The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns

[Extended Abstract]

Guizhen Yang
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
gzyang@cse.buffalo.edu

ABSTRACT

Mining maximal frequent itemsets is one of the most fundamental problems in data mining. In this paper we study the complexity-theoretic aspects of maximal frequent itemset mining, from the perspective of counting the number of solutions. We present the first formal proof that the problem of counting the number of distinct maximal frequent itemsets in a database of transactions, given an arbitrary support threshold, is #P-complete, thereby providing strong theoretical evidence that the problem of mining maximal frequent itemsets is NP-hard. This result is of particular interest since the associated decision problem of checking the existence of a maximal frequent itemset is in P.

We also extend our complexity analysis to other similar data mining problems dealing with complex data structures, such as sequences, trees, and graphs, which have attracted intensive research interests in recent years. Normally, in these problems a partial order among frequent patterns can be defined in such a way as to preserve the downward closure property, with maximal frequent patterns being those without any successor with respect to this partial order. 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.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Theory

Keywords

data mining, maximal frequent itemset, maximal frequent pattern, #P-complete, #P-hard

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

1. INTRODUCTION

Since the introduction of the Apriori algorithm about a decade ago [2], the field of data mining has flourished into a research area of significant technological and social importance, with applications ranging from business intelligence to security to bioinformatics. However, in spite of the multitude of data mining algorithms developed, not much effort has been made on the theoretical frontend to study the inherent complexity nature of data mining 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 lights 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 [10] — 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 has become 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 [10].

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 (compute) all (maximal) frequent patterns should be able to count them as efficiently as well. This counting aspect reveals the inherent complexity nature of data mining problems rather deeply — the expected output is merely a number instead of a presentation of all solutions; and 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.

We use the notion of #P-completeness as a theoretical analysis tool to study the complexity of counting problems. The class #P was first introduced in [22] to include all counting problems for which any single solution can be computed by a nondeterministic Turing machine in polynomial time [8]. 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 [8], 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.[1]

Our theoretical investigation begins with the problem of mining maximal frequent itemsets — one of the most fundamental problems studied in data mining [28, 10, 15, 5, 1, 7, 9]. We present the *first* formal proof (see Section 4) that the problem of counting the number of maximal frequent itemsets in a database of transaction, given an arbitrary support threshold, is #P-complete, thereby providing strong 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.

Note that number of maximal frequent itemsets can be exponentially smaller than the number of frequent itemsets [28, 10]. In contrast to mining frequent itemsets, several algorithms have been shown to be able to gain computational efficiency substantially for mining maximal frequent itemsets [28, 10, 15, 5, 1, 7, 9]. Given that the problem of counting the number of frequent itemsets has also been shown to be #P-complete [10], our new complexity result implies a rather unexpected analogy: the problem of mining maximal frequent itemsets is of as great worst-case computational complexity as the problem of mining frequent itemsets.

Having established the #P-completeness of the counting problem for maximal frequent itemsets, we also extend our complexity analysis to other similar problems that deal with complex data structures, such as sequences [3, 25], trees [26, 4, 24], and graphs [12, 13], 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 notions 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 itemsets 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.

[1]All the complexity results presented in this paper should be interpreted as *worst-case* time complexity.

2. PRELIMINARIES

In this section 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 [27] for a more detailed survey). Then we formalize several notions of itemsets with different support characteristics. Table 1 summarizes the notations that will be used in this paper and their meaning.

$\mathcal{D}$	a database of transactions
I	an itemset
tid	transaction identifier
$\|S\|$	the cardinality of a set S
$\mathcal{D}(I)$	all transactions in database $\mathcal{D}$ that contain I
$f_{\mathcal{D}}(I)$	the support of I in database $\mathcal{D}$
$\mathcal{F}_\sigma(\mathcal{D})$	all σ-frequent itemsets in database $\mathcal{D}$
$\mathcal{M}_\sigma(\mathcal{D})$	all maximal σ-frequent itemsets in database $\mathcal{D}$
$\mathcal{C}_\delta(\mathcal{D})$	all maximal δ-occurrent itemsets in database $\mathcal{D}$

Table 1: Summary of Notations and Their Meaning

2.1 Databases and Itemsets

A database comprises a set of transactions. 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.*, $\mathcal{D}(I) \stackrel{\text{def}}{=} \{t \,|\, I \subseteq t, t \in \mathcal{D}\}$.

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 represent 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. Given a a bipartite graph $\mathcal{G}$, we will use $\mathcal{D}_\mathcal{G}$ to denote its corresponding database of transactions.

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 $t_1, t_2, \cdots, t_m$ (corresponding to rows 1 to m of a matrix) and all the items as $x_1, x_2, \cdots, x_n$ (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.

Example 1 Consider a database $\mathcal{D}$ that consists of the following transactions, t_1, t_2, t_3, t_4, t_5, where $t_1 = \{x_1, x_2\}$, $t_2 = \{x_1, x_2, x_3\}$, $t_3 = \{x_1, x_2, x_3, x_4\}$, $t_4 = \{x_3, x_4\}$, $t_5 = \{x_3, x_4\}$. Here x_1, x_2, x_3, x_4 denote different 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. □

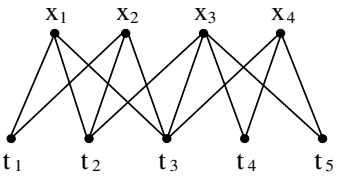

Figure 1: Bipartite Graph Representation of the Database in Example 1

	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

Figure 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.

2.3 Support and Maximality

Classical Data mining problems are usually concerned with itemsets that frequently occur in a database of transactions. The number of occurrences of an itemset in a database is commonly referred to as the *support* of this itemset, formalized as follows.[2]

[2]The support of an itemset can be also defined as the *percentage* of transactions that are a superset of it. 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 transactions.

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) For $\delta \geq 1$, we will say that an itemset I is δ-*occurrent* in a database $\mathcal{D}$, if the support of I in $\mathcal{D}$ is δ, *i.e.*, $f_\mathcal{D}(I) = \delta$.

Definition 3 (σ-Frequent Itemsets) For $1 \leq \sigma \leq |\mathcal{D}|$, an itemset I is called σ-*frequent* in a database $\mathcal{D}$, 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-occurrent itemsets, then $\mathcal{D}(I) = \mathcal{D}(J)$.

In the sequel, when we discuss properties of itemsets with respect to a database $\mathcal{D}$, for simplicity 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 such that $1 \leq \sigma \leq |\mathcal{D}|$, enumerate all σ-frequent itemsets in $\mathcal{D}$.*

We should point out the difference between δ-occurrent and σ-frequent itemsets. If an itemset is δ-occurrent, then its support must be *exactly* δ. The 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 corresponding 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 ($\mathcal{D}(\{x_1, x_2\}) = \{t_1, t_2, t_3\}$). □

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 Itemsets) Let I be a δ-occurrent itemset in a database $\mathcal{D}$. 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 $\mathcal{D}$.[3]

[3]A maximal δ-occurrent itemset is essentially a frequent closed itemset with support δ [23], if δ is not less than the support threshold. This explicit notation will be handy for our complexity analysis.

Definition 5 (Maximal σ-Frequent Itemsets) Let I be a σ-frequent itemset in a database $\mathcal{D}$. 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 such that $1 \leq \sigma \leq |\mathcal{D}|$, 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 $\lambda \geq \delta$.

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

Example 3 Continue with the previous database example 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 a 2-occurrent itemset; $\{x_1, x_2, x_3\}$ 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. □

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 shown 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. □

3. THEORETICAL FOUNDATIONS

Most data mining problems espouse two different but in fact closely related perspectives: enumeration of all solutions and counting the number of solutions. In this section we will first discuss the counting aspect of the problem of mining maximal frequent itemsets and then 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 maximal frequent itemsets?*

Since all the maximal frequent itemsets must be enumerated, clearly the computational cost must be proportional to *at least* the *number* of all maximal frequent itemsets. So it is natural to ask the following question: *Is the number of maximal frequent itemsets always polynomial in the size of the database?*[4]

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.

Example 5 Let $X = \{x_1, x_2, \cdots, x_{2n-1}, x_{2n}\}$ denote a set of $2n$ items. We will construct a database $\mathcal{D}$ with $2n$ transactions, $t_1, t_2, \cdots, t_{2n-1}, t_{2n}$, 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. Now we can claim that the number of maximal n-frequent itemsets in $\mathcal{D}$ is exactly $\binom{2n}{n}$. To see why, first we can show that for any itemset $I \subseteq X$, $\mathcal{D}(I) = \{t_i \,|\, x_i \notin I, x_i \in X\}$. It follows that if $|I| = k$ then $f_\mathcal{D}(I)$ (the support of I in $\mathcal{D}$) is exactly $2n-k$. Therefore any itemset I must be 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 exactly $\binom{2n}{n}$ 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 $\mathcal{D}$. □

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 maximal frequent itemsets using some efficient data structure in polynomial time. Indeed, such discrepancy between "printing" and "computing" is not uncommon. For

[4]Here we will use the size of its corresponding binary matrix (number of entries) to refer to the size of a database. Although in practice this is not the most efficient way of storing data, such generalization does not affect our complexity analysis.

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 [20].

Note that if an algorithm can enumerate all maximal frequent itemsets then it should be able to *count* them also. This counting aspect of data mining problems is important because in contrast to enumeration, the associated counting problem might have "lower" 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, arithmetic operations, such as addition, subtraction, multiplication, division, etc., only take a polynomial (in the sizes of the two operands) number of steps to finish. 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 such that $1 \leq \sigma \leq |\mathcal{D}|$, 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

The theory of NP-completeness is mainly concerned with decision problems asking about the *existence* of a solution. On the contrary, whereas enumeration problems require explicit output of all solutions, counting problems need to calculate the *number* of solutions only.[5] 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 [22]. Here we follow the same definitions as in [22].

Definition 6 (Counting Turing Machines) 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 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.

[5]Note that the counting problems we describe here are termed as enumeration problems in [8]. But we follow the terminology in [16].

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 [8] and [22] 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.

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

Definition 8 (#P-Completeness) 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 NP-complete problems, their associated counting problems are #P-complete[6] — the hardness of counting the number of solutions for such problems originates from the computational difficulty of searching for just one! For instance, the problem of counting the number of satisfying truth assignments for an arbitrary 3CNF formula is #P-complete [8].

However, there are very hard counting problems whose associated decision problems can actually be solved in polynomial time. The first such problem was proved by Valiant [22] — 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. Clearly, if a counting problem is #P-complete or #P-hard, then its associated problem of enumerating (mining) all solutions must be NP-hard [8, 16].

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 immediately 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 of frequent item-

[6]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.

sets does not hold for maximal frequent itemsets in general, as illustrated by the example below.

Example 6 Consider the database $\mathcal{D}$ that is 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. □

From the example above, we can see that maximal frequent itemsets behave rather "randomly" and this adds much difficulty to our search of a complexity-theoretic characterization for them. Next we need to establish lemmas to 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.*, $I \not\subset J$ and $J \not\subset I$.

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

Lemma 7 If 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.

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

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

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 equality, as illustrated by the example below.

Example 7 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 $\sum_{i=2}^{|\mathcal{D}|} |\mathcal{C}_i(\mathcal{D})| = 4$. □

We will now present our proof that the problem of counting the number of maximal frequent itemsets is #P-complete. We will reduce the problem of counting the number of maximal bipartite cliques in a bipartite graph, which has been shown to be #P-complete, to the problem of counting the number of maximal frequent itemsets in a database of transactions.

Theorem 10 ([18]) The problem of counting the number of maximal bipartite cliques in a given bipartite graph is #P-complete.[7]

Corollary 11 Let $\mathcal{D}$ be a database of transactions. It is a #P-complete problem to count the number $\sum_{\delta=1}^{|\mathcal{D}|} |\mathcal{C}_\delta(\mathcal{D})|$.[8]

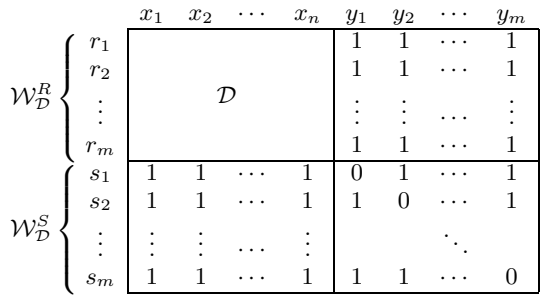

Figure 3: Database Transformation Scheme

Let $\mathcal{D}$ be a database of transactions. First we transform $\mathcal{D}$ to a new database $\mathcal{W}_\mathcal{D}$ as follows. Let $T = \{t_1, t_2, \cdots, t_m\}$ be the set of transactions of $\mathcal{D}$, where $m = |\mathcal{D}|$, and $X = \{x_1, x_2, \cdots, x_n\}$ the set of all items appearing in $\mathcal{D}$. We will introduce a set, $Y = \{y_1, y_2, \cdots, y_m\}$, of m new items into $\mathcal{W}_\mathcal{D}$. The new database $\mathcal{W}_\mathcal{D}$ is the union of two databases, $\mathcal{W}_\mathcal{D}^R = \{r_1, r_2, \cdots, r_m\}$, and $\mathcal{W}_\mathcal{D}^S = \{s_1, s_2, \cdots, s_m\}$. 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 in T 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 3. Note that we can establish a one-to-one correspondence between transactions in R and T. Clearly, if the size of $\mathcal{D}$ is $O(d)$, then the size of $\mathcal{W}_\mathcal{D}$ is $O(d^2)$.

Example 8 The database shown in Figure 4 is transformed from the database shown in Figure 2. □

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 $\mathcal{W}_\mathcal{D}^S(I \cup Y_k) = \mathcal{W}_\mathcal{D}^S(J \cup Y_z)$.

[7]This result is not explicitly stated in [18], but follows readily from the results and reductions in [18]. Note that although a similar claim is also made in [14], the proof presented in [14] is not correct.

[8]This is in fact the number of all closed itemsets in $\mathcal{D}$ [23].

	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

Figure 4: The New Database Transformed from the One in Figure 2

Proposition 13 Let $\mathcal{D}$ be a database of transactions, $|\mathcal{D}| = m$, $\mathcal{W}_\mathcal{D} = \mathcal{W}_\mathcal{D}^R \cup \mathcal{W}_\mathcal{D}^S$ the new database transformed from $\mathcal{D}$, X the set of items appearing in $\mathcal{D}$, 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 $|Y_k| = k$.

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

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

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 $1 \leq \sigma \leq m$:

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

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

Proof. Checking whether an itemset is maximal frequent or not can be done in polynomial time. Therefore, the problem of counting the number of maximal frequent itemsets is in #P. We know that the problem of counting the number $\sum_{\sigma=1}^{m} |\mathcal{C}_\sigma(\mathcal{D})|$ is #P-complete, by Corollary 11. We will show how this counting problem can be Turing reduced to the problem of counting the number of maximal frequent itemsets, thereby proving that the latter problem is #P-hard.

First, we transform the database $\mathcal{D}$ into its corresponding new database $\mathcal{W}_\mathcal{D}$. We will assume binary matrix representation for databases. Therefore, if the size of $\mathcal{D}$ is $O(d)$, then the size of $\mathcal{W}_\mathcal{D}$ is $O(d^2)$, and the running time of the transformation algorithm is also $O(d^2)$.

Let $C_\sigma = |\mathcal{C}_\sigma(\mathcal{D})|$, $M_\sigma = |\mathcal{M}_{\sigma+m}(\mathcal{W}_\mathcal{D})|$, for $1 \leq \sigma \leq m$. Then by Proposition 16, we can construct the following 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, the linear equations above have a unique solution for $(C_1, C_2, \cdots, C_m)$, given the values of $(M_1, M_2, \cdots, M_m)$.

For all $0 \leq k \leq m-1$, $\binom{m}{k} \leq m^m$ and so $\binom{m}{k}$ can be stored using $O(m \log_2 m)$ bits in binary notation and computed in time polynomial in m. Let the size of $\mathcal{D}$ be $O(d)$. Then $m = O(d)$ and so 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 $(M_1, M_2, \cdots, M_m)$, the linear equations above can be solved and hence the values of $(C_1, C_2, \cdots, C_m)$ can be calculated in time polynomial in d, the size of $\mathcal{D}$. Note that the size of $\mathcal{W}_\mathcal{D}$ is $O(d^2)$. So if there is a polynomial-time algorithm for counting the number of maximal frequent itemsets, then the values of $(M_1, M_2, \cdots, M_m)$ can be computed in time polynomial in d. Hence the values of $(C_1, C_2, \cdots, C_m)$ 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. □

5. MAXIMAL FREQUENT PATTERNS

A large number of data mining problems dealing with frequent patterns can be viewed as instances of the *theory extraction* problem [10]. 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: $f_\mathcal{D}(p) \stackrel{\text{def}}{=} |\{t \mid p \preceq t, t \in \mathcal{D}\}|$.[9] A pattern whose support exceeds a user-specified 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 this line of generalization above. For example, in the problem of mining maximal frequent itemsets, the patterns are sets of items and the partial order is defined on subset inclusion. In this section, we will extend our complexity analysis to several problems of mining maximal frequent

[9]This kind of support is called *unweighted* support, in which every database transaction contributes at most one count to the supported of a pattern. However, our complexity results can be easily extended to data mining problems that use *weighted support* [24], which takes into account multiple occurrences of a pattern in a database transaction.

patterns studied recently in the literature, 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. Support and maximality of patterns will be defined in the same way as in Section 2.3. Our goal is to show the complexity of the associated counting problems for these maximal frequent patterns.

5.1 Subsequences

In problems of mining frequent sequences [3, 25], each database transaction is considered as a sequence (string) instead of a set, in which the order of symbols appearing in this 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 below.

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

5.2 Subtrees and Subgraphs

In recent years mining frequent patterns from trees [26, 4, 24] and graphs [12, 13] has attracted a lot of research interests. Normally, the patterns of interest are subtrees or subgraphs. Note that trees are just a special form of connected, acyclic graphs. 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 any $(a, b) \in E_1$, $(\rho(a), \rho(b)) \in E_2$. Moreover, G_1 is called an *induced* subgraph of G_2 if ρ satisfies the following additional condition: for all $(\rho(a), \rho(b)) \in E_2$, $(a, b) \in E_1$. The complexity results to be presented here apply to either subgraphs or induced subgraphs. But for simplicity we will only mention subgraphs.

The definition of subgraph isomorphism can be readily used to define subtree isomorphism on tree data structures, even though definitions of trees usually need to take into account several factors: rooted or unrooted (called free trees); ordered or unordered (the order of sibling nodes is not important); labeled (edge-labeled or node-labeled or both) or unlabeled. Next we present the complexity results on labeled trees and graphs, followed by unlabeled trees and graphs.

Theorem 19 Let $\mathcal{D}$ be a database of (rooted) unordered labeled trees, $|D| = m$. The problem of counting the number of maximal σ-frequent subtrees in $\mathcal{D}$, where $1 \leq \sigma \leq m$, is #P-complete.

Theorem 20 Let $\mathcal{D}$ be a database of (rooted) ordered labeled trees, $|D| = m$. The problem of counting the number of maximal σ-frequent subtrees in $\mathcal{D}$, where $1 \leq \sigma \leq m$, is #P-complete.

We should point out that our complexity results can be readily extended to prove the #P-hardness of counting maximal frequent embedded subtrees [26] in a database of labeled trees. Moreover, we can also immediately derive the following corollary for labeled graphs. Note that Corollary 21 below applies to labeled graphs that are either directed or undirected, ordered or unordered.

Corollary 21 Let $\mathcal{D}$ be a database of labeled graphs, $|D| = m$. It is #P-hard to count the number of maximal σ-frequent subgraphs in $\mathcal{D}$, where $1 \leq \sigma \leq m$.

We will now extend our complexity results to unlabeled trees and graphs. First we will state the result for unlabeled trees. Extending this result to unlabeled graphs is rather straightforward. To prove the #P-completeness of the problem of counting the number of maximal frequent unlabeled subtrees, we will reduce to it the problem of counting the number of maximal frequent itemsets. Here the proof is rather tricky and omitted for want of space.

Theorem 22 Let $\mathcal{D}$ be a database of (rooted) unlabeled trees, $|D| = m$. The problem of counting the number of maximal σ-frequent subtrees in $\mathcal{D}$, where $1 \leq \sigma \leq m$, is #P-complete.

It is worth pointing out that the claims in Theorems 19, 20, and 22 still remain valid even if *binary* trees are supplied as input. From Theorem 22 we can immediately derive the following claim about unlabeled graphs.

Corollary 23 Let $\mathcal{D}$ be a database of unlabeled graphs, $|D| = m$. The problem of counting the number of maximal σ-frequent subgraphs in $\mathcal{D}$, where $1 \leq \sigma \leq m$, is #P-hard.

6. RELATED WORK

Valiant introduced the class #P of counting problems and proved that counting the number of distinct perfect matchings in a bipartite graph is #P-complete [22] — the first counting problem known to be #P-complete whose associated decision problem can be solved in polynomial time. In [18], 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 [18], although it was not explicitly stated there. A similar claim was also made in [14], but its proof was not correct. However, in [14] it was shown 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 [11]. Some of these results were later extended by Vadhan to even more restricted bipartite graphs of bounded degree [21].

Many algorithms have been proposed in the literature for mining maximal frequent itemsets, such as MaxClique [28], Dualize and Advance [10], Pincer-Search [15], Max-Miner [5], DepthProject [1], MAFIA [7], and GenMax [9]. These algorithms exploited different heuristics for optimization and were shown to have different good scaleup characteristics on certain benchmark datasets. However, theoretical analysis was not the main focus of these works and none of them

proved that the problem of counting the number of maximal frequent itemsets is #P-complete.

There is a large body of work in the literature on mining frequent and maximal frequent patterns from complex data structures, such as sequences [3, 25], trees [26, 4, 24], and graphs [12, 13]. Our complexity results (summarized in Table 2) can be easily extended to problems studied in these works. Specifically, in [26] the patterns of interest are embedded subtrees. Our proof of Theorem 19 implies that it is #P-hard to count the number of maximal frequent embedded subtrees in a database of labeled trees. The #P-completeness of counting the number of frequent closed itemsets [23] also readily follows our complexity results here.

The problem of counting the number of frequent itemsets was first shown to be #P-complete in [10]. In [10] it was 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 [6] 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 a new maximal frequent itemset. These results do not lend themselves to the #P-completeness of counting the number of 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!

Finally, the work of [27] aimed at providing a lattice-theoretic framework for mining frequent itemsets and association rules. Interesting work was also recently reported in [19] on characterization of length distributions of frequent and maximal frequent itemset collections, with a focus on computing tight bounds for feasible distribution. We should point out that 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 [19] 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 of the 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). Explicit output of all solutions is expected.
2. Counting Problem (Column 3 of Table 2). The goal is to compute the number of all solutions.
3. Search Problem (Column 4 of Table 2). Output of only one solution is desired, if there is any.
4. Decision Problem (Column 5 of Table 2). The primary concern is about the existence of one 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.

These four different aspects of data mining problems in fact exhibit different levels of computational complexity (see 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 labeled 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 [8]. 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 $\mathrm{FP}^{\mathrm{NP}}$ [16]. 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 [8]. 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 [20] — this is not really surprising since substrings do not manifest a combinatorial nature.

The complexity results presented in this paper should be interpreted as *worst-case* time complexity only — the implication being 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 respective to various test datasets, either synthetic or from real 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 [19] 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 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 [17]. Recently, in [10], 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 will be needed.

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 results apply to (binary) trees that are either rooted or unrooted, ordered or unordered, labeled or unlabeled (with or without duplicate labels).

[b] The same complexity results apply to graphs that are either directed or undirected, labeled or unlabeled (with or without duplicate labels).

Table 2: Summary of Complexity Results

8. ACKNOWLEDGMENT

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

9. REFERENCES

[1] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *KDD*, 2000.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

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

[4] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Satamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *SDM*, 2002.

[5] R. J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD*, 1998.

[6] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In *STACS*, 2002.

[7] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *ICDE*, 2001.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[9] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *ICDM*, 2001.

[10] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM Transactions on Database Systems (TODS)*, 28(2):140–174, 2003.

[11] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, 27(4):1142–1167, 1998.

[12] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, 2000.

[13] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, 2001.

[14] 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):23–27, 1989.

[15] D.-I. Lin and Z. M. Kedem. Pincer-Search: An efficient algorithm for discovering the maximum frequent set. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(3):553–566, 2002.

[16] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[17] C. H. Papadimitriou. NP-completeness: A retrospective. In *ICALP*, 1997.

[18] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[19] G. Ramesh, W. Maniatty, and M. J. Zaki. Feasible itemset distributions in data mining: Theory and application. In *PODS*, 2003.

[20] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[21] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[22] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[23] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *KDD*, 2003.

[24] Y. Xiao, J.-F. Yao, Z. Li, and M. H. Dunham. Efficient data mining for maximal frequent subtrees. In *ICDM*, 2003.

[25] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[26] M. J. Zaki. Efficiently mining frequent trees in a forest. In *KDD*, 2002.

[27] M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *DMKD*, 1998.

[28] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, 1997.