still not convincing enough. First, it does not constitute a formal proof that the problem of mining (enumerating) all maximal frequent itemsets is “hard”. Second, one may, albeit arguably, claim that only “printing” but not “computing” of all the maximal frequent itemsets takes an exponential amount of time — an algorithm smart enough might be able to “compute” and “compress” all the maximal frequent itemsets using some efficient data structure in polynomial time. Indeed, such discrepancy between “printing” and “computing” is not uncommon. For instance, it takes *quadratic time* to print out all the suffixes of a string; but an efficient data structure, the so-called suffix tree, can be constructed in *linear time* which encodes all the suffixes of a given string [39].

It is natural to assume that if an algorithm can enumerate all maximal frequent itemsets, then it should be able to count them as efficiently as well. This counting aspect of data mining problems is important because in contrast to enumeration, the associated counting problem might have “lower” time and/or space complexity. In fact, an exponential number requires only a polynomial number of bits to store. For instance, the number $\binom{2n}{n}$ needs just $O(n \log_2 n)$ bits to encode in binary notation. Moreover, mathematical operations, such as addition, subtraction, multiplication, division, etc., only take a polynomial (in the sizes of the two operands) number of steps to finish. The following example illustrates another interesting problem for which an exponential number of solutions can be counted efficiently in polynomial time.

Example 6 Given the following context-free grammar:

$$S \rightarrow SS \mid c$$

consider the problem of counting the number of different parse trees, denoted $T(n)$, for any string of n number of c 's, where $n \geq 1$. Clearly, $T(n)$ can be defined recursively as follows:

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 1 \\ T(n) &= \sum_{k=1}^{n-1} T(k) \cdot T(n-k), \text{ where } n \geq 3 \end{aligned}$$

One can verify that $\frac{2^n}{5} \leq T(n) \leq 2^{n^3}$. Clearly, the number $T(n)$ is exponential and can be computed in polynomial time using a dynamic programming approach. The example above may seem trivial, but in the same vein our result can be easily extended to more general settings. In fact, given an arbitrary context-free grammar in Chomsky normal form, the number of different

parse trees for any input string can be computed in polynomial time using the Coke-Younger-Kasami algorithm [40]. Note that under this setting, the number of different parse trees is bounded by $q^n \cdot 2^{n^3}$ but may be greater than $\frac{q^n \cdot 2^n}{5}$ in the worse case, where q is the size of the grammar and n the size of the input string (see [40] for more details). \square

Therefore, if an algorithm is claimed to be able to “compute” all maximal frequent itemsets in polynomial time, then it is natural to assume that it should be able to count them as efficiently as well; otherwise, such a claim is not justified. In light of this intrinsic connection between computing and counting, from now on we will focus on the counting aspect of data mining problems. First we revise our original problem definition accordingly as follows:

Given a database of transactions \mathcal{D} and an arbitrary integer value σ , where $1 \leq \sigma \leq |\mathcal{D}|$, count the number of all maximal σ -frequent itemsets in \mathcal{D} .

In the rest of this paper, we will develop theorems to show that the “easier” counting problem above is in fact computationally difficult, thereby presenting a formal proof that its associated problem of enumerating (computing) all solutions is hard.

3.2 The Complexity of Counting

In the theory of NP-completeness, the kind of problems under consideration are invariably decision problems which are mainly concerned with the *existence* of a solution to a problem. On the contrary, whereas enumeration problems require explicit output of all solutions, counting problems need to calculate the number of solutions only.⁶ Clearly, if a decision problem is NP-complete, then its associated enumeration or counting problem must be NP-hard, because being able to enumerate all solutions or knowing the number of solutions is enough to answer the question of whether there is one.

The class #P of counting problems was first introduced by Valiant to give a complexity-theoretic characterization of the computational difficulty of counting [13]. Here we follow the same definitions as in [13].

Definition 6 (Counting Turing Machine) A counting Turing machine is a standard nondeterministic Turing machine with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input. It has (worst-case) time complexity $f(n)$ if the longest accepting computation induced by the set of all

⁶ Note that the counting problems we describe here are termed as enumeration problems in [15]. But we follow the terminology in [41].

inputs of size n takes $f(n)$ steps (when the Turing machine is regarded as a standard nondeterministic machine without the auxiliary device).

Definition 7 (#P) A counting problem belongs to #P if this problem can be solved by a counting Turing machine of polynomial time complexity.

A problem is said to be #P-hard if all problems in #P *reduce* to it. Note that the notion of reduction used here is polynomial-time Turing reduction (or simply Turing reduction; see [15] and [13] for more details). More specifically, a Turing reduction from one problem Π to another problem Π' is an algorithm that solves Π using a hypothetical oracle for solving Π' such that, if this oracle solves Π' in polynomial time, then the overall algorithm would be a polynomial-time algorithm for Π . We will use the notion, $\Pi \propto_T \Pi'$, to imply that problem Π Turing reduces to problem Π' . Clearly, if $\Pi \propto_T \Pi'$ and $\Pi' \propto_T \Pi''$, then $\Pi \propto_T \Pi''$.

Just as the concept of NP-completeness is introduced for the “hardest” problems in NP, the concept of #P-completeness is used to capture the notion of the “hardest” problems in #P. Formally, we have the following definition.

Definition 8 (#P-Complete) A counting problem is called #P-complete if all problems in #P Turing reduce to it and it belongs to #P.

It is easy to see that #P is the set of counting problems naturally associated with the decision problems in NP. Therefore, for those NP-complete problems in NP, their associated counting problems are #P-complete⁷ — the hardness of counting the number of solutions for such problems originates from the computational difficulty of searching for just one solution! For instance, the problem of counting the number of satisfying truth assignments for an arbitrary 3CNF formula is #P-complete [15].

However, there are other hard counting problems whose associated decision problems can actually be solved in polynomial time. The first such problem was proved by Valiant [13] — the problem of counting the number of perfect matchings in a bipartite graph is #P-complete. In the following Section 4 we will show that the problem of counting the number of maximal frequent itemsets falls into this same category also. Clearly, if a counting problem is #P-complete or #P-hard, then its associated enumeration (mining) problem must be NP-hard [15,41].

⁷ Strictly speaking, this claim is still a conjecture. But the associated counting problems of many known NP-complete problems have been formally proved to be #P-complete.

4 Complexity Analysis

In this section we will present a formal proof that the problem of counting the number of maximal frequent itemsets is #P-complete. First we introduce the new notations to be used here. Let \mathcal{D} be a database of transactions. We will use the notation $\mathcal{F}_\sigma(\mathcal{D})$ to denote the set of all σ -frequent itemsets, $\mathcal{M}_\sigma(\mathcal{D})$ to denote the set of all maximal σ -frequent itemsets, and $\mathcal{C}_\delta(\mathcal{D})$ to denote the set of all maximal δ -occurrent itemsets.

Since our focus is on the number of maximal frequent itemsets, it is important to see how this number changes with respect to different support thresholds. We will begin with the number of frequent itemsets.

Lemma 4 If $\sigma > \lambda$, then $\mathcal{F}_\sigma(\mathcal{D}) \subseteq \mathcal{F}_\lambda(\mathcal{D})$.

From the lemma above, we can infer that if $\sigma > \lambda$, then $|\mathcal{F}_\sigma(\mathcal{D})| \leq |\mathcal{F}_\lambda(\mathcal{D})|$, i.e., the number of frequent itemsets decreases with increase in support thresholds. However, this nice *antimonotonicity* property does not hold for maximal frequent itemsets in general, as illustrated by the example below.

Example 7 Consider the database \mathcal{D} shown in Figure 1. We have the following maximal frequent itemsets at different support thresholds.

$$\begin{aligned}\mathcal{M}_1(\mathcal{D}) &= \{\{x_1, x_2, x_3, x_4\}\} \\ \mathcal{M}_2(\mathcal{D}) &= \{\{x_1, x_2, x_3\}, \{x_3, x_4\}\} \\ \mathcal{M}_3(\mathcal{D}) &= \{\{x_1, x_2\}, \{x_3, x_4\}\} \\ \mathcal{M}_4(\mathcal{D}) &= \{\{x_3\}\}\end{aligned}$$

So $|\mathcal{M}_1(\mathcal{D})| = 1$, $|\mathcal{M}_2(\mathcal{D})| = 2$, $|\mathcal{M}_3(\mathcal{D})| = 2$, $|\mathcal{M}_4(\mathcal{D})| = 1$. Clearly, the aforementioned antimonotonicity property does not hold here. \square

From the example above, we can see that maximal frequent itemsets behave rather “randomly”, i.e., there is no well defined correlation between the number of maximal frequent itemsets and their support threshold. This adds much difficulty to our search of a complexity-theoretic characterization for them. The next few lemmas reveal the connections between maximal frequent and maximal occurrent itemsets. Our final formal proof builds on these lemmas.

Lemma 5 Let I and J be two different maximal δ -occurrent itemsets in a database \mathcal{D} . Then neither I nor J is a subset of the other, i.e., $I \not\subseteq J$ and $J \not\subseteq I$.

PROOF. By Definition 4. \square

Lemma 6 Suppose I is a maximal δ -occurrent itemset and J a maximal λ -occurrent itemset in a database \mathcal{D} . If $I \subset J$, then $\delta > \lambda$.

PROOF. Since $I \subset J$, it follows that $\mathcal{D}(I) \supseteq \mathcal{D}(J)$ and so $|\mathcal{D}(I)| \geq |\mathcal{D}(J)|$. Moreover, $\delta = |\mathcal{D}(I)| \neq \lambda = |\mathcal{D}(J)|$ by Lemma 5. It follows that $\delta > \lambda$. \square

Lemma 7 Suppose I is a maximal σ -frequent itemset in a database \mathcal{D} and $f_{\mathcal{D}}(I) = \delta$. Then I is a maximal δ -occurrent itemset.

PROOF. By contradiction. Suppose on the contrary there is an itemset J such that $I \subset J$ and J is δ -occurrent in \mathcal{D} . But $\delta \geq \sigma$ and so J is σ -frequent in \mathcal{D} , a contradiction to the fact that I is maximal σ -frequent in \mathcal{D} . \square

Lemma 8 If I is a maximal δ -occurrent itemset in a database \mathcal{D} , then I is a maximal δ -frequent itemset in \mathcal{D} .

PROOF. By contradiction. First note that $f_{\mathcal{D}}(I) = \delta$ and so I is δ -frequent in \mathcal{D} . Suppose on the contrary there is an itemset J such that $I \subset J$ and J is maximal δ -frequent in \mathcal{D} . Let $f_{\mathcal{D}}(J) = \lambda$. It must be true that $\lambda \geq \delta$. So J must be maximal λ -occurrent in \mathcal{D} by Lemma 7. Since $I \subset J$, it follows that $\delta > \lambda$ by Lemma 6, a contradiction. \square

Proposition 9 Let \mathcal{D} be a database of transactions. Then $\mathcal{M}_{\sigma}(\mathcal{D}) \supseteq \mathcal{C}_{\sigma}(\mathcal{D})$, $|\mathcal{M}_{\sigma}(\mathcal{D})| \geq |\mathcal{C}_{\sigma}(\mathcal{D})|$, $\mathcal{M}_{\sigma}(\mathcal{D}) \subseteq \bigcup_{i=\sigma}^{|\mathcal{D}|} \mathcal{C}_i(\mathcal{D})$, and $|\mathcal{M}_{\sigma}(\mathcal{D})| \leq \sum_{i=\sigma}^{|\mathcal{D}|} |\mathcal{C}_i(\mathcal{D})|$.

PROOF. By Lemmas 7 and 8. \square

Note that Proposition 9 above gives an upper bound on the number of maximal frequent itemsets. However, the claim in Proposition 9 does not always hold with respect to equality, as illustrated by the following example.

Example 8 Consider the database \mathcal{D} in Figure 1. We have the following different categories of itemsets.

$$\begin{aligned} \mathcal{M}_1(\mathcal{D}) &= \{\{x_1, x_2, x_3, x_4\}\} \\ \mathcal{M}_2(\mathcal{D}) &= \{\{x_1, x_2, x_3\}, \{x_3, x_4\}\} \\ \mathcal{C}_1(\mathcal{D}) &= \{\{x_1, x_2, x_3, x_4\}\} \\ \mathcal{C}_2(\mathcal{D}) &= \{\{x_1, x_2, x_3\}\} \\ \mathcal{C}_3(\mathcal{D}) &= \{\{x_1, x_2\}, \{x_3, x_4\}\} \\ \mathcal{C}_4(\mathcal{D}) &= \{\{x_3\}\} \\ \mathcal{C}_5(\mathcal{D}) &= \emptyset \end{aligned}$$

Clearly, $|\mathcal{M}_1(\mathcal{D})| = 1$ but $\sum_{i=1}^{|\mathcal{D}|} |\mathcal{C}_i(\mathcal{D})| = 5$; $|\mathcal{M}_2(\mathcal{D})| = 2$ but $|\mathcal{C}_2(\mathcal{D})| = 1$ and $\sum_{i=2}^{|\mathcal{D}|} |\mathcal{C}_i(\mathcal{D})| = 4$. \square

Next we will present our proof that the problem of counting the number of maximal frequent itemsets is #P-complete. Our proof is based on a Turing reduction from the counting problem for maximal bipartite cliques, which is known to be #P-complete, to the counting problem for maximal frequent itemsets. In Section 2.2 we already show that a one-to-one correspondence can be easily established between bipartite graphs and databases of transactions. Therefore, we can interpret the complexity result regarding maximal bipartite cliques in terms of maximal occurrent itemsets as follows.

Theorem 10 ([42]) The problem of counting the number of maximal bipartite cliques in a given bipartite graph is #P-complete.⁸

Corollary 11 Let \mathcal{D} be a database of transactions. It is #P-complete to compute $\sum_{\delta=1}^{|\mathcal{D}|} |\mathcal{C}_{\delta}(\mathcal{D})|$.⁹

PROOF. By Lemma 3 and Theorem 10. \square

		x_1	x_2	\cdots	x_n	y_1	y_2	\cdots	y_m
$\mathcal{W}_{\mathcal{D}}^R$	r_1	\mathcal{D}				1	1	\cdots	1
	r_2					1	1	\cdots	1
	\vdots					\vdots	\vdots	\cdots	\vdots
	r_m					1	1	\cdots	1
$\mathcal{W}_{\mathcal{D}}^S$	s_1	1	1	\cdots	1	0	1	\cdots	1
	s_2	1	1	\cdots	1	1	0	\cdots	1
	\vdots	\vdots	\vdots	\cdots	\vdots			\ddots	
	s_m	1	1	\cdots	1	1	1	\cdots	0

Fig. 4. Database Transformation Scheme

The key ideas underlying our Turing reduction are as follows. We transform a database of transactions \mathcal{D} to a new database $\mathcal{W}_{\mathcal{D}}$ using a polynomial-time algorithm. The transformation is constructed in such a way that the size of $\mathcal{W}_{\mathcal{D}}$ is polynomial in the size of \mathcal{D} . Moreover, a system of linear equations can be established, in which the different numbers of maximal occurrent itemsets of \mathcal{D} appear as variables, and the different numbers of maximal frequent itemsets of $\mathcal{W}_{\mathcal{D}}$ as constants. Therefore, given a polynomial-time oracle for counting the number of maximal frequent itemsets, we can design a deterministic, polynomial-time algorithm which: (i) first invokes this oracle multiple times to get the different numbers of maximal frequent itemsets in $\mathcal{W}_{\mathcal{D}}$; (ii) then solves the system of linear equations to compute the different numbers of maximal

⁸ This result is not explicitly stated in [42], but follows readily from the results and reductions in [42].

⁹ This is in fact the number of all closed itemsets in \mathcal{D} [38].

occurrent itemsets in \mathcal{D} ; and (iii) finally sum them up to obtain the value of $\sum_{\delta=1}^{|\mathcal{D}|} |\mathcal{C}_\delta(\mathcal{D})|$, which it is already known to be #P-complete to compute.

Now we explain how to transform a database of transactions \mathcal{D} to a new database $\mathcal{W}_\mathcal{D}$. Let $\{t_1, t_2, \dots, t_m\}$ be the set of transactions of \mathcal{D} , $m = |\mathcal{D}|$, and $X = \{x_1, x_2, \dots, x_n\}$ the set of all items appearing in \mathcal{D} . We will introduce a set, $Y = \{y_1, y_2, \dots, y_m\}$, of m new items into $\mathcal{W}_\mathcal{D}$. The new database $\mathcal{W}_\mathcal{D}$ comprises two parts: $\mathcal{W}_\mathcal{D} = \mathcal{W}_\mathcal{D}^R \cup \mathcal{W}_\mathcal{D}^S$, where $\mathcal{W}_\mathcal{D}^R = \{r_1, r_2, \dots, r_m\}$ and $\mathcal{W}_\mathcal{D}^S = \{s_1, s_2, \dots, s_m\}$. The transactions of $\mathcal{W}_\mathcal{D}^R$ and $\mathcal{W}_\mathcal{D}^S$ are constructed as follows: (i) $r_i = t_i \cup Y$ for all $1 \leq i \leq m$, i.e., the transactions of $\mathcal{W}_\mathcal{D}^R$ are obtained by extending each transaction of \mathcal{D} with the entire set Y of new items; (ii) $s_i = X \cup (Y - \{y_i\})$ for all $1 \leq i \leq m$, i.e., each transaction s_i of $\mathcal{W}_\mathcal{D}^S$ contains all the items in X and Y except item y_i . This transformation is illustrated schematically in Figure 4. Note that we can establish a one-to-one correspondence between the transactions in \mathcal{D} and $\mathcal{W}_\mathcal{D}^R$. Clearly, if the size of \mathcal{D} is $O(d)$, then the size of $\mathcal{W}_\mathcal{D}$ is $O(d^2)$.

	x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4	y_5
r_1	1	1	0	0	1	1	1	1	1
r_2	1	1	1	0	1	1	1	1	1
r_3	1	1	1	1	1	1	1	1	1
r_4	0	0	1	1	1	1	1	1	1
r_5	0	0	1	1	1	1	1	1	1
s_1	1	1	1	1	0	1	1	1	1
s_2	1	1	1	1	1	0	1	1	1
s_3	1	1	1	1	1	1	0	1	1
s_4	1	1	1	1	1	1	1	0	1
s_5	1	1	1	1	1	1	1	1	0

Fig. 5. The New Database Transformed from the Database in Figure 2

Example 9 The database shown in Figure 5 is transformed from the database in Figure 2. \square

Lemma 12 Let \mathcal{D} be a database of transactions, $\mathcal{W}_\mathcal{D} = \mathcal{W}_\mathcal{D}^R \cup \mathcal{W}_\mathcal{D}^S$ the new database transformed from \mathcal{D} , X the set of items appearing in \mathcal{D} , Y the set of new items introduced into $\mathcal{W}_\mathcal{D}$. Then for all $I, J \subseteq X$ and $Y_k, Y_z \subseteq Y$: $Y_k = Y_z$ iff $\mathcal{W}_\mathcal{D}^S(I \cup Y_k) = \mathcal{W}_\mathcal{D}^S(J \cup Y_z)$.

PROOF. Let $|\mathcal{D}| = m$ and $Y = \{y_1, y_2, \dots, y_m\}$. Recall that for all $s_i \in \mathcal{W}_\mathcal{D}^S$, $s_i = X \cup (Y - \{y_i\})$. So $\mathcal{W}_\mathcal{D}^S(L \cup Y_x) = \mathcal{W}_\mathcal{D}^S(Y_x) = \{s_i \mid y_i \in (Y - Y_x)\}$ for all $L \subseteq X, Y_x \subseteq Y$. Therefore, $Y_k = Y_z$, iff $Y - Y_k = Y - Y_z$, iff $\mathcal{W}_\mathcal{D}^S(Y_k) = \mathcal{W}_\mathcal{D}^S(Y_z)$, iff $\mathcal{W}_\mathcal{D}^S(I \cup Y_k) = \mathcal{W}_\mathcal{D}^S(J \cup Y_z)$. \square

Proposition 13 Let \mathcal{D} be a database of transactions, $|\mathcal{D}| = m$, $\mathcal{W}_{\mathcal{D}}$ the new database transformed from \mathcal{D} , X the set of items appearing in \mathcal{D} , and Y the set of new items introduced into $\mathcal{W}_{\mathcal{D}}$. If $I \subseteq X$ is a maximal $(\sigma + k)$ -occurrent itemset in \mathcal{D} , where $1 \leq \sigma \leq m$, $0 \leq k \leq m - \sigma$, then $I \cup Y_k$ is a maximal $(\sigma + m)$ -occurrent and maximal $(\sigma + m)$ -frequent itemset in $\mathcal{W}_{\mathcal{D}}$, where Y_k is an arbitrary itemset such that $Y_k \subseteq Y$ and $|Y_k| = k$.

PROOF. Let $\mathcal{W}_{\mathcal{D}} = \mathcal{W}_{\mathcal{D}}^R \cup \mathcal{W}_{\mathcal{D}}^S$, $\mathcal{D}(I) = \{t_1, t_2, \dots, t_{\sigma+k}\}$. It follows that $\mathcal{W}_{\mathcal{D}}^R(I) = \{r_1, r_2, \dots, r_{\sigma+k}\}$, where each r_i , $1 \leq i \leq \sigma + k$, is obtained from t_i by adding the entire set Y of new items. Since $Y_k \subseteq Y$, it follows that $\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k) = \mathcal{W}_{\mathcal{D}}^R(I)$. Recall that for all $s_i \in \mathcal{W}_{\mathcal{D}}^S$, $s_i = X \cup (Y - \{y_i\})$. So $\mathcal{W}_{\mathcal{D}}^S(I \cup Y_k) = \mathcal{W}_{\mathcal{D}}^S(Y_k) = \{s_i \mid y_i \in (Y - Y_k)\}$. Since $|Y_k| = k$ and $|Y| = m$, so $|\mathcal{W}_{\mathcal{D}}^S(Y_k)| = m - k$. Clearly, $\mathcal{W}_{\mathcal{D}}(I \cup Y_k) = \mathcal{W}_{\mathcal{D}}^R(I \cup Y_k) \cup \mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)$. Therefore, $f_{\mathcal{W}_{\mathcal{D}}}(I \cup Y_k) = |\mathcal{W}_{\mathcal{D}}(I \cup Y_k)| = (\sigma + k) + (m - k) = \sigma + m$. Thus $I \cup Y_k$ is a $(\sigma + m)$ -occurrent itemset in $\mathcal{W}_{\mathcal{D}}$.

Next we will show that $I \cup Y_k$ is a maximal $(\sigma + m)$ -occurrent itemset in $\mathcal{W}_{\mathcal{D}}$. Suppose on the contrary there is an itemset U such that $(I \cup Y_k) \subset U$ and U is $(\sigma + m)$ -occurrent in $\mathcal{W}_{\mathcal{D}}$. Let $U = J \cup Y_z$, where $I \subseteq J \subseteq X$ and $Y_k \subseteq Y_z \subseteq Y$. By Lemma 2, $\mathcal{W}_{\mathcal{D}}(U) = \mathcal{W}_{\mathcal{D}}(I \cup Y_k) = \mathcal{W}_{\mathcal{D}}^R(I \cup Y_k) \cup \mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)$. It follows that $\mathcal{W}_{\mathcal{D}}^R(J \cup Y_z) = \mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)$ and $\mathcal{W}_{\mathcal{D}}^S(J \cup Y_z) = \mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)$. So $Y_z = Y_k$ by Lemma 12. Then it must be the case that $I \subset J$, because $(I \cup Y_k) \subset (J \cup Y_z)$. But $\mathcal{W}_{\mathcal{D}}^R(J \cup Y_z) = \mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)$. So J must be $(\sigma + k)$ -occurrent in \mathcal{D} , which contradicts the fact that I is a maximal $(\sigma + k)$ -occurrent itemset in \mathcal{D} .

Therefore, $I \cup Y_k$ must be a maximal $(\sigma + m)$ -occurrent itemset in $\mathcal{W}_{\mathcal{D}}$ and hence a maximal $(\sigma + m)$ -frequent itemset in $\mathcal{W}_{\mathcal{D}}$ by Lemma 8. \square

Proposition 14 Let \mathcal{D} be a database of transactions, $|\mathcal{D}| = m$, $\mathcal{W}_{\mathcal{D}}$ the new database transformed from \mathcal{D} , X the set of items appearing in \mathcal{D} , and Y the set of new items introduced into $\mathcal{W}_{\mathcal{D}}$. Suppose U is a maximal $(\sigma + m)$ -frequent itemset in $\mathcal{W}_{\mathcal{D}}$, where $1 \leq \sigma \leq m$, such that $U = I \cup Y_k$, $I \subseteq X$, $Y_k \subseteq Y$, $|Y_k| = k$. Then it must be the case that $0 \leq k \leq m - \sigma$, I is a maximal $(\sigma + k)$ -occurrent itemset in \mathcal{D} , and U is a maximal $(\sigma + m)$ -occurrent itemset in $\mathcal{W}_{\mathcal{D}}$.

PROOF. Let $\mathcal{W}_{\mathcal{D}} = \mathcal{W}_{\mathcal{D}}^R \cup \mathcal{W}_{\mathcal{D}}^S$. Note that $\mathcal{W}_{\mathcal{D}}(U) = \mathcal{W}_{\mathcal{D}}^R(I \cup Y_k) \cup \mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)$ and $\mathcal{W}_{\mathcal{D}}^S(I \cup Y_k) = \{s_i \mid y_i \in (Y - Y_k)\}$. So $|\mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)| = m - k$. Moreover, $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| \leq m$. Since U is a maximal $(\sigma + m)$ -frequent itemset in $\mathcal{W}_{\mathcal{D}}$, so $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| + |\mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)| \geq \sigma + m$. It follows that $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| \geq \sigma + k$. Thus $0 \leq k \leq m - \sigma$.

We will now show that $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| = \sigma + k$. Suppose on the contrary $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| \geq \sigma + k + 1$. It must be the case that $Y - Y_k \neq \emptyset$, $|Y - Y_k| \leq m - 1$; otherwise, $Y_k = Y$, $|\mathcal{W}_{\mathcal{D}}^S(I \cup Y_k)| = \emptyset$, $|\mathcal{W}_{\mathcal{D}}^R(I \cup Y_k)| \geq \sigma + m$, and so $\sigma = 0$,

a contradiction. Thus for all $y_i \in (Y - Y_k)$, $|\mathcal{W}_D^S(I \cup Y_k \cup \{y_i\})| = m - k - 1$. So $I \cup Y_k \cup \{y_i\}$ must be $(\sigma + m)$ -frequent in \mathcal{W}_D , contradicting the fact that $U = I \cup Y_k$ is a maximal $(\sigma + m)$ -frequent itemset in \mathcal{W}_D . Therefore, $|\mathcal{W}_D^R(I \cup Y_k)| = \sigma + k$ and $I \cup Y_k$ is $(\sigma + m)$ -occurent in \mathcal{W}_D .

Next we will show that I is a maximal $(\sigma + k)$ -occurent itemset in \mathcal{D} . Suppose on the contrary there is an itemset $J \subseteq X$ such that $I \subset J$ and J is a $(\sigma + k)$ -occurent itemset in \mathcal{D} . Then $\mathcal{D}(I) = \mathcal{D}(J)$ by Lemma 2. It follows that $\mathcal{W}_D^R(I \cup Y_k) = \mathcal{W}_D^R(J \cup Y_k)$. Moreover, $\mathcal{W}_D^S(I \cup Y_k) = \mathcal{W}_D^S(J \cup Y_k)$ by Lemma 12. So $\mathcal{W}_D(J \cup Y_k) = \mathcal{W}_D(I \cup Y_k)$ and $J \cup Y_k$ is also $(\sigma + m)$ -frequent in \mathcal{W}_D . But $U = (I \cup Y_k) \subset (J \cup Y_k)$, which contradicts the fact that U is maximal $(\sigma + m)$ -frequent in \mathcal{W}_D .

To show that U is maximal $(\sigma + m)$ -occurent in \mathcal{W}_D , suppose on the contrary there is a $(\sigma + m)$ -occurent itemset $J \cup Y_z$ in \mathcal{W}_D such that $U \subset (J \cup Y_z)$, $I \subseteq J$, $Y_k \subseteq Y_z$. Then $\mathcal{W}_D(U) = \mathcal{W}_D(I \cup Y_k) = \mathcal{W}_D^R(I \cup Y_k) \cup \mathcal{W}_D^S(I \cup Y_k)$ by Lemma 2. Thus $\mathcal{W}_D^R(J \cup Y_z) = \mathcal{W}_D^R(I \cup Y_k)$ and $\mathcal{W}_D^S(J \cup Y_z) = \mathcal{W}_D^S(I \cup Y_k)$. So $Y_z = Y_k$ by Lemma 12. Then it must be the case that $I \subset J$, because $(I \cup Y_k) = U \subset (J \cup Y_z)$. But $\mathcal{W}_D^R(J \cup Y_z) = \mathcal{W}_D^R(I \cup Y_k)$. It follows that J must be $(\sigma + k)$ -occurent in \mathcal{D} , a contradiction to the fact that I is a maximal $(\sigma + k)$ -occurent itemset in \mathcal{D} . \square

Proposition 15 Let \mathcal{D} be a database of transactions, $|\mathcal{D}| = m$, \mathcal{W}_D the new database transformed from \mathcal{D} , and $1 \leq \sigma \leq m$. Then U is a maximal $(\sigma + m)$ -frequent itemset in \mathcal{W}_D iff U is a maximal $(\sigma + m)$ -occurent itemset in \mathcal{W}_D .

PROOF. Let X be the set of items appearing in \mathcal{D} and Y the set of new items introduced into \mathcal{W}_D . If U is a maximal $(\sigma + m)$ -frequent itemset in \mathcal{W}_D , then U is a maximal $(\sigma + m)$ -occurent itemset in \mathcal{W}_D , by Proposition 14.

On the other hand, suppose U is a maximal $(\sigma + m)$ -occurent itemset in \mathcal{W}_D , $U = I \cup Y_k$, where $I \subseteq X$, $Y_k \subseteq Y$, $|Y_k| = k$. Next we will show that I is maximal $(\sigma + k)$ -occurent in \mathcal{D} . Note that $\mathcal{W}_D(U) = \mathcal{W}_D^R(U) \cup \mathcal{W}_D^S(U)$ and $\mathcal{W}_D^S(I \cup Y_k) = \{s_i \mid y_i \in (Y - Y_k)\}$. So $|\mathcal{W}_D^S(I \cup Y_k)| = m - k$ and hence $|\mathcal{W}_D^R(I \cup Y_k)| = (\sigma + m) - (m - k) = \sigma + k$. Therefore, I is $(\sigma + k)$ -occurent in \mathcal{D} . Suppose on the contrary I is not a maximal $(\sigma + k)$ -occurent itemset in \mathcal{D} . Then there is a $(\sigma + k)$ -occurent itemset $J \subseteq X$ such that $I \subset J$. It follows that $\mathcal{D}(I) = \mathcal{D}(J)$ by Lemma 2. So $\mathcal{W}_D^R(J \cup Y_k) = \mathcal{W}_D^R(I \cup Y_k)$. By Lemma 12, $\mathcal{W}_D^S(J \cup Y_k) = \mathcal{W}_D^S(I \cup Y_k)$. It follows that $J \cup Y_k$ is $(\sigma + m)$ -occurent in \mathcal{W}_D and $U = (I \cup Y_k) \subset (J \cup Y_k)$, which contradicts the fact that U is a maximal $(\sigma + m)$ -occurent itemset in \mathcal{W}_D . Therefore, I must be maximal $(\sigma + k)$ -occurent in \mathcal{D} . So $U = I \cup Y_k$ must be a maximal $(\sigma + m)$ -frequent itemset in \mathcal{W}_D by Propositions 13. \square

Proposition 16 Let \mathcal{D} be a database of transactions, $|\mathcal{D}| = m$, $\mathcal{W}_{\mathcal{D}}$ the new database transformed from \mathcal{D} . Then for all $1 \leq \sigma \leq m$:

$$|\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})| = \sum_{k=0}^{m-\sigma} |\mathcal{C}_{\sigma+k}(\mathcal{D})| \cdot \binom{m}{k}$$

PROOF. Let X be the set of items appearing in \mathcal{D} and Y the set of new items introduced into $\mathcal{W}_{\mathcal{D}}$. Note that $X \cap Y = \emptyset$ and $|Y| = m$. For any k such that $0 \leq k \leq m - \sigma$, let $\mathcal{A}_{\sigma+k}(\mathcal{D}) = \{I \cup Y_k \mid I \in \mathcal{C}_{\sigma+k}(\mathcal{D}), Y_k \subseteq Y, |Y_k| = k\}$. Then $|\mathcal{A}_{\sigma+k}(\mathcal{D})| = |\mathcal{C}_{\sigma+k}(\mathcal{D})| \cdot \binom{m}{k}$. Next we show $\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}}) = \bigcup_{k=0}^{m-\sigma} \mathcal{A}_{\sigma+k}(\mathcal{D})$. First, if $U \in \mathcal{A}_{\sigma+k}(\mathcal{D})$, where $0 \leq k \leq m - \sigma$, then $U \in \mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})$ by Proposition 13. So $\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}}) \supseteq \bigcup_{k=0}^{m-\sigma} \mathcal{A}_{\sigma+k}(\mathcal{D})$. Second, if $U \in \mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})$, then by Proposition 14, it must be the case that $U = I \cup Y_k$, where $I \subseteq X$, $Y_k \subseteq Y$, $|Y_k| = k$, $0 \leq k \leq m - \sigma$, and I is a maximal $(\sigma + k)$ -occurrent itemset in \mathcal{D} , i.e., $I \in \mathcal{C}_{\sigma+k}(\mathcal{D})$. So $U \in \mathcal{A}_{\sigma+k}(\mathcal{D})$ and $\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}}) \subseteq \bigcup_{k=0}^{m-\sigma} \mathcal{A}_{\sigma+k}(\mathcal{D})$. Therefore, $\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}}) = \bigcup_{k=0}^{m-\sigma} \mathcal{A}_{\sigma+k}(\mathcal{D})$. Note that $\mathcal{C}_{\delta}(\mathcal{D}) \cap \mathcal{C}_{\lambda}(\mathcal{D}) = \emptyset$ for all $\delta \neq \lambda$. So $\mathcal{A}_{\sigma+i}(\mathcal{D}) \cap \mathcal{A}_{\sigma+j}(\mathcal{D}) = \emptyset$ for all $i \neq j$. It follows that $|\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})| = \sum_{k=0}^{m-\sigma} |\mathcal{A}_{\sigma+k}(\mathcal{D})| = \sum_{k=0}^{m-\sigma} |\mathcal{C}_{\sigma+k}(\mathcal{D})| \cdot \binom{m}{k}$. \square

Theorem 17 Let \mathcal{D} be a database of transactions, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent itemsets in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete.

PROOF. Recall that it is #P-complete to count the number $\sum_{\sigma=1}^m |\mathcal{C}_{\sigma}(\mathcal{D})|$, by Corollary 11. We will show how this counting problem can be Turing reduced to the counting problem for maximal frequent itemsets, thereby proving that the latter is #P-hard.

Let $\mathcal{W}_{\mathcal{D}}$ be the new database transformed from \mathcal{D} . For all $1 \leq \sigma \leq m$, let $C_{\sigma} = |\mathcal{C}_{\sigma}(\mathcal{D})|$ and $M_{\sigma} = |\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})|$. Then by Proposition 16, we can construct the following system of linear equations, represented in matrix notation:

$$\begin{pmatrix} 1 & \binom{m}{1} & \binom{m}{2} & \cdots & \binom{m}{m-1} \\ 0 & 1 & \binom{m}{1} & \cdots & \binom{m}{m-2} \\ 0 & 0 & 1 & \cdots & \binom{m}{m-3} \\ & & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_m \end{pmatrix} = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \\ \vdots \\ M_m \end{pmatrix}$$

Clearly, given the values of (M_1, M_2, \dots, M_m) , the system of linear equations above can be solved, i.e., there is a unique solution for (C_1, C_2, \dots, C_m) . Moreover, $\binom{m}{k} \leq m^m$ for all $0 \leq k \leq m - 1$. So $\binom{m}{k}$ can be stored using $O(m \log_2 m)$ bits in binary notation and computed in time polynomial in m .

Assume that we use bipartite graphs to represent databases. Suppose the size of \mathcal{D} is $O(d)$. Then the size of $\mathcal{W}_{\mathcal{D}}$ is $O(d^2)$. So $m = O(d)$ and all $\binom{m}{k}$ can be stored using $O(d \log_2 d)$ bits in binary notation and computed in time polynomial in d . For all $1 \leq \sigma \leq m$, $C_\sigma = |\mathcal{C}_\sigma(\mathcal{D})| = O(2^d)$. So all C_σ can be stored using $O(d)$ bits in binary notation. Moreover, the size of $\mathcal{W}_{\mathcal{D}}$ is $O(d^2)$ and so $M_\sigma = |\mathcal{M}_{\sigma+m}(\mathcal{W}_{\mathcal{D}})| = O(2^{d^2})$ for all $1 \leq \sigma \leq m$. It follows that all M_σ can be represented using $O(d^2)$ bits in binary notation. Therefore, given the values of (M_1, M_2, \dots, M_m) , the linear equations above can be solved and hence the values of (C_1, C_2, \dots, C_m) can be computed in time polynomial in d . Note that the size of $\mathcal{W}_{\mathcal{D}}$ is $O(d^2)$. So if there is a polynomial-time oracle for counting the number of maximal frequent itemsets, then the values of (M_1, M_2, \dots, M_m) can be computed in time polynomial in d . Hence the values of (C_1, C_2, \dots, C_m) and the number $\sum_{\sigma=1}^m |\mathcal{C}_\sigma(\mathcal{D})| = \sum_{\sigma=1}^m C_\sigma$ can be computed in time polynomial in d . This proves that the counting problem for maximal frequent itemsets is #P-hard.

Clearly, checking whether an itemset is maximal frequent or not can be done in polynomial time. Therefore, the counting problem for maximal frequent itemsets is in #P. This completes our proof. \square

5 Complex Patterns

A large number of data mining problems dealing with frequent patterns can be viewed as instances of the *theory extraction* problem [12]. In general, every transaction in a database \mathcal{D} is considered as a (large) pattern. A partial order, \preceq , can be defined on all patterns such that the support of a pattern p can be formalized as follows: $f_{\mathcal{D}}(p) \stackrel{\text{def}}{=} |\{t \mid p \preceq t, t \in \mathcal{D}\}|$.¹⁰ A pattern whose support exceeds a preset threshold is called a frequent pattern. Note that this partial order, \preceq , preserves the *downward closure* property, i.e., given any patterns p_1 and p_2 , if $p_1 \preceq p_2$ and p_2 is frequent, so is p_1 . We will write $p_1 \prec p_2$ if $p_1 \preceq p_2$ and $p_1 \neq p_2$. Hence a frequent pattern p is maximal if there is no frequent pattern q such that $p \prec q$.

Many problems of mining maximal frequent patterns fall into the above category of generalization. For example, in the problem of mining maximal frequent itemsets, the patterns of interest are sets of items and the partial order is defined on subset inclusion. In this section, we will extend our complexity

¹⁰ This kind of support is called *unweighted* support, in which every database transaction contributes at most one count to the support of a pattern. However, our complexity results can be easily extended to data mining problems that use *weighted support* [32], which takes into account multiple occurrences of a pattern in a database transaction.

analysis to several problems of mining maximal frequent patterns, in which the patterns of interest are subsequences, subtrees, or subgraphs. In the following, we will just define the data structures used in these problems and specify the partial orders on patterns. Our goal is to prove the computational complexity of the associated counting problems for these maximal frequent patterns.

5.1 Subsequences

In problems of mining frequent sequences [23–29], each database transaction is considered as a sequence (string) instead of a set, in which the order of symbols appearing in a sequence is important. Normally the patterns of interest are *subsequences*. The partial order, \preceq , on any two sequences, s_1 and s_2 , is defined as follows: $s_1 \preceq s_2$ iff s_1 is a subsequence of s_2 , i.e., s_1 can be obtained from s_2 by removing zero or more symbols from s_2 . For example, ac is a subsequence of abc . Our complexity result about mining maximal frequent subsequences is stated in Theorem 18.

Theorem 18 Let \mathcal{D} be a database of sequences, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subsequences in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete.

PROOF. To check if a sequence is a subsequence of another sequence takes only linear time. So checking if a sequence is a maximal frequent subsequence in a database of sequences can be done in polynomial time — it suffices to check that this sequence is frequent and expanding it with any one symbol results in its support below the preset threshold. It follows that the problem of counting the number of maximal frequent subsequences belongs to #P. To show that this problem is #P-hard, we will reduce to it the problem of counting the number of maximal frequent itemsets.

Let \mathcal{D}' be a database of transactions in which each transaction contains a set of items. We will construct a database, \mathcal{D} , of sequences from \mathcal{D}' as follows. First, define an arbitrary total order for all the items in \mathcal{D}' . Then sort the items in each transaction of \mathcal{D}' accordingly. Each such sorted sequence of items will become one transaction of \mathcal{D} . Let I denote an itemset and $s(I)$ the corresponding sorted sequence of items in I . Clearly, I is a maximal σ -frequent itemset in \mathcal{D}' iff $s(I)$ is a maximal σ -frequent subsequence in \mathcal{D} . \square

5.2 Labeled Subtrees and Subgraphs

Mining frequent patterns from databases of trees [30–32] and graphs [33–36] has become the focus of intensive research efforts in recent years. Normally,

the patterns of interest are subtrees or subgraphs. A graph, $G = (V, E)$, consists of a set of vertices, V , and a set of edges, $E \subseteq V \times V$. Subgraph isomorphism can be viewed as defining a partial order among graphs. Given two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, G_1 is called a *subgraph* of G_2 , denoted $G_1 \preceq G_2$, if there is an *injective* function $\rho : V_1 \mapsto V_2$ such that for all $(u, v) \in E_1$, $(\rho(u), \rho(v)) \in E_2$ [43]. Moreover, G_1 is called an *induced subgraph* of G_2 if ρ satisfies the following additional condition: for all $(\rho(u), \rho(v)) \in E_2$, $(u, v) \in E_1$. The complexity results to be presented here apply to either subgraphs or induced subgraphs. But for simplicity we will only mention subgraphs.

Note that trees are just a special form of connected, acyclic graphs. The definition of subgraph isomorphism can be readily used to define subtree isomorphism on tree data structures. Definitions of trees, however, usually need to take into account several different factors:

- (1) Rooted or Unrooted. A tree is rooted if there is a distinguished node called the root. In a rooted tree, every node except the root must have a unique parent node. This parent-child relationship is not important in unrooted trees, which are also commonly called free trees.
- (2) Ordered or Unordered. Rooted trees can be either ordered or unordered. For rooted ordered trees, the order of sibling nodes is important whereas it is not for rooted unordered trees.
- (3) Labeled or Unlabeled. Trees and graphs can be either edge-labeled or node-labeled or both. For labeled trees and graphs, subtree and subgraph isomorphism should preserve not only topology but labeling as well.

In this section we will present the complexity results on labeled trees and graphs. The complexity results on unlabeled trees and graphs will be presented separately in the following Section 5.3, in which we will also introduce a new proof technique.

Theorem 19 Let \mathcal{D} be a database of labeled trees, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subtrees in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete. This complexity result holds no matter whether the labeled trees of \mathcal{D} are rooted or unrooted, and in the case of rooted labeled trees, ordered or unordered.

PROOF. Testing of subtree isomorphism can be done in polynomial time [44]. It follows that checking if a subtree is maximal frequent can be done in polynomial time, by simply expanding it and checking for subtree isomorphism. Therefore, the problem of counting the number of maximal frequent subtrees belongs to #P. To show that this problem is #P-hard, we will reduce to it the problem of counting the number of maximal frequent itemsets, which has just been proved to be #P-complete in Section 4.

Let \mathcal{D}' be a database of transactions in which each transaction contains a set of items. We will construct a database, \mathcal{D} , of rooted unordered labeled trees (we will show later how these conditions can be easily relaxed or tightened) from \mathcal{D}' as follows. First, we use $X = \{x_1, x_2, \dots, x_n\}$, where $|X| = n$, to represent the identifiers of items in \mathcal{D}' . Then for each transaction of \mathcal{D}' we construct a two-level tree in \mathcal{D} : (i) its root is labeled with a new identifier x_0 ($x_0 \notin X$); and (ii) for each item x_i in this transaction, create a child node under the root and label it using x_i . For instance, the database of itemsets in Example 1 is transformed to the database of rooted unordered labeled trees shown in Figure 6.

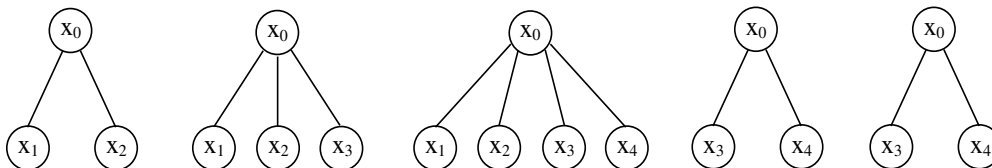


Fig. 6. Rooted Unordered Labeled Trees for the Database in Example 1

We will use the notation, $t(I)$, to denote the tree constructed from I as described above. Clearly, if I is a maximal σ -frequent itemset in \mathcal{D}' , then $t(I)$ is a maximal σ -frequent subtree in \mathcal{D} ; but not vice versa! The only exception is the subtree that contains the single node labeled with x_0 , which is always σ -frequent in \mathcal{D} , regardless of the value of σ and whether \mathcal{D}' has a σ -frequent itemset or not.

But the problem above can be easily fixed. Let $n_{\mathcal{D}'}$ denote the number of maximal σ -frequent itemsets in \mathcal{D}' and $n_{\mathcal{D}}$ the number of maximal σ -frequent subtrees in \mathcal{D} . The following algorithm can be used to compute the value of $n_{\mathcal{D}'}$, given the value of $n_{\mathcal{D}}$. First, if $n_{\mathcal{D}} > 1$, then $n_{\mathcal{D}'} = n_{\mathcal{D}}$, because in this case every maximal σ -frequent subtree in \mathcal{D} corresponds to a maximal σ -frequent itemset in \mathcal{D}' . Second, if $n_{\mathcal{D}} = 1$ (note that $n_{\mathcal{D}} \geq 1$ since the subtree with the single node labeled with x_0 is always σ -frequent), then we need to check whether \mathcal{D}' has any σ -frequent itemset at all. This can be done by scanning \mathcal{D}' once and counting the support of each single item, which only takes linear time. If there is at least one σ -frequent item in \mathcal{D}' , then $n_{\mathcal{D}'} = 1$; otherwise $n_{\mathcal{D}'} = 0$. Thus we have constructed a Turing reduction from the problem of counting the number of maximal frequent itemsets to the problem of counting the number of maximal frequent subtrees.

Note that the claims above remain true even if the trees constructed are not rooted, since in every tree each node has a distinct label, and the node labeled with x_0 is connected to all other nodes which are pairwise disconnected. So conceptually the node labeled with x_0 can be treated as the root. For the case of rooted ordered labeled trees, the proof above can be extended using the same sorting technique as in Theorem 18. \square

We should point out that our complexity results can be readily extended to the problem of mining maximal frequent embedded subtrees [30]. Given two rooted labeled trees, T_1 and T_2 , we will say that T_1 is an *embedded subtree* of T_2 , if there is an *injective* function ρ from the nodes of T_1 to the nodes of T_2 , such that ρ preserves the labels of nodes, and for all nodes u, v of T_1 , if u is the parent of v , then $\rho(u)$ must be an ancestor or the parent of $\rho(v)$ in T_2 (whereas for traditional subtree isomorphism $\rho(u)$ must be the parent of $\rho(v)$ in T_2).

Observe that in the proof of Theorem 19, the height of every tree in \mathcal{D} is one. As a result, the two notions of subtrees and embedded subtrees are essentially equivalent in the context of the database \mathcal{D} so constructed. This observation immediately leads to the #P-hardness of counting the number of maximal frequent embedded subtrees. Moreover, we can also reduce to it the problem of counting the number of maximal frequent subsequences: every sequence can be transformed to a *linear* tree in which the symbols of this sequence are chained together in the original order. One can easily verify that this transformation establishes a one-to-one correspondence between maximal frequent subsequences and maximal frequent embedded subtrees.

Note that checking if a tree is an embedded subtree of another one can be done in polynomial time for ordered trees [45].¹¹ So the problem of counting the number of maximal frequent embedded subtrees is in #P for databases of rooted ordered labeled trees. This is very unlikely, however, for databases of rooted unordered labeled trees, because in such a case it becomes NP-complete to decide if one tree is an embedded subtree of another [45]. We summarize the complexity results for maximal embedded subtrees in the following corollary.

Corollary 20 Let \mathcal{D} be a database of rooted labeled trees, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent embedded subtrees in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete if the trees of \mathcal{D} are ordered, and is #P-hard if the trees of \mathcal{D} are unordered. This complexity holds even if the height of every tree in \mathcal{D} is one, or in each tree of \mathcal{D} every node is incident to at most two edges.

Trees are basically connected, acyclic graphs, and rooted trees can be viewed as directed graphs also. So it is rather straightforward to extend our complexity results for labeled trees to labeled graphs. However, unlike subtree isomorphism, which can be checked in polynomial time, subgraph isomorphism is known to be an NP-complete problem [15]. Consequently, it is very unlikely that the problem of counting the number of maximal frequent subgraphs would belong to #P.

¹¹ This problem is called *tree inclusion* in [45].

Corollary 21 Let \mathcal{D} be a database of labeled graphs, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subgraphs in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-hard. This complexity result holds no matter whether the labeled graphs of \mathcal{D} are directed or undirected.

5.3 Unlabeled Subtrees and Subgraphs

In this section we will present the complexity results for unlabeled trees and graphs. We should note that the complexity results here basically subsume the complexity results for labeled trees and graphs in Section 5.2, since unlabeled trees and graphs can be viewed as labeled using just one label everywhere. However, the complexity results for labeled trees and graphs deserve a standalone discussion because of the simplicity of their proofs. Moreover, in most real-world data mining problems the data is labeled.

Here we will formally prove that the problem of counting the number of maximal frequent unlabeled subtrees is #P-complete. This result is not only of theoretical interest — it reveals a new, interesting proof technique for #P-completeness — but in fact also makes a very strong claim about labeled trees — Theorem 19 remains valid even for the most simplistic form of trees, i.e., binary trees (see Corollary 25).

To give an overview of our new proof technique, let us use D_{Π} to denote the set of instances of a counting problem Π , and $S_{\Pi}(I)$ the set of solutions for an instance $I \in D_{\Pi}$. Given two counting problems, Π and Π' , the most straightforward way of proving that Π Turing reduces to Π' is to show that there is a polynomial-time *parsimonious transformation*, $f : D_{\Pi} \mapsto D_{\Pi'}$, such that $|S_{\Pi}(I)| = |S_{\Pi'}(f(I))|$ for all $I \in D_{\Pi}$ [15]. However, in many cases a parsimonious transformation is very difficult, if not impossible, to construct. So most #P-completeness proofs seen in the literature rely on the very powerful idea of solving a system of linear equations between the solution spaces of two counting problems [13,14,42,46,47] (our proof of Theorem 17 uses this approach).

However, many problems still do not lend themselves easily to this approach, due to the lack of structure in these problems. For such problems our idea is to relax the requirements of a parsimonious transformation: instead of mandating a one-to-one mapping between the two solution spaces $S_{\Pi}(I)$ and $S_{\Pi'}(f(I))$, we allow “dangling” solutions in $S_{\Pi'}(f(I))$ which do not have any counterpart in $S_{\Pi}(I)$; the number of such solutions, however, must be *polynomially bounded*. More formally, a polynomial-time transformation between two counting problems Π and Π' , $f : D_{\Pi} \mapsto D_{\Pi'}$, is called a *semiparsimonious transformation*, if for all $I \in D_{\Pi}$, there is $F \subseteq S_{\Pi'}(f(I))$ such that $|S_{\Pi}(I)| = |S_{\Pi'}(f(I)) - F|$, and

$|F|$ is polynomial and can be computed in polynomial time.¹² For instance, the transformation used in the proof of Theorem 19 is a simple semiparsimonious transformation.

Now we will show a more complicated semiparsimonious transformation from the counting problem for maximal frequent itemsets to the counting problem for maximal frequent unlabeled trees, thereby providing a Turing reduction between these two problems and proving the #P-completeness of the latter. It will soon be clear from the proof of Theorem 24 that this transformation is indeed semiparsimonious.

Let \mathcal{D} be a database of itemsets. We will transform \mathcal{D} to a database, $\mathcal{T}_{\mathcal{D}}$, of *unrooted unlabeled* trees. The key idea is that although each tree created from a transaction of \mathcal{D} is not rooted nor labeled, its topology encodes enough information to allow inference of which item appears in the original transaction. We will use $X = \{x_1, x_2, \dots, x_n\}$, where $|X| = n$, to represent the set of identifiers of items in \mathcal{D} . Given a transaction $I \in \mathcal{D}$, we create a tree from I , denoted $t(I)$, as follows: (i) Build a single chain of $n^3 + \frac{n(n+1)}{2} + 3n^3$ edges. We will fix one end of this chain as if it were the root and (implicitly) number a node on this chain as k if its distance (number of edges on its path) to the root is k ; (ii) For each item $x_i \in I$, create a new node and connect it to the node numbered $\sum_{k=1}^i (n^2 + k)$ on the chain. Note that $t(I)$ is essentially a *binary tree* — every node in $t(I)$ is incident to *at most* three edges. For instance, Figure 7 illustrates the tree constructed from a transaction containing the itemset $\{x_1, x_3\}$.

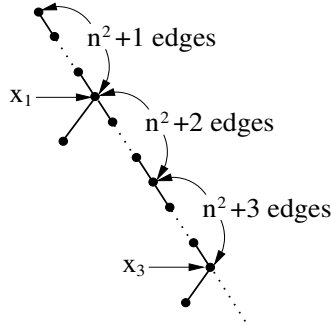


Fig. 7. An Unrooted Unlabeled Tree Created from Itemset $\{x_1, x_3\}$

Observe that trees created this way are skewed in that nodes incident to exactly three edges are all positioned closer to one end of the chain. That is why we can think of this end as the (implicit) root of the entire tree (we will call the other end its tail), although technically the trees so created are not rooted. Moreover, if a node in such a tree is incident to three edges, then this node

¹² We can actually relax this condition and simply require that $|F|$ be computed in polynomial time. The resulting Turing reduction would still have polynomial time complexity.

must be (implicitly) numbered $\sum_{k=1}^i (n^2 + k)$, for some i such that $1 \leq i \leq n$. Therefore, implicitly we could use x_i to label this node. We will often say that a tree branches at x_i if the node (implicitly) labeled with x_i is incident to three edges (which indicates that item x_i appears in the corresponding transaction).

Lemma 22 Let \mathcal{D} be a database of itemsets and $\mathcal{T}_{\mathcal{D}}$ the database of unrooted unlabeled trees constructed from \mathcal{D} . If s is a maximal frequent subtree in $\mathcal{T}_{\mathcal{D}}$, then there must exist a tree $t \in \mathcal{T}_{\mathcal{D}}$, such that s is isomorphic to a subtree of t starting at the (implicit) root of t .

PROOF. By contradiction. Clearly, if this claim is not true, then s could be grown towards the (implicit) root of every tree $g \in \mathcal{T}_{\mathcal{D}}$ in which s is a subtree, a contradiction to the fact that s is maximal frequent. \square

Therefore, by Lemma 22, every maximal frequent subtree in $\mathcal{T}_{\mathcal{D}}$ matches with at least one tree in $\mathcal{T}_{\mathcal{D}}$ starting at its (implicit) root node. So we can refer to a maximal frequent subtree in $\mathcal{T}_{\mathcal{D}}$ as if it had a root and some of its nodes were labeled with x_1, x_2, \dots, x_n . The following lemma shows how some of the maximal frequent subtrees in $\mathcal{T}_{\mathcal{D}}$ may branch.

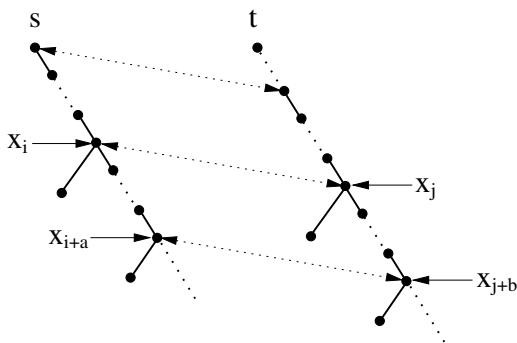


Fig. 8. Subtree Isomorphism

Lemma 23 Let \mathcal{D} be a database of itemsets, $X = \{x_1, x_2, \dots, x_n\}$ the set of items appearing in \mathcal{D} , and $\mathcal{T}_{\mathcal{D}}$ the database of unrooted unlabeled trees constructed from \mathcal{D} . Suppose s is a maximal frequent subtree in $\mathcal{T}_{\mathcal{D}}$ and branches at a node (implicitly) labeled with $x_i \in X$. If s is isomorphic to a subtree of $t \in \mathcal{T}_{\mathcal{D}}$ starting at a node other than the (implicit) root of t , then s can only branch at x_i .

PROOF. By contradiction. Without loss of generality we will assume that the node labeled with x_i in s is the first one at which s branches. Let the node (implicitly) labeled with x_i in s map to a node labeled with x_j in t . Since s is isomorphic to a subtree of t starting at a node other than the (implicit) root of t , it follows that $i < j$. Suppose on the contrary s branches at another node labeled with x_{i+a} , $a \geq 1$. Since s is isomorphic to a subtree of t , it follows that t must branch at x_{j+b} , $b \geq 1$. Such a situation is illustrated in Figure 8.

Therefore, the distance from x_{i+a} to x_i should be equal to the distance from x_{j+b} to x_j , i.e.,

$$\sum_{k=1}^a (n^2 + i + k) = \sum_{k=1}^b (n^2 + j + k)$$

From the equation above we can derive the following:

$$(a - b)n^2 + \frac{(a - b)(a + b + 1)}{2} = bj - ai$$

Note that $i < j$. So it must be the case that $a > b$. It follows that the left hand side of the equation above is strictly greater than n^2 . But $b \leq n$, $j \leq n$. So the right hand side of equation above is strictly less than n^2 . Thus a contradiction follows. \square

We will now present the main complexity result of this section.

Theorem 24 Let \mathcal{D} be a database of unlabeled trees in which every node is incident to at most three edges, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subtrees in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete. This complexity result holds no matter whether the unlabeled trees of \mathcal{D} are rooted or unrooted, and in the case of rooted unlabeled trees, ordered or unordered.

PROOF. Subtree isomorphism can be checked in polynomial time even for unrooted unlabeled trees [44]. It follows that checking if a tree is a maximal frequent subtree can be done in polynomial time. Therefore, the problem of counting the number of maximal frequent unlabeled subtrees belongs to #P. To show that this problem is #P-hard, we will reduce to it the problem of counting the number of maximal frequent itemsets.

Let \mathcal{D}' be a database of itemsets, $X = \{x_1, x_2, \dots, x_n\}$ the set of items in \mathcal{D}' , $|X| = n$, and $\mathcal{D} = \mathcal{T}_{\mathcal{D}'}$ the database of unrooted unlabeled trees constructed from \mathcal{D}' as described above. We will use $t(I)$ to denote the tree constructed from an itemset $I \in \mathcal{D}'$. Clearly, if I is a maximal σ -frequent itemset in \mathcal{D}' , then $t(I)$ is a maximal σ -frequent subtree in \mathcal{D} . However, it is not necessarily true vice versa. Let $N_{\mathcal{D}'}$ be the set of maximal σ -frequent itemsets in \mathcal{D}' , $N_{\mathcal{D}}$ the set of maximal σ -frequent subtrees in \mathcal{D} , and $F_{\mathcal{D}} \subseteq N_{\mathcal{D}}$ the set of maximal σ -frequent subtrees in \mathcal{D} which do not correspond to any maximal σ -frequent itemset in \mathcal{D}' . Clearly, $|N_{\mathcal{D}'}| = |N_{\mathcal{D}}| - |F_{\mathcal{D}}|$. Next we will prove that $|F_{\mathcal{D}}| = O(n)$ and $F_{\mathcal{D}}$ can be computed in polynomial time (which implies that our transformation is indeed semiparsimonious).

Let $X_{\sigma} = \{x \mid x \in X, \{x\} \text{ is } \sigma\text{-frequent in } \mathcal{D}'\}$ and $\overline{X}_{\sigma} = X - X_{\sigma}$. Note that both X_{σ} and \overline{X}_{σ} can be computed in polynomial time by scanning the database \mathcal{D}' once. Suppose $s \in F_{\mathcal{D}}$ and s branches somewhere. Then by Lemma 22, we can refer to s as if it had a root and some of its nodes had

labels in X . Since s does not correspond to any maximal σ -frequent itemset in \mathcal{D}' , there must exist a tree $t \in \mathcal{D}$ such that s is isomorphic to a subtree of t starting at a node other than the root of t ; otherwise, for all the trees of \mathcal{D} in which s is a subtree, s would be isomorphic to the subtree rooted at their root and so s would correspond to some maximal σ -frequent itemset in \mathcal{D} . It follows that s can only branch at some $x_i \in X$, by Lemma 23. So s must consist of a single chain of $(d+e)$ edges which branches at the node numbered d , such that $d = \sum_{k=1}^i (n^2 + k)$, $e \geq 3n^3$, $d + e < n^3 + \frac{n(n+1)}{2} + 3n^3$. Then it must be the case that $x_i \in \overline{X}_\sigma$; otherwise, if $x_i \in X_\sigma$, then s would be a proper subtree of the frequent subtree $t(\{x_i\})$ and hence not maximal. Therefore, if $s \in F_{\mathcal{D}}$, then s can branch at some $x_i \in \overline{X}_\sigma$ only. It follows that $|F_{\mathcal{D}}| = O(n)$.

We will now show how to compute $F_{\mathcal{D}}$. Let \bar{t}_0 denote the single chain of $n^3 + \frac{n(n+1)}{2} + 3n^3$ edges, and \bar{t}_i the single chain of $3n^3 + \sum_{k=1}^i (n^2 + k)$ edges which branches at x_i , where $1 \leq i \leq n$. Based on our discussion above, we know that if $s \in F_{\mathcal{D}}$, then s must be either \bar{t}_0 , or can be obtained by adding a single chain of zero or more edges to the tail (not the implicit root) of some \bar{t}_i , such that $x_i \in \overline{X}_\sigma$. Therefore, we can use the following algorithm to compute $F_{\mathcal{D}}$. For each $x_i \in \overline{X}_\sigma$, we first check if \bar{t}_i is σ -frequent in \mathcal{D} . If yes, then we keep adding one edge to the current tail of \bar{t}_i as long as it still remains σ -frequent. After this we check whether the current \bar{t}_i (already maximally expanded at its tail) would still remain σ -frequent if we add one edge to its (implicit) root. If not, then it must belong to $F_{\mathcal{D}}$. Otherwise, the current \bar{t}_i is not maximal and hence cannot belong to $F_{\mathcal{D}}$. Note that \bar{t}_0 is maximal σ -frequent iff $X_\sigma = \emptyset$, which can be checked in polynomial time. Moreover, checking if a tree is σ -frequent in \mathcal{D} can be done in polynomial time. Therefore, $F_{\mathcal{D}}$ can be computed using the algorithm just described and so counted in polynomial time. It follows that the problem of counting the number of maximal frequent itemsets can be Turing reduced to the problem of counting the number of maximal frequent unlabeled subtrees.

Note that the trees of \mathcal{D} are not rooted, but can be used as if they were rooted in the proof above, thanks to our encoding scheme. Therefore, our claims still hold if the trees are actually rooted. We make no assumption about whether the trees are ordered or not — either way our claims remain valid. \square

From Theorem 24 we can immediately derive the following claims about labeled trees and unlabeled graphs.

Corollary 25 Let \mathcal{D} be a database of labeled trees in which every node is incident to at most three edges, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subtrees in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-complete. This complexity result holds even if in each tree of \mathcal{D} every node has a distinct label (but different trees of \mathcal{D} share labels), and no matter whether the trees of \mathcal{D} are rooted or unrooted, and in the case of rooted trees, ordered or unordered.

Corollary 26 Let \mathcal{D} be a database of unlabeled graphs, $|\mathcal{D}| = m$. The problem of counting the number of maximal σ -frequent subgraphs in \mathcal{D} , where $1 \leq \sigma \leq m$, is #P-hard. This complexity result holds no matter whether the unlabeled graphs of \mathcal{D} are directed or undirected.

6 Related Work

Valiant introduced the class #P of counting problems and proved that it is #P-complete to count the number of distinct perfect matchings in a bipartite graph [13] — the first counting problem known to be #P-complete whose associated decision problem can be solved in polynomial time. In [42], many counting problems on bipartite graphs, such as vertex cover, independent set, were proved to be #P-complete. The #P-completeness of counting the number of maximal bipartite cliques in a bipartite graph follows readily from the results in [42], although it was not explicitly stated there. In [48], it was proved that it is NP-complete to decide whether there is a maximal bipartite $(k, *)$ -clique in a bipartite graph. More recently, Hunt et al. proved the #P-hardness of many graph counting problems when restricted to planar instances [46]. Some of these results were later extended by Vadhan to even more restricted bipartite graphs of bounded degree [47].

Several algorithms were proposed in the literature for mining maximal frequent itemsets, such as MaxClique [16], Dualize and Advance [12], Max-Miner [18], Pincer-Search [17], MAFIA [20], GenMax [21], and DepthProject [19]. These algorithms exploited different heuristics for optimization and were shown to have different good scaleup characteristics on certain benchmark datasets. There is also a large body of work on mining frequent and maximal frequent patterns from complex data structures, such as sequences [23–29,38], trees [30–32], and graphs [33–36]. However, formal analysis of computational complexity was not the main focus of these works. Our new complexity results (summarized in Table 2) provide a complexity-theoretic characterization of the problems studied in these works.

The problem of counting the number of frequent (but not maximal frequent) itemsets was first shown to be #P-complete in [12]. In [12], it was also shown that it is NP-complete to decide if there is a maximal σ -frequent itemset with at least k items. The NP-hardness of mining maximal frequent itemsets was also recently established in [22] by proving the following claim: given a set of maximal frequent itemsets, it is NP-complete to decide whether this set can be grown with new maximal frequent itemsets. We should note, however, that these results do not directly lend themselves to the #P-completeness of counting the number of maximal frequent itemsets.

Finally, the work of [37] aimed at providing a lattice-theoretic framework for mining frequent itemsets and association rules. Interesting work was also recently reported in [49] on characterization of length distributions of frequent and maximal frequent itemset collections, with a focus on computing tight bounds for feasible distribution. Neither of these two works subsumes any of the complexity results proved in this paper. Moreover, our results on the complexity of counting maximal frequent itemsets provide theoretical underpinnings for the algorithms proposed in [49] for computing distribution.

7 Discussion and Conclusion

In this paper we study the complexity of mining maximal frequent patterns, from the perspective of counting the number of solutions. We present the first formal proof that the problem of counting the number of maximal frequent itemsets is $\#P$ -complete, thereby providing a complexity-theoretic explanation for the (worst-case) computational difficulty of this problem. We also extend our complexity analysis to other data mining problems dealing with complex data structures, in which the patterns of interest are maximal frequent subsequences, subtrees, or subgraphs. The complexity results proved in this paper are summarized in Table 2. To the best of our knowledge, our work is the first comprehensive study on the computational complexity of mining maximal frequent patterns.

We should point out that there are four different but closely related computational aspects of data mining problems:

- (1) Enumeration Problem (Column 2 of Table 2). The expected output is an explicit presentation of all solutions.
- (2) Counting Problem (Column 3 of Table 2). The goal is to compute the number of solutions.
- (3) Search Problem (Column 4 of Table 2). Instead of all solutions, output of one solution is desired, if there is any.
- (4) Decision Problem (Column 5 of Table 2). The primary concern is about the existence of any solution.

Take as an example the problem of mining maximal frequent itemsets. Its associated search problem is to output one maximal frequent itemset, if there is any, while the decision problem is to answer the question of whether a maximal frequent itemset exists.

maximal frequent patterns	enumeration	counting	search	decision
substrings	P	P	P	P
itemsets	NP-hard	#P-complete	P	P
subsequences	NP-hard	#P-complete	P	P
subtrees ^a	NP-hard	#P-complete	P	P
subgraphs ^b	NP-hard	#P-hard	FP ^{NP}	P

^a The same complexity result holds for trees that are either rooted or unrooted, ordered or unordered, labeled or unlabeled (with/without duplicate labels).

^b The same complexity result holds for graphs that are either directed or undirected, labeled or unlabeled (with/without duplicate labels).

Table 2. Summary of Complexity Results

These four different aspects of data mining problems in fact exhibit different levels of computational complexity (shown in Table 2). For all the problems we study in this paper, their associated decision problems (whether a maximal frequent pattern exists) can all be solved in polynomial time (even for graphs). Their search problems can also be solved in polynomial time except the search for a maximal frequent subgraph. This is due to the computational difficulty of testing for subgraph isomorphism, which is NP-complete [15]. Nevertheless, one can easily design a deterministic polynomial time algorithm to compute a maximal frequent subgraph, given an oracle for solving subgraph isomorphism: start with a graph with one node and grow it until the subgraph isomorphism test fails. This implies that the complexity of searching for a maximal frequent subgraph is FP^{NP} [41]. For the same reason, it is unlikely that the counting problem for maximal frequent subgraphs would belong to #P; but we have proved that it is #P-hard. Finally, we should point out that the NP-hardness of enumeration problems can be readily derived from the #P-hardness of their associated counting problems [15]. Also note that the problem of mining maximal frequent substrings can be efficiently solved in polynomial time, utilizing the data structure of generalized suffix trees [39] — this is not really surprising since substrings do not manifest a combinatorial nature.

The complexity results in this paper should be interpreted as *worst-case* time complexity only — the implication being that there is little hope a data mining algorithm can execute efficiently on *any* dataset (if the problem is #P-hard or NP-hard). In recent years many data mining algorithms have been developed for important applications. Most of them have been shown to be efficient or even exhibit linear scaleup property with respect to various benchmark datasets, either synthetic or from real-world applications. A different analysis tool will be needed to provide a complexity-theoretic explanation for the efficiency of these algorithms and datasets. Recently interesting work was reported in [49] on characterization of length distributions of frequent and

maximal frequent itemset collections. We believe research along this line will provide us with good guidance on understanding the algorithms themselves as well as the datasets upon which the algorithms are tested.

Another important problem is concerned with data mining algorithms that can “adapt” efficiently — if the size of the output is polynomial, then the algorithm runs in polynomial time — the so-called *output polynomial* algorithms [50]. Recently, in [12], a mildly subexponential algorithm was developed for mining maximal frequent itemsets. But currently it is still an open problem whether an output polynomial algorithm exists for mining maximal frequent itemsets. Our complexity results do not address this problem and we believe that different complexity analysis techniques need to be developed to solve this open problem.

Acknowledgements

The author would like to thank Leslie G. Valiant, Alan Selman, Heikki Mannila, Mitsunori Ogihara, Mohammed Javeed Zaki, Jian Pei, and Yongqiao Xiao for their helpful comments.

References

- [1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: International Conference on Very Large Data Bases (VLDB), 1994.
- [2] M. Richardson, P. Domingos, Mining knowledge-sharing sites for viral marketing, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002.
- [3] O. Etzioni, R. Tuchinda, C. A. Knoblock, A. Yates, To buy or not to buy: Mining airfare data to minimize ticket purchase price, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003.
- [4] R. C.-W. Wong, A. W.-C. Fu, K. Wang, MPIS: Maximal-profit item selection with cross-selling considerations, in: IEEE International Conference on Data Mining (ICDM), 2003.
- [5] Q. Yang, H. Cheng, Mining plans for customer-class transformation, in: IEEE International Conference on Data Mining (ICDM), 2003.
- [6] K. Sequeira, M. J. Zaki, ADMIT: Anomaly-based data mining for intrusions, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002.

- [7] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, J. Srivastava, A comparative study of anomaly detection schemes in network intrusion detection, in: SIAM International Conference on Data Mining (SDM), 2003.
- [8] M. V. Mahoney, P. K. Chan, Learning nonstationary models of normal network traffic for detecting novel attacks, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002.
- [9] J. Hu, X. Shen, Y. Shao, C. Bystroff, M. J. Zaki, Mining protein contact maps, in: ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD), 2002.
- [10] S. Zhang, L. Liao, J.-F. Tomb, J. T.-L. Wang, Clustering and classifying enzymes in metabolic pathways: Some preliminary results, in: ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD), 2002.
- [11] C. Tang, A. Zhang, J. Pei, Mining phenotypes and informative genes from gene expression data, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003.
- [12] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, R. S. Sharm, Discovering all most specific sentences, *ACM Transactions on Database Systems (TODS)* 28 (2) (2003) 140–174.
- [13] L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science* 8 (1979) 189–201.
- [14] L. G. Valiant, The complexity of enumeration and reliability problems, *SIAM Journal on Computing* 8 (3) (1979) 410–421.
- [15] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.
- [16] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, New algorithms for fast discovery of association rules, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 1997.
- [17] D.-I. Lin, Z. M. Kedem, Pincer-Search: An efficient algorithm for discovering the maximum frequent set, *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 14 (3) (2002) 553–566.
- [18] R. J. Bayardo Jr., Efficiently mining long patterns from databases, in: ACM International Conference on Management of Data (SIGMOD), 1998.
- [19] R. C. Agarwal, C. C. Aggarwal, V. V. V. Prasad, Depth first generation of long patterns, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2000.
- [20] D. Burdick, M. Calimlim, J. Gehrke, MAFIA: A maximal frequent itemset algorithm for transactional databases, in: International Conference on Data Engineering (ICDE), 2001.

- [21] K. Gouda, M. J. Zaki, Efficiently mining maximal frequent itemsets, in: IEEE International Conference on Data Mining (ICDM), 2001.
- [22] E. Boros, V. Gurvich, L. Khachiyan, K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in: International Symposium on Theoretical Aspects of Computer Science (STACS), 2002.
- [23] R. Agrawal, R. Srikant, Mining sequential patterns, in: International Conference on Data Engineering (ICDE), 1995.
- [24] M. J. Zaki, SPADE: An efficient algorithm for mining frequent sequences, *Machine Learning* 42 (1/2) (2001) 31–60.
- [25] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, FreeSpan: Frequent pattern-projected sequential pattern mining, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2000.
- [26] S. Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas, Incremental and interactive sequence mining, in: ACM International Conference on Information and Knowledge Management (CIKM), 1999.
- [27] J. Pei, J. Han, W. Wang, Mining sequential patterns with constraints in large databases, in: ACM International Conference on Information and Knowledge Management (CIKM), 2002.
- [28] M. Seno, G. Karypis, SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint, in: IEEE International Conference on Data Mining (ICDM), 2002.
- [29] M. N. Garofalakis, R. Rastogi, K. Shim, Mining sequential patterns with regular expression constraints, *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 14 (3) (2002) 530–552.
- [30] M. J. Zaki, Efficiently mining frequent trees in a forest, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002.
- [31] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Satamoto, S. Arikawa, Efficient substructure discovery from large semi-structured data, in: SIAM International Conference on Data Mining (SDM), 2002.
- [32] Y. Xiao, J.-F. Yao, Z. Li, M. H. Dunham, Efficient data mining for maximal frequent subtrees, in: IEEE International Conference on Data Mining (ICDM), 2003.
- [33] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), 2000.
- [34] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: IEEE International Conference on Data Mining (ICDM), 2001.

- [35] X. Yan, J. Han, gSpan: Graph-based substructure pattern mining, in: IEEE International Conference on Data Mining (ICDM), 2002.
- [36] X. Yan, J. Han, CloseGraph: Mining closed frequent graph patterns, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003.
- [37] M. J. Zaki, M. Ogihara, Theoretical foundations of association rules, in: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), 1998.
- [38] J. Wang, J. Han, J. Pei, CLOSET+: Searching for the best strategies for mining frequent closed itemsets, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003.
- [39] E. Ukkonen, On-line construction of suffix trees, *Algorithmica* 14 (3) (1995) 249–260.
- [40] D. H. Younger, Recognition and parsing of context-free languages in time n^3 , *Information and Control* 10 (2) (1967) 189–208.
- [41] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
- [42] J. S. Provan, M. O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, *SIAM Journal on Computing* 12 (4) (1983) 777–788.
- [43] T. H. Cormen, C. E. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, Second Edition, The MIT Press, 2001.
- [44] D. W. Matula, Subtree isomorphism in $O(n^{5/2})$, *Annals of Discrete Mathematics* 2 (1978) 91–106.
- [45] P. Kilpeläinen, H. Mannila, Ordered and unordered tree inclusion, *SIAM Journal on Computing* 24 (2) (1995) 340–356.
- [46] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, R. E. Stearns, The complexity of planar counting problems, *SIAM Journal on Computing* 27 (4) (1998) 1142–1167.
- [47] S. P. Vadhan, The complexity of counting in sparse, regular, and planar graphs, *SIAM Journal on Computing* 31 (2) (2001) 398–427.
- [48] S. O. Kuznetsov, Interpretation on graphs and complexity characteristics of a search for specific patterns, *Nauchno-Tekhnicheskaya Informatsiya, Seriya 2 (Automatic Documentation and Mathematical Linguistics)* 23 (1) (1989) 23–27.
- [49] G. Ramesh, W. Maniatty, M. J. Zaki, Feasible itemset distributions in data mining: Theory and application, in: ACM International Symposium on Principles of Database Systems (PODS), 2003.
- [50] C. H. Papadimitriou, NP-Completeness: A retrospective, in: International Colloquium on Automata, Languages and Programming (ICALP), 1997.