The Knowledge Engineering Review, Vol. 28:1, 75–105.
doi:10.1017/S0269888912000331
First published online 20 November 2012

A survey of frequent subgraph mining algorithms

CHUNTAO JIANG, FRANS COENEN and MICHELE ZITO

Department of Computer Science, The University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, UK; e-mail: cjiang@csc.liv.ac.uk, frans@csc.liv.ac.uk, michele@csc.liv.ac.uk

Abstract

Graph mining is an important research area within the domain of data mining. The field of study concentrates on the identification of frequent subgraphs within graph data sets. The research goals are directed at: (i) effective mechanisms for generating candidate subgraphs (without generating duplicates) and (ii) how best to process the generated candidate subgraphs so as to identify the desired frequent subgraphs in a way that is computationally efficient and procedurally effective. This paper presents a survey of current research in the field of frequent subgraph mining and proposes solutions to address the main research issues.

1 Introduction

The primary goal of data mining is to extract statistically significant and useful knowledge from data (Chen *et al*., 1996; Han & Kamber, 2006). The data of interest can take many forms: vectors, tables, texts, images, and so on. Data can also be represented by various means. Structured data and semi-structured data are naturally suited to graph representations. To give one example, if we consider protein–protein interaction networks (a common application area for graph mining), these can be represented in a graph format such that the vertexes indicate genes, and the directed or undirected edges indicate physical interactions or functional associations (Alm & Arkin, 2003). Because of the ease with which structured and semi-structured data can be represented in graph formats, there has been much interest in the mining of graph data (often referred to as graph-based data mining or graph mining). A number of popular research sub-domains of graph mining are listed in Table 1.

Frequent subgraph mining (FSM) is the essence of graph mining. The objective of FSM is to extract all the frequent subgraphs, in a given data set, whose occurrence counts are above a specified threshold. Figure 1 presents an overview of the domain of FSM in terms of the number of significant FSM algorithms that have been proposed over the period 1994 to the present. From the figure, we can see periods of activity in the early 90s (coinciding with the introduction of the concept of data mining), followed by another period of activity from 2002 to 2007. No 'new' algorithms have been introduced over the past few years, indicating that the field is reaching maturity, although there has been much work focused on variations of existing algorithms.

Other than the research activity associated with FSM, the importance of FSM is also reflected in its many areas of its application. Figure 1b presents an overview of the application domain of FSM in terms of the number of FSM algorithms reported in the literature and the specific application domain at which they have been directed. From the figure, it can be seen that three application domains (chemistry, web, and biology) dominated the usage of FSM algorithms.

The straightforward idea behind FSM is to 'grow' candidate subgraphs, in either a breadth-first or depth-first manner (candidate generation), and then determine whether the identified candidate subgraphs occur frequently enough in the graph data set for them to be considered interesting

Table 1 Popular graph mining research sub-domains

Frequent subgraph mining (Cook & Holder, 1994, 2000; Inokuchi *et al*., 2000; Yan & Han, 2002)
Correlated graph pattern mining (Ke *et al*., 2007, 2009; Ozaki & Ohkawa, 2008)
Optimal graph pattern mining (Fan *et al*., 2008; Yan *et al*., 2008)
Approximate graph pattern mining (Kelley *et al*., 2003; Sharan *et al*., 2005; Chen *et al*., 2007a)
Graph pattern summarization (Xin *et al*., 2006; Chen *et al*., 2008)
Graph classification (Kudo *et al*., 2004; Huan *et al*., 2004; Deshpande *et al*., 2005)
Graph clustering (Flake *et al*., 2004; Newman, 2004; Huang & Lai, 2006)
Graph indexing (Shasha *et al*., 2002; Yan *et al*., 2004)
Graph searching (Yan *et al*., 2005b, 2006; Chen *et al*., 2007b)
Graph kernels (Gärtner *et al*., 2003; Kashima *et al*., 2003; Borgwardt & Kriegel 2005)
Link mining (Chakrabarti *et al*., 1999; Kosala & Blockeel, 2000; Getoor & Diehl, 2005; Liu, 2008)
Web structure mining (Brin & Page, 1998; Kleinberg, 1998)
Work-flow mining (Greco *et al*., 2005)
Biological network mining (Hu *et al*., 2005)

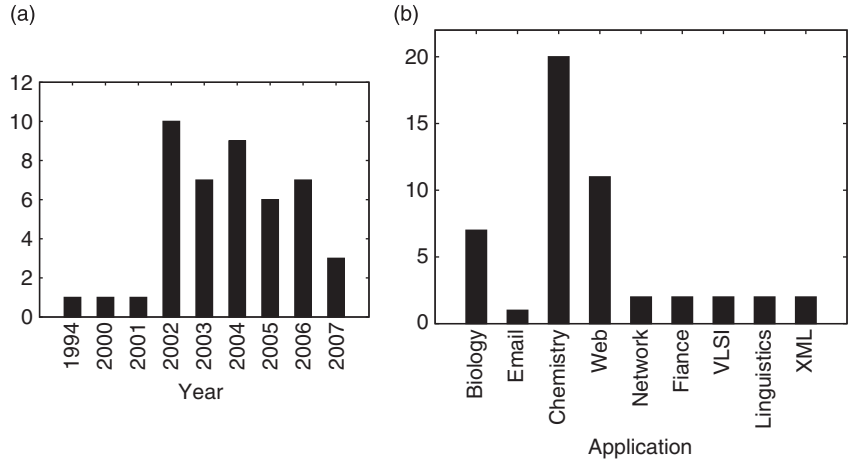

Figure 1 The distribution of the most significant FSM algorithms with respect to the year of introduction and application domain

(support counting). Thus, the two main research issues in FSM are how to efficiently and effectively (i) generate the candidate frequent subgraphs and (ii) determine the frequency of occurrence of the generated subgraphs. Effective candidate subgraph generation requires that the generation of duplicate or superfluous candidates is avoided. Occurrence counting requires repeated comparison of candidate subgraphs with subgraphs in the input data, a process known as isomorphism checking. FSM, in many respects, can be viewed as an extension of Frequent Itemset Mining (FIM) popularized in the context of Association Rule Mining (ARM; see e.g. Agrawal & Srikant, 1994). Consequently, many of the proposed solutions to addressing the main research issues effecting FSM are based on similar techniques found in the domain of FIM. For example, the downward closure property (DCP) associated with itemsets has been widely adopted with respect to candidate subgraph generation.

In this paper, the authors present a survey of the current 'state of the art' of FSM. With reference to the literature, we can identify many different types of mining strategies, with respect to many different types of graph, to produce many different kinds of patterns. So as to impose some form of order to the domain of FSM, we have focused on the nature of FSM algorithms, categorizing such algorithms according to: (i) candidate generation strategy, (ii) the mechanism for traversing the search space, and (iii) the occurrence counting process. To further facilitate understanding of the field of FSM, we distinguish between frequent subtree mining and the much

general domain of FSM. The rest of this paper is organized as follows. We begin in Section 2 by introducing some formal definitions and terminology, followed in Section 3 with a generic overview of the FSM process. In Sections 4 and 5, we then consider current frequent subtree and subgraph mining algorithms, respectively. A brief summary is provided at the end of both the sections. Finally, Section 6 presents some conclusions and future directions.

2 Formalism

There are two separate problem formulations for FSM: (i) *graph transaction-based FSM* and (ii) *single graph-based FSM*. In graph transaction-based FSM, the input data comprise a collection of medium-size graphs called *transactions*. Note that the term 'transaction' is borrowed from the field of ARM (Agrawal & Srikant, 1994). In single graph-based FSM the input data, as the name implies, comprise one very large graph.

A subgraph g is considered to be frequent if its *occurrence count* is greater than some predefined threshold value. The occurrence count for a subgraph is usually referred to as its *support*, and consequently the threshold is referred to as the *support threshold*. The support of g may be computed using either *transaction-based counting* or *occurrence-based counting*. Transaction-based counting is only applicable to graph transaction-based FSM, while occurrence-based counting may be applied to either transaction-based FSM or single graph-based FSM. However, occurrence-based counting is typically used with single graph-based FSM.

In transaction-based counting the support is defined by the number of graph transactions that g occurs in, one count per transaction regardless of whether g occurs once or more than once in a particular graph transaction. Thus, given a database $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ consisting of a collection of graph transactions, and a support threshold σ $(0 < \sigma \leqslant 1)$; then the set of graph transactions where a subgraph g occurs is defined by $\delta_{\mathcal{G}}(g) = \{G_i | g \subseteq G_i\}$. Thus, the *support* of g is defined as

$$sup_{\mathcal{G}}(g) = |\delta_{\mathcal{G}}(g)|/T \tag{1}$$

where $|\delta_{\mathcal{G}}(g)|$ denotes the cardinality of $\delta_{\mathcal{G}}(g)$, and T the number of graphs (transactions) in G. Therefore, g is *frequent* if and only if $sup_{\mathcal{G}}(g) \geqslant \sigma$. In occurrence-based counting, we simply count up the number of occurrences of g in the input set.

Transaction-based counting offers the advantage that the well-known DCP[1] can be employed to significantly reduce the computation overhead associated with candidate generation in FSM. In the case of occurrence-based counting, either an alternative frequency measure, which maintains the DC property, must be established or some heuristics adopted to keep the computation as inexpensive as possible. There are a variety of support measures (Vanetik, 2002; Kuramochi & Karypis, 2004c, 2005; Vanetik *et al.*, 2006) that may be adopted for single graph-based FSM, these will be discussed further in Subsection 5.1.2.

2.1 Preliminary definitions

Generally speaking, a *graph* is defined to be a set of vertexes (nodes), which are interconnected by a set of edges (links) (Gibbons, 1985). The graphs used in FSM are assumed to be *labelled simple graphs*[2]. In the following paragraphs, a number of widely used definitions, used later in this paper, are introduced.

Labelled graph: A labelled graph can be represented as $G(V, E, L_V, L_E, \varphi)$, where V is a set of vertexes, $E \subseteq V \times V$ is a set of edges; L_V and L_E are sets of vertex and edge labels,

[1] If a graph is frequent, then all of its subgraphs will also be frequent.

[2] A *simple graph* is an unweighted and undirected graph with no loops and no multiple links between any two distinct nodes (Gibbons, 1985; West, 2000).

respectively; and φ is a label function that defines the mappings $V \rightarrow L_V$ and $E \rightarrow L_E$. G is *(un)directed* if $\forall e \in E$, e is an (un)ordered pair of vertexes. A *path* in G is a sequence of vertexes that can be ordered such that two vertexes form an edge if and only if they are consecutive in the list (West, 2000). G is *connected*, if it contains a path for every pair of vertexes in it and *disconnected* otherwise. G is *complete* if each pair of vertexes is joined by an edge, and G is *acyclic* if it contains no cycle.

Subgraph: Given two graphs $G_1(V_1, E_1, L_{V_1}, L_{E_1}, \varphi_1)$ and $G_2(V_2, E_2, L_{V_2}, L_{E_2}, \varphi_2)$, G_1 is a subgraph of G_2, if G_1 satisfies: (i) $V_1 \subseteq V_2$ and $\forall v \in V_1, \varphi_1(v) = \varphi_2(v)$ and (ii) $E_1 \subseteq E_2$ and $\forall (u, v) \in E_1, \varphi_1(u, v) = \varphi_2(u, v)$. G_1 is an *induced subgraph* of G_2, if G_1 further satisfies: $\forall u, v \in V_1, (u, v) \in E_1 \Leftrightarrow (u, v) \in E_2$, in addition to the above conditions. G_2 is also a supergraph of G_1 (Inokuchi *et al.*, 2002; Huan *et al.*, 2003).

Graph isomorphism: A graph $G_1(V_1, E_1, L_{V_1}, L_{E_1}, \varphi_1)$ is isomorphic to another graph $G_2(V_2, E_2, L_{V_2}, L_{E_2}, \varphi_2)$, if and only if a bijection $f: V_1 \rightarrow V_2$ exists such that: (i) $\forall u \in V_1$, $\varphi_1(u) = \varphi_2(f(u))$, (ii) $\forall (u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$, and (iii) $\forall (u, v) \in E_1$, $\varphi_1(u, v) = \varphi_2(f(u), f(v))$. The bijection f is an isomorphism between G_1 and G_2. A graph G_1 is *subgraph isomorphic* to a graph G_2, if and only if there exists a subgraph $g \subseteq G_2$ such that G_1 is isomorphic to g (Huan *et al.*, 2003). In this case g is called an *embedding* of G_1 in G_2.

Lattice: Given a database $\mathcal{G}$, a lattice is a structural form used to model the search space for finding frequent subgraphs, where each vertex represents a connected subgraph of the graph in $\mathcal{G}$ (Thomas *et al.*, 2006). The lowest vertex depicts the empty subgraph and the vertexes at the highest level depict the graphs in $\mathcal{G}$. A vertex p is a parent of the vertex q in the lattice, if q is a subgraph of p, and q is different from p by exactly one edge. The vertex q is a child of p. All the subgraphs of each graph $G_i \in \mathcal{G}$ that occur in the database are present in the lattice, and every subgraph occurs only once in it.

Example: given a graph data set $\mathcal{G} = \{G_1, G_2, G_3, G_4\}$, the corresponding *Lattice*($\mathcal{G}$) is given in Figure 2. In the figure, the lowest vertex ϕ represents the empty subgraph, and the vertexes at the highest level correspond to G_1, G_2, G_3, and G_4. The parents of the subgraph *B–D* are subgraphs *A–B–D* (joining the edge *A–B*) and *B–D–G* (joining the edge *D–G*). Similarly, subgraphs *B–C* and *C–F* are the children of the subgraph *B–C–F*.

Free tree: An undirected graph that is connected and acyclic (Chi *et al.*, 2004, 2004a).

Labelled unordered tree: A labelled unordered tree (an unordered tree, for short) is a directed acyclic graph (DAG) denoted as $T(V, \phi, E, v_r)$, where V is a set of vertexes of T; ϕ is a labelling function, such that $\forall v_i \in V$, $\phi(v_i) \rightarrow v_i$; $E \subseteq V \times V$ is a set of edges of T; and v_r is a distinguished vertex called *root* of T. For $\forall v_i \in V$, there is a unique path $(v_r, v_1, v_2, \ldots, v_i)$ from the root v_r to v_i (Asai *et al.*, 2002, 2003). If a vertex v_i is on the path from the root to the vertex v_j, then v_i is an *ancestor* of v_j, and v_j is a *descendant* of v_i. For each edge $(v_i, v_j) \in E$, v_i is the *parent* of v_j, and v_j is a *child* of v_i. Vertexes that share the same parent are *siblings*. The *size* of T is defined to be the number of vertexes in T. A vertex without any child is a *leaf* vertex; otherwise it is an *intermediate* vertex. The *right-most path* of T is the path from the root vertex to the *right-most leaf*. The *depth* (*level*) of a vertex is the *length of the path*[3] from the root to that vertex. The *degree* of a vertex v, denoted by *degree* (v), is the number of edges incident to it (West, 2000; Chi *et al.*, 2004, 2004a; Tan *et al.*, 2005).

Labelled ordered tree: A labelled ordered tree[4] (an ordered tree, for short) is a labelled unordered tree but with a left-to-right ordering imposed among the children of each vertex (Asai *et al.*, 2002, 2003; Chi *et al.*, 2004).

Bottom-up subtree: Given a rooted tree $T(V, \phi, E, v_r)$ (ordered or unordered), $\vec{T}(\vec{V}, \vec{\phi}, \vec{E}, \vec{v}_r)$ is a bottom-up subtree of T, if and only if: (i) $\vec{V} \subseteq V$, (ii) $\vec{E} \subseteq E$, (iii) the labelling of $\vec{V}$ and $\vec{E}$ in T is preserved in $\vec{T}$, (iv) $\forall v \in V$, if $v \in \vec{V}$ then all descendants of v must also be in $\vec{V}$, and

[3] The *length of a path* is equivalent to the number of edges in the path.

[4] A labelled ordered tree, in graph theory, is also called a *rooted plane tree* (West, 2000).

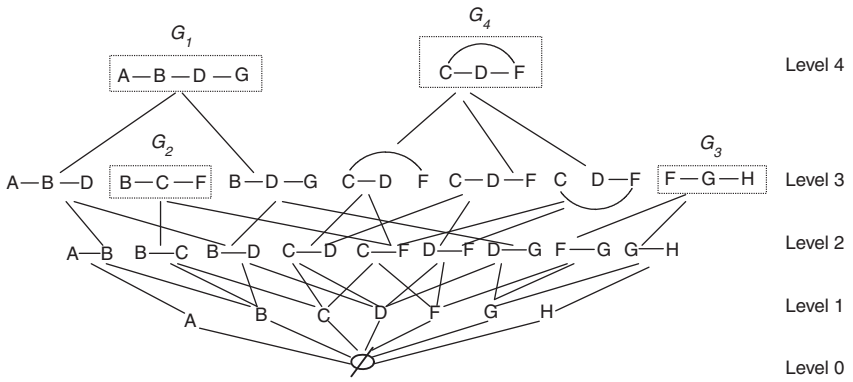

Figure 2 *Lattice*($\mathcal{G}$) (Figure based on a similar figure presented in Thomas *et al*., 2006)

(v) if T is ordered, then the left-to-right ordering among the siblings in T should be preserved in $\vec{T}$ (Valiente, 2002; Chi *et al*., 2004).

Induced subtree: Given a labelled tree $T(V, \phi, E, v_r)$ (free tree or unordered tree or ordered tree), $\vec{T}(\vec{V}, \vec{\phi}, \vec{E}, \vec{v}_r)$ is an induced subtree of T, if and only if: (i) $\vec{V} \subseteq V$, (ii) $\vec{E} \subseteq E$, (iii) the labelling of $\vec{V}$ and $\vec{E}$ in T is preserved in $\vec{T}$, and (iv) if defined for ordered trees, the left-to-right ordering among the siblings in $\vec{T}$ should be a sub-ordering of the corresponding vertexes in T (Chi *et al*., 2004; Tan *et al*., 2006).

Embedded subtree: Given a labelled tree $T(V, \phi, E, v_r)$, $\vec{T}(\vec{V}, \vec{\phi}, \vec{E}, \vec{v}_r)$ is an embedded subtree of T, if and only if: (i) $\vec{V} \subseteq V$, (ii)$\forall v \in \vec{V}, \vec{\phi}(v) = \phi(v)$, (iii) $\forall (u, v) \in \vec{E}$, such that u is the parent of v, u is an ancestor of v in T, and (iv) in the case of ordered trees, $\forall (u, v) \in \vec{V}, preorder(u)$ $preorder(v)$ in $\vec{T}$ if and only if *preorder* $(u) <$ *preorder* (v) in T, where the pre-order of a vertex is its index in the tree, according to the pre-order traversal[5].

To summarize the above, Figure 3 gives some examples of bottom-up subtrees, induced subtrees, and embedded subtrees. In the figure: tree (a) on the left represents a data tree, trees (d) and (e) are two bottom-up subtrees of (a), trees (f) and (g) are two induced subtrees of (a), and trees (b) and (c) are two embedded subtrees of (a). The relationship among these three types of subtrees can be denoted as: bottom-up subtree $\subseteq$ induce subtree $\subseteq$ embedded subtree.

2.2 Graph isomorphism detection

The kernel of FSM is *(sub)graph isomorphism detection*. Graph isomorphism is neither known to be solvable in polynomial time nor *NP*-complete, while subgraph isomorphism, where we wish to establish whether a subgraph is wholly contained within a supergraph, is known to be NP-complete (Garey & Johnson, 1979). When restricting the graphs to trees, (sub)graph isomorphism detection becomes (sub)tree isomorphism detection. Tree isomorphism detection can be solved in a linear time (see algorithm proposed in Hopcroft & Tarjan, 1972). Faster subtree isomorphism detection algorithms, with worst case time complexity of $\mathcal{O}(k^{1.5}n)$, were proposed by Matula (1978) and Chung (1987), and further improved upon by Shamir and Tsur (1999) in $\mathcal{O}\left(\frac{k^{1.5}}{\log k}n\right)$ time (k and n are the sizes of the subtree and the tree to be searched in terms of the number of vertexes).

Subgraph isomorphism detection is fundamental to FSM. A significant number of 'efficient' techniques have been proposed, all directed at reducing, as far as possible, the computational overhead associated with subgraph isomorphism detection. Subgraph isomorphism detection techniques can be roughly categorized as being either: exact matching (Schmidt & Druffel, 1976;

[5] *A pre-order traversal* is where a sequence of operations are performed recursively as follows: visit the root first, and then do a pre-order traversal of each of the subtrees of the root one-by-one in the order given (Preiss, 1998).

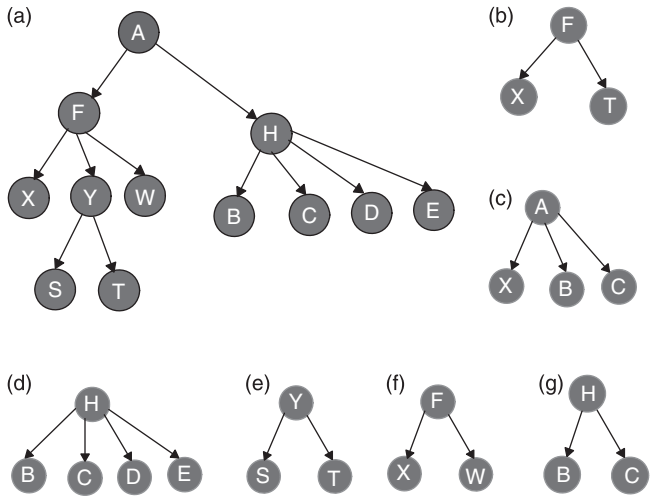

Figure 3 Different types of trees

Table 2 Categorization of exact matching (sub)graph isomorphism testing algorithms

Algorithms	Main techniques	Matching types
Ullmann	Backtracking + look ahead function	Graph and subgraph isomorphism
SD	Distance matrix + backtracking	Graph isomorphism
Nauty	Group theory + canonical labelling	Graph isomorphism
VF	DFS strategy + feasibility rules	Graph and subgraph isomorphism
VF2	VF's rationale + advanced data structures	Graph and subgraph isomorphism

DFS = Depth First Search; SD = Schmidt and Druffel.

Ullmann, 1976; McKay, 1981; Cordella *et al.*, 1998, 2001) or error tolerant matching (Shapiro & Haralick, 1981; Bunke & Allerman, 1983; Christmas *et al.*, 1995; Messmer & Bunke, 1998). Most FSM algorithms adopt exact matching. A categorization of the main exact matching subgraph isomorphism detection algorithms is presented in Table 2. In Table 2, column two indicates the main methods employed to carry out the isomorphism detection, and column three indicates whether the isomorphism detection algorithm applies to graph isomorphism or subgraph isomorphism.

With reference to Table 2, Ullmann's algorithm employs a backtracking procedure with a look-ahead function to reduce the size of the search space (Ullmann, 1976). The SD (Schmidt and Druffel) algorithm, in turn, utilizes a distance matrix representation of a graph with a backtracking procedure to reduce the search (Schmidt & Druffel, 1976). The Nauty algorithm (McKay, 1981) uses group theory to transform graphs to be matched into a canonical form so as to provide for more efficient and effective graph isomorphism checking. However, it has been noted (Conte *et al.*, 2004) that the construction of the canonical forms can lead to exponential complexity in the worst case. Although Nauty was regarded as the fastest graph isomorphism algorithm by Conte *et al.* (2004), Miyazaki (1997) demonstrated that there exists some categories of graphs, which required exponential time to generate the canonical labelling. The VF (Cordella *et al.*, 1998) and VF2 (Cordella *et al.*, 2001) algorithms use a Depth First Search (DFS) strategy, assisted by a set of feasibility rules to prune the search tree. VF2 is an improved version of VF that explores the search space more effectively so that the matching time and the memory consumption are significantly reduced. In Foggia *et al.* (2001), a detailed experimental analysis of these five algorithms is provided to indicate that none of the existing algorithms is completely superior to the others. In general, VF2 was found to give the best performance with respect to the size and the type of graphs to be matched.

3 Overview of frequent subgraph mining

This section provides a generic overview of the process of FSM. It is widely accepted that FSM techniques can be divided into two categories: (i) Apriori-based approaches and (ii) pattern growth-based approaches. These two categories are similar in spirit to counterparts found in ARM, namely the Apriori algorithm (Agrawal & Srikant, 1994) and Frequent Pattern (FP)-growth algorithm (Han *et al.*, 2000), respectively. The Apriori-based approach proceeds in a generate-and-test manner using a Breadth First Search (BFS) strategy to explore the subgraph lattice of the given database. Therefore, before considering $(k+1)$ subgraphs, this approach has to first consider all k subgraphs. The pattern growth-based adopts a DFS strategy is depicted where, for each discovered subgraph g, the subgraph is extended recursively until all frequent supergraphs of g are discovered (Han & Kamber, 2006). The distinction between the two approaches is illustrated in Figure 4.

The basic Apriori-based algorithm is presented in 3.1. In line 5, all frequent $(k-1)$ subgraphs are used to generate k subgraph candidates. If any of the $k-1$ candidate subgraphs are not frequent, then the DCP (see Section 2) can be used to safely prune the candidates. Most existing FSM approaches adopt an iterative pattern mining strategy, where each iteration can be divided into two phases: (i) candidate generation (line 5 in Algorithm 3.1) and (ii) support computation (lines 6–12 in Algorithm 3.1). Generally, research on FSM focuses on these two phases using a variety of techniques. Since it is harder to address subgraph isomorphism detection, more research

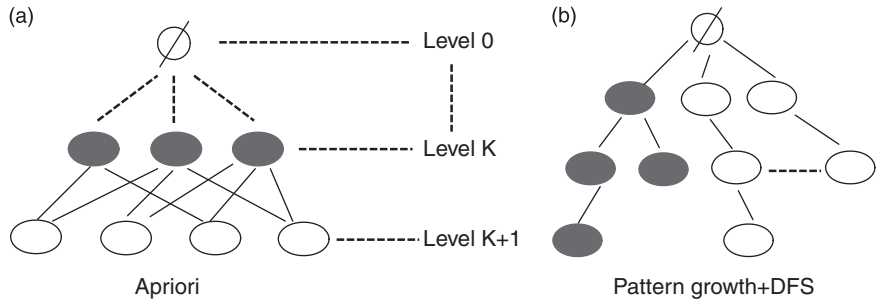

Figure 4 Two types of search space. Note that the subgraph lattice is shown 'upside-down'. Vertexes corresponding to graphs with fewer edges are displayed at the top of the picture in each case

Algorithm 3.1: Apriori-based approach

Input: $\mathcal{G}$ = a graph data set, σ = minimum support
Output: $F_1, F_1, \ldots, F_k$, a set of frequent subgraphs of cardinality 1 to k

```
1  F_1 ← detect all frequent 1 subgraphs in G
2  k ← 2
3  while F_{k-1} ≠ ∅ do
4      F_k ← ∅
5      C_k ← candidate−gen (F_{k-1})
6      foreach candidate g ∈ C_k do
7          g.count ← 0
8          foreach G_i ∈ G do
9              if subgraph-isomorphism (g, G_i) then
10                 g.count ← g.count + 1
11             end
12         end
13         if g.count ⩾ σ|G| ∧ g ∉ F_k then
14             F_k = F_k ∪ g
15         end
16     end
17     k ← k + 1
18 end
```

effort is directed at how to efficiently generate subgraph candidates. Because subtree isomorphism detection can be solved in $\mathcal{O}\left(\frac{k^{1.5}}{log\, k}n\right)$ time, the computational complexity is reduced within the context of FSM. Therefore, the survey presented in this paper makes a distinction between FSM and frequent subtree mining. In the rest of this paper, we will continue to use the acronym FSM to mean both frequent subgraph and subtree mining; and the acronyms FGM and FTM to indicate frequent subgraph and subtree mining, respectively, where a distinction is required.

Before considering specific subgraph and subtree mining algorithms in detail (Sections 4 and 5), techniques used to represent graphs and trees will first be considered. The aim here is to represent graphs and trees in such a manner that subgraphs can be enumerated efficiently so as to facilitate the desired FSM.

3.1 Canonical representations

The simplest mechanism whereby a graph structure can be represented is by employing an *adjacency matrix* or *adjacency list*. Using an adjacency matrix, the rows and columns represent vertexes, and the intersection of row i and column j represents a potential edge, connecting the vertexes v_i and v_j. The value held at intersection $<i, j>$ typically indicates the number of links from v_i to v_j. However, the use of adjacency matrices, although straightforward, does not lend itself to isomorphism detection, because a graph can be represented in many different ways depending on how the vertexes (and edges) are enumerated (Washo & Motoda, 2003). With respect to isomorphism testing, it is therefore desirable to adopt a consistent labelling strategy that ensures that any two identical graphs are labelled in the same way regardless of the order in which vertexes and edges are presented (i.e. a *canonical* labelling strategy).

A canonical labelling strategy defines a unique code for a given graph (Read & Corneil, 1977; Fortin, 1996). Canonical labelling facilitates isomorphism checking because it ensures that if a pair of graphs are isomorphic, then their canonical labellings will be identical (Kuramochi & Karypis, 2001). One simple way of generating a canonical labelling is to flatten the associated adjacency matrix by concatenating rows or columns to produce a code comprising a list of integers with a minimum (or maximum) lexicographical ordering imposed. To further reduce the computation resulting from the permutations of the matrix, canonical labellings are usually compressed, using what is known as a *vertex invariant scheme* (Read & Corneil, 1977), that allows the content of an adjacency matrix to be partitioned according to the vertex labels. Various canonical labelling schemes have been proposed, some of the more significant are described in this subsection.

Minimum depth first search code (*M-DFSC*): There are a number of variants of DFS encodings, but essentially each vertex is given a unique identifier (ID) generated from a DFS traversal of a graph (DFS subscripting). Each constituent edge of the graph in the DFS code is then represented by a 5-tuple: (i, j, l_i, l_e, l_j), where i and i are the vertex IDs, l_i and l_j are the labels for the corresponding vertexes, and l_e is the label for the edge connecting the vertexes. On the basis of the DFS lexicographic order, the M-DFSC of a graph g can be defined as the canonical labelling of g (Yan & Han, 2002). The DFS codes for the left-most branch and the right-most branch of the example graph in Figure 5c are {(0, 1, *a*, 1, *b*), (1, 2, *b*, 1, *e*), (2, 3, *e*, 1, *f*), (3, 4, *f*, 1, *c*), (4, 2, *c*, 1, *e*)} and {(0, 9, *a*, 1, *d*), (9, 10, *d*, 1, *f*), (10, 11, *f*, 1, *g*), (11, 9, *g*, 1, *d*)}, respectively.

Canonical adjacency matrix (*CAM*): Given an adjacency matrix M of a graph g, an encoding of M can be obtained by the sequence obtained from concatenating the lower (or upper) triangular entries of M, including entries on the diagonal. Since different permutations of the set of vertexes correspond to different adjacency matrices, the canonical (CAM) form of g is defined as the maximal (or minimal) encoding. The adjacency matrix from which the canonical form is generated defines the *Canonical Adjacency Matrix* or CAM (Inokuchi *et al.*, 2000, 2002; Kuramochi & Karypis, 2001; Huan *et al.*, 2003). The encoding for the example graph given in Figure 5c, represented by the matrix in Figure 5b, is thus $\{a1b00c100d0110e00111f000101g1000010h00000001k000001000w\}$.

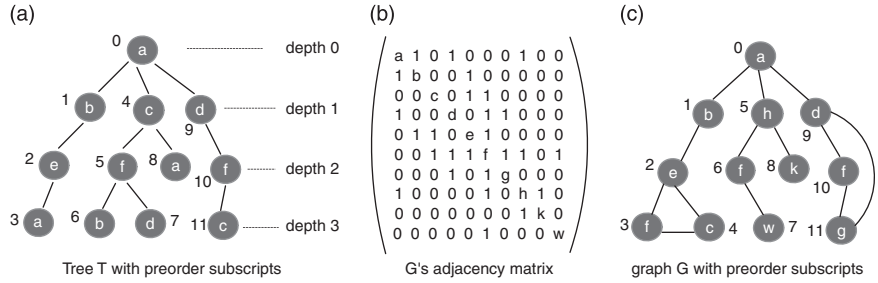

Figure 5 Graph examples to illustrate the canonical representations discussed in Subsection 3.1 (for ease of illustration, all edge labels are assumed to be the same and represented by '1')

The above two schemes are applicable to any simple undirected graph. However, it is easier to define a canonical labelling for trees than graphs, because trees have an inherent structure associated with them. There also exist more specific schemes that are uniquely focused on trees. Among these, DFS-LS and DLS are directed at rooted ordered trees, BFCS and DFCS are used for rooted unordered trees. Each of these will be briefly described below.

DFS label sequence (*DFS-LS*): Given a labelled ordered tree T, the labels of $\forall v_i \in V$ are added to a string S, during a DFS traversal of T. Whenever backtracking occurs, a unique symbol, such as '-1' or '\$' or '/', is added to S (Zaki, 2002, 2005a; Tan *et al*., 2006). The DFS-LS code for the example tree given in Figure 5a is thus {*abea*\$\$\$*cfb*\$*d*\$\$*a*\$\$*dfc*\$\$\$}.

Depth-label sequence (*DLS*): Given a labelled ordered tree T, depth-label pairs comprising the depth and label $\forall v_i \in V$, $(d(v_i), l(v_i))$, are added to a string S during a DFS traversal of T. The DLS of T is defined as $S = \{(d(v_1), l(v_1)), \ldots, (d(v_k), l(v_k))\}$ (Asai *et al*., 2002; Wang *et al*., 2004a). The DLS code for the example tree given in Figure 5a is {(0,*a*), (1,*b*), (2,*e*), (3,*a*), (1,*c*), (2,*f*), (3,*b*), (3,*d*), (2,*a*), (1,*d*), (2,*f*), (3,*c*)}.

Breadth-first canonical string (*BFCS*): For a labelled ordered tree, every vertex label is added to a string S, by traversing the tree in a BFS manner. In addition, a '\$' symbol is used to partition the families of siblings, and a '#' symbol is used to indicate the end of the string encoding. '\$' is considered to be lexicographically before '#' and both of them order greater than any other vertex and edge labels. Given an unordered tree T, different ordered trees with corresponding BFS string encodings can be produced by imposing different orderings on the children of the intermediate vertexes. The BFCS of T is the lexicographically minimal of these encodings, and the corresponding rooted ordered tree defines the breadth-first canonical form (BFCF) of T (Chi *et al*., 2005). BFCS's variants can be found in Chi *et al*. (2003, 2004a). Thus, the BFS string encoding of the example tree given in Figure 5a is *a*\$*bcd*\$*e*\$*fa*\$*f*\$*a*\$*bd*\$\$*c*#.

Depth-first canonical string (*DFCS*): Similar to the BFCS but using DFS, the depth-first string encoding, for a labelled ordered tree, labels each vertex by traversing the tree in a DFS manner. The DFCS of a unordered tree T is then the minimal of all the possible DFS encodings, according to the lexicographical ordering. The corresponding rooted ordered tree defines the depth-first canonical form (DFCF) of T (Chi *et al*., 2005). DFCS's variants can also be found in Chi *et al*. (2003, 2004b). The DFS string encoding of the example tree given in Figure 5a is *abea*\$\$\$\$*cfb*\$*d*\$\$*a*\$\$*dfc*\$\$\$#.

Canonical representation of free trees: Free trees do not have roots. In this case, a unique representation for a free tree is usually constructed by selecting one vertex or a pair of vertexes as the root(s). The procedure starts with removing all leaf vertexes and their incident edges recursively until a single vertex or two adjacent vertexes are left. In the first case, the remaining vertex is called the centre, and a rooted unordered tree is obtained with the centre as the root. The procedure is displayed in Figure 6a. In the second case, the pair of remaining vertexes are called the bi-centre; a pair of rooted unordered trees are obtained with the bi-centre as the roots (along with an edge connecting the two roots). The procedure is

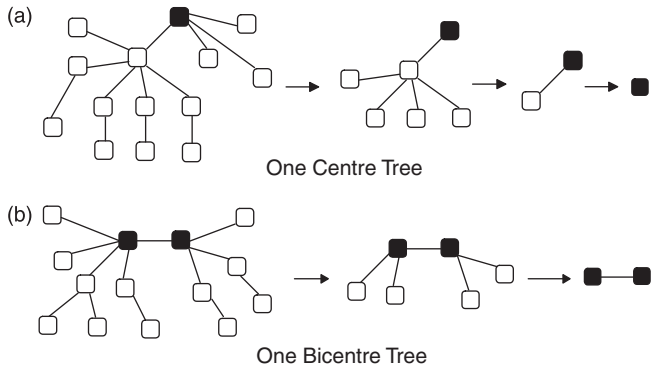

Figure 6 An example of two types of free trees

illustrated in Figure 6b. Thus, a pair of trees is ordered so that the root of the smaller one is chosen as the root of the whole tree (Chi *et al.*, 2003; Rückert & Kramer, 2004). After obtaining rooted unordered trees, any canonical representations for rooted unordered trees (see above) can be employed to represent the free trees.

3.2 Candidate generation

As noted earlier, candidate generation is an essential phase in FSM. How to systematically generate candidate subgraphs without redundancy (i.e. each subgraph should be generated only once) is a key issue. Many FSM algorithms can be characterized by the strategy adopted for candidate generation. A number of the most significant are briefly described below. Since a significant proportion of strategies employed in FTM is interwoven with those employed in FGM, no clear distinction can be made between candidate generation strategies in terms of FTM and FGM, that is, strategies initially proposed for (say) FGM are equally applicable to FTM and vice versa.

3.2.1 Level-wise join

The level-wise join strategy was introduced by Kuramochi and Karypis (2001). Basically, a $(k + 1)$ subgraph[6] candidate is generated by combining two frequent k subgraphs that share the same $(k - 1)$ subgraph. This common $(k - 1)$ subgraph is referred to as a *core* for these two frequent k subgraphs. The main issue concerning this strategy is that one k subgraph can have at most k different $(k - 1)$ subgraphs and the joining operation may generate many redundant candidates. In Kuramochi and Karypis (2004a), this issue was addressed by limiting the $(k - 1)$ subgraphs to the two $(k - 1)$ subgraphs with the smallest and the second smallest canonical labels. By carrying out this adapted join operation, the number of duplicate candidates generated was significantly reduced. Other algorithms that adopted this strategy, and its variants, are AGM (Inokuchi *et al.*, 2000), DPMine (Vanetik *et al.*, 2002; Gudes *et al.*, 2006), and HSIGRAM (Kuramochi & Karypis, 2005), which will be discussed later.

3.2.2 Right-most path expansion

Right-most path expansion is the most common candidate generation strategy, which generates $(k + 1)$-subtrees from frequent k-subtrees by adding vertexes only to the right-most path of the tree (Asai *et al.*, 2002, 2003; Zaki, 2002; Nijssen & Kok, 2003). In Figure 7a, 'RMB' denotes the right-most branch, which is the path from the root to the right-most leaf $(k - 1)$, and a new vertex k is added by attaching it to any vertexes along the RMB. An enumeration DAG using right-most

[6] k refers to the expansion unit for growing the candidate subtrees, which can be expressed in terms of vertexes, edges.

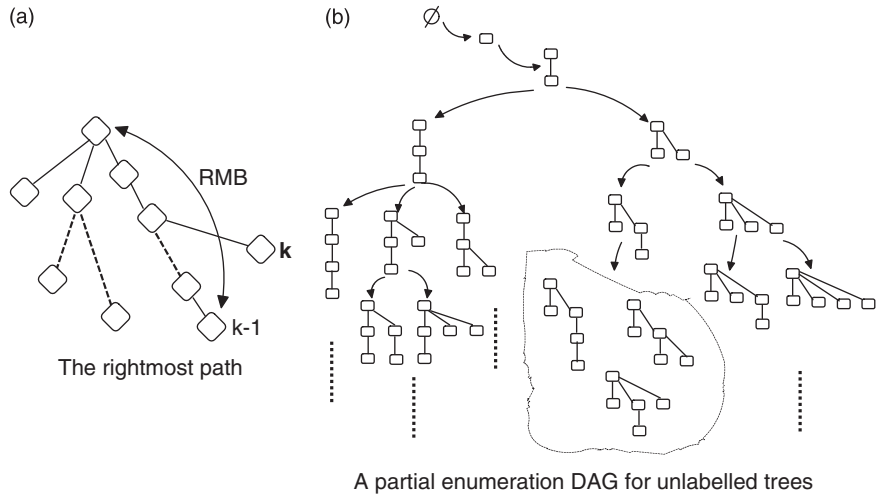

Figure 7 An illustration of right-most path expansion

expansion is a tree with a root ϕ, where each node is a subtree pattern. A node S is linked by another node T if and only if T is a right-most expansion of S. Every 1-subtree is a right-most expansion of the root ϕ and every $(k+1)$-subtree is a right-most expansion of the k-subtree. Hence, all subtree patterns can be enumerated by traversing in either BFS or DFS manner (Asai *et al*., 2002). Figure 7b shows a part of an enumeration DAG grown by right-most path expansion. Each square in the figure represents a vertex in the tree. An enumeration DAG (sometimes also simplified as an *enumeration tree*) is used to illustrate how a set of patterns is completely enumerated in a search problem. Enumeration DAGs have been used extensively in ARM (Bayardo, 1998; Agrawal *et al*., 2001); and subsequently, in a variety of ways, by many subtree mining algorithms (Asai *et al*., 2002, 2003; Nijssen & Kok, 2003; Chi *et al*., 2004a, 2005).

3.2.3 Extension and join

The extension and join strategy was first proposed by Huan *et al*. (2003), and later used by Chi *et al*. (2004a). It employed a BFCS representation, whereby a leaf at the bottom level of a BFCF tree is defined as a 'leg'. For a node 'V_n' in an enumeration tree, if the height of the BFCF tree corresponding to 'V_n' is assumed to be h, all children of 'V_n' can be obtained by either of the following two operations:

(a) *Extension operation*: Adding a new leg at the bottom level of the BFCF tree yields a new BFCF with height $h+1$.
(b) *Join operation*: joining 'V_n' and one of its sibling yields a new BFCF with height h.

3.2.4 Equivalence class-based extension

Equivalence class-based extension (Zaki, 2002, 2005b) is founded on a DFS-LS representation for trees. Basically, a $(k+1)$-subtree is generated by joining two frequent k-subtrees. The two k-subtrees must be in the same *equivalence class* $[C]$[7]. An equivalence class consists of the class prefix encoding and a list of members. Each member of the class can be represented as a (l, p) pair, where l is the k-th vertex label and p is the depth-first position of the k-th vertex's parent. It is verified, in Zaki (2002), that all potential $(k+1)$-subtrees with the prefix $[C]$ of size $(k-1)$ can be generated by joining each pair of members of the same equivalent class $[C]$.

[7] In Zaki (2002), two k-subtrees T_1, T_2 are in the same 'prefix' equivalence class if and only if they share the same encoding up to the $(k-1)$-th vertex.

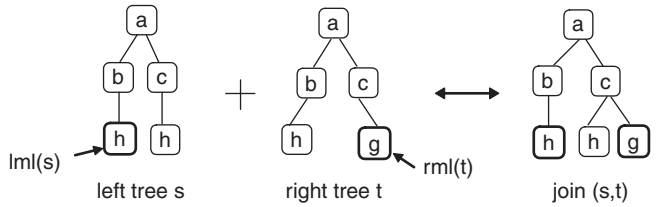

Figure 8 An illustration of right-and-left tree join

3.2.5 Right-and-left tree join

The right-and-left tree join strategy was proposed by Hido and Kawano (2005). It essentially uses the right-most leaf (see 2) and left-most leaf[8] of the tree to generate candidates in a BFS manner. Let $lml(T)$ denote the left-most leaf of T and $Right(T)$ the right-most tree obtained by removing $lml(T)$; and let $rml(T)$ denote the right-most leaf and $left(T)$ the left tree obtained by removing $rml(T)$. Given two trees s and t, where $Right(s) = Left(t)$, their right-and-left tree join is defined as: $join(s, t) = s \cup rml(t) = lml(s) \cup t$ A diagram depicting this join operation is given in Figure 8.

Among these candidate generation strategies, the level-wise join and the extension and join are directed at FGM, and all others at FTM.

4 Frequent subtree mining algorithms

The previous section considered the joint issues of representation (canonical forms) and candidate generation, in terms of both trees and graphs. In this section, a number of prominent FTM algorithms are reviewed. FTM has attracted a great deal of research interest in areas such as: network IP multicast[9] routing (Cui *et al.*, 2005), web usage mining (Zaki, 2005b), computer vision (Liu & Geiger, 1999), XML mining (Zaki & Aggarwal, 2003; Tan *et al.*, 2005), bio-informatics (Hein *et al.*, 1996; Rückert & Kramer, 2004; Zhang & Wang, 2006), and so on. The attraction of frequent subtree mining is that subgraph isomorphism detection becomes subtree isomorphism detection, which can be solved in $\mathcal{O}\left(\frac{k^{1.5}}{log\, k} n\right)$ time (Shamir & Tsur, 1999). In addition, the structure of trees may be usefully employed to simplify the overall mining process.

The FTM algorithms discussed in this section have been categorized as in Table 3, according to the nature of the trees that the FTM algorithm is directed at: (i) unordered trees, (ii) ordered trees, (iii) free trees, or (iv) hybrid trees (any combinations of (i), (ii), and (iii)). The algorithms are also categorizations according to the nature of the subtrees to be output (*maximal subtrees*, *closed subtrees*, *induced subtrees*, or *embedded subtrees*), and the nature of the support metrics employed (transaction-based counting, denoted by T_c; or occurrence-based counting, denoted by O_c). For an alternative review of FTM algorithms, readers may like to refer to Chi *et al.* (2004), who provide a theoretical foundation and performance study of a representative collection of FTM algorithms proposed prior to 2004.

4.1 Unordered tree mining

Labelled unordered trees are often used to model structural data, two popular areas of application are the analysis of chemical compounds and the hyper-link structure of the web (Asai *et al.*, 2003).

Unordered tree FTM tends to use DLS-LS, DLS, or BFCS to represent the trees (as described in Subsection 3.1). An often cited, example of a DLS-DS-based algorithm is the SLEUTH Algorithm (Zaki, 2005a). SLEUTH is founded on earlier work directed at the FTM of other

[8] The left-most leaf of the tree is the first leaf vertex in the DFS traversal of that tree (Hido & Kawano, 2005).

[9] *IP multicast*: a method for building multicast trees at the Internet Protocol layer so as to send packets to multiple receivers in a single transmission (Paul, 1998).

Table 3 Categorization of common frequent subtree mining algorithms

	Maximal	*Closed*	*Induced*	*Embedded*	T_c	O_c
Unordered tree mining						
TreeFinder	*			*	*	
uFreqT			*		*	
cousinPair				*	*	
RootedTreeMiner			*		*	
SLEUTH				*	*	
Ordered tree mining						
FREQT			*		*	
TreeMiner				*	*	
Chopper				*	*	
XSpanner				*	*	
AMIOT			*		*	
IMB3-Miner			*	*		*
TRIPS			*		*	
TIDS			*		*	
Free tree mining						
FreeTreeMiner			*		*	
FTMiner			*		*	
F3TM			*		*	
CFFTree		*	*		*	
Hybrid tree mining						
CMTreeMiner	*	*	*		*	
HybridTreeMiner			*		*	

types of trees. This algorithm uses *scope-lists* to compute the support. Zaki *et al.* (2005a) considered two extension mechanisms for candidate generation: (i) *class-based extension* and (ii) *canonical extension*. Using class-based extension, not all candidates generated by this mechanism necessarily adhere to the desired canonical form, consequently it is necessary to check each candidate subtree to ensure that it is in canonical form. Alternatively, canonical extension can be applied only to canonical frequent subtrees that have a known frequent edge; however, this results in many infrequent but canonical candidates. As noted by Zaki (2005a), there is a trade-off between using the two extension mechanisms. Experiments conducted by Zaki (2005a) demonstrated that using class-based extension is more efficient than canonical extension.

An established example of an unordered tree FTM algorithm that uses the DLS representation is uFreqT (Nijssen & Kok, 2003). At the candidate generate phase, uFreqT uses the right-most path expansion technique to generate candidates. At the support counting phase, a tree mapping algorithm, to determine the frequency of the current pattern, is translated into a more computationally efficient maximum bipartite-matching algorithm. In order to facilitate this support counting, uFreqT maintains a data structure in which to store all potential mappings for the vertexes on the right-most path and pointers to the parent mappings.

Chi *et al.* (2005) presented an algorithm, RootedTreeMiner, founded on a BFCS encoding. Unlike SLEUTH or uFreqT, RootedTreeMiner is directed at finding only frequent induced subtrees. Thus, at the candidate generation phase, the range of allowable vertexes at a given position can be computed beforehand. At the support counting phase, an *occurrence list* is first built for each discovered subtree t. This list records the ID of each graph transaction in the tree data set that contains t, along with the mapping between the vertex indexes in t and those in the transaction. Using this occurrence list, the support of t equates to the number of elements in the list that have distinct IDs.

The above all employ exact matching techniques. An example of an unordered tree FTM that employs inexact matching is TreeFinder (Termier *et al.*, 2002). Treefinder employs an

Apriori-based approach, using an ancestor–descendant relationship, to mine embedded subtrees. Algorithms that employ inexact matching are, of course, not guaranteed to discover the complete set of frequent subtrees; however, they tend to be very efficient.

Some unordered tree FTM algorithms are directed at specific applications and can use features of these applications to enhance the efficiency of the operation of the algorithm. For example, Shasha *et al.* (2004) presented an unordered tree FTM algorithm, cousinPair, for application to phylogeny. They defined an interesting pattern as being a 'cousin pair', a pair of vertexes satisfying some cousin distance and minimum occurrence threshold. By using such constraints, interesting patterns were mined from a tree database. The objective here was to get a better understanding of the evolutionary history of species. The obvious disadvantage of algorithms such as cousinPair is that they are not generally applicable.

4.2 Ordered tree mining

In contrast to unordered tree mining, the ordering inherent in ordered trees can be used to introduce efficiencies with respect to subtree generation and subtree isomorphism testing. Candidate subtrees are typically grown using right-most path extension or equivalence class-based extension. For example, Asai *et al.* (2002) used right-most expansion with respect to their FREQT algorithm. In addition, only the right-most leaf occurrences of the patterns are saved so as to make the support counting more efficient. Asai *et al.* modelled semi-structured data (namely Web pages), using a labelled ordered tree to evaluate FREQT.

The advantage of right-most expansion with respect to ordered trees is that the generation of duplicate candidate sets can be avoided. Hido and Kawano (2005) noted that enumeration using right-most path expansion, as adopted by FREQT and other FSM algorithms, tends to generate many non-frequent candidates, thus resulting in unnecessary support counting. Consequently, Hido and Kawano introduced an algorithm, AMIOT, to utilize a new enumeration scheme to reduce the number of non-frequent candidates while maintaining the advantage offered by right-most expansion. This scheme, *right-and-left tree join*, guaranteed that the set of subtree candidates was always a subset of that achieved by the enumeration using right-most path expansion. The performance of AMIOT, with respect to both synthetic data and XML data, demonstrated that it was scalable and performed faster than FREQT. However, the memory usage of AMIOT is larger than that of FREQT, due to the nature of the BFS strategy used by AMIOT.

Zaki (2002) proposed a FTM algorithm, TreeMiner, that uses equivalence class-based extension (coupled with a DFS-LS representation). The notion of *scope-lists*, later also employed in SLEUTH (see above) was also developed to facilitate fast support counting. Unlike FREQT and AMIOT, TreeMiner is directed at discovering frequent embedded subtrees. The performance of the algorithm was compared with a *base algorithm*, PatternMatcher, which employed a BFS strategy. The experimental results demonstrated that TreeMiner outperformed PatternMatcher when applied to real data. However, the pruning technique adopted by TreeMiner is not as efficient as that used by PatternMatcher, given a low support threshold. TreeMiner is a frequently referenced FTM algorithm.

Wang *et al.* (2004a) also proposed an algorithm, Chopper, to mine frequent embedded subtrees from tree data sets, but used a DLS representation. Chopper first uses a revised PrefixSpan (Pei *et al.*, 2001) to mine frequent sequential patterns. Then the tree database is again scanned, with reference to the discovered sequential patterns, to generate candidate patterns and support counts. The two processes of sequential pattern mining and subtree pattern verification are separated in Chopper, and thus an additional computational overhead is incurred. In order to improve the efficiency of Chopper, the XSpanner algorithm was subsequently produced to integrate the sequential pattern mining into the process of subtree pattern verification. Using projected database techniques, XSpanner grew larger frequent subtrees from smaller ones starting from one vertex. Both Chopper and XSpanner outperform TreeMiner when the support threshold is below 5%. However, XSpanner was found to be more stable than Chopper when the support threshold was further reduced.

IMB3-Miner (Tan *et al.*, 2006) is also directed at frequent embedded subtree mining (from ordered tree data sets), but uses a parameter with which to control the *level of embedding*[10]. When the level of embedding is equal to 1, the discovered frequent subtrees are induced subtrees. Thus, by adjusting the embedding level, the algorithm can be used to mine both induced and embedded subtrees. By combining an *Embedding List* data structure with the *TMG* enumeration strategy (a specialized right-most path expansion strategy), IMB3-Miner guarantees that candidate subtrees are generated without duplication. Furthermore, an occurrence list is stored for each generated subtree to speed-up support counting. Unlike the foregoing, instead of using T_c, O_c is employed to calculate the support of patterns. It has been experimentally demonstrated that IMB3-Miner achieves higher performance and scalability than TreeMiner and FREQT. The usage of O_c, instead of T_c, is typically adopted where the repetition and order of the patterns are important.

Tatikonda *et al.* (2006) proposed the TRIPS and TIDS algorithms to mining induced or embedded subtrees in a database of rooted ordered trees. TRIPS uses prüfer sequencing[11] and the left-most path[12] of the pattern as the extension position. TIDS uses DFS sequencing and right-most path extension. The support computation for both algorithms employs an *embedding list*, an array-based structure, to facilitate the recursive generation of the patterns. There is a trade-off between the cost of maintaining the embedding lists and the efficiency of the support computation, when the number of distinct vertex labels is low compared with the total number of vertexes in the data set. Experiments demonstrated that both TRIPS and TIDS performed better than TreeMiner in terms of execution time and memory usage on both synthetic and real data sets. Both TRIPS and TIDS were found to be scalable when the database size increased and able to mine large databases even when using low support threshold values.

4.3 Free tree mining

Free tree mining algorithms, as the name suggests, are directed at the discovery frequent subtrees in collections of free trees. An early example is FreeTreeMiner (Chi *et al.*, 2003) where a *self-join* operation was used for candidate subtree generation and a subtree isomorphism algorithm or support computation (Chung, 1987). Experiments demonstrated that FreeTreeMiner can handle large real data well with a large range of support values; however, it was not found to be scalable when the size of the maximal frequent subtrees was increased, due to the exponential growth of the subtrees.

Similar work conducted by Rückert and Kramer (2004) defined a canonical representation for labelled free trees. This was embedded in a free tree mining algorithm, FTMiner. The algorithm extended more than one vertex at each recursive step during the candidate generation phase. It also adopted the concept of an *extension table*, which is a data structure for storing all the extensions for a subtree pattern along with the set of graph transactions containing the pattern. Utilizing this extension table, the algorithm not only kept track of the frequency of each subtree pattern, but also gathered information required for the extension of the current pattern, thus reducing significantly the number of database scans. Experiments on a large scale database suggest that the algorithm is able to mine frequent patterns among a collection of more than 37 330 chemical compounds using support threshold of 2%.

With the focus mainly on reducing the cost of candidate generation, Zhao and Yu (2006) presented the F3TM free tree mining algorithm. The algorithm introduced the idea of an *extension frontier* to define the positions (vertexes) for growing frequent subtrees in the candidate generation phase, and uses automorphism-based pruning and canonical pruning techniques to enhance the efficiency of candidate generation. Compared with other free tree mining algorithms, performance

[10] The *level of embedding* is defined as 'the length of path between two vertexes that form an ancestor–descendant relationship' (Tan *et al.*, 2006).

[11] The prüfer sequence (Prüfer, 1918) of a labelled tree on n vertexes is a unique $(n-2)$ length sequence that can be formed by an iterative algorithm (Tatikonda *et al.*, 2006).

[12] The path from the root to the left-most leaf is the *left-most path* (Tatikonda *et al.*, 2006).

studies indicated that F3TM was more efficient than FTMiner and FreeTreeMiner with respect to a chemical database of 42 390 compounds. CFFTree (Zhao & Yu, 2007) is an extension of F3TM directed at closed FTM. CFFTree employs a mechanism called *safe position pruning* to grow subtrees only on safe positions, thus introducing efficiencies when deciding which branch of the enumeration tree to prune. In addition, CFFTree employs *safe label pruning* to grow subtrees only on the vertexes with labels lexicographically less than the new 'adding vertex'; this then serves to remove some unnecessary enumeration. The reported evaluation of CFFTree demonstrated that it outperformed its base algorithm, F3TM, using post-processing when finding closed patterns.

4.4 Hybrid tree mining

Hybrid tree mining algorithms are directed at more generic tree formats. As such they can be categorized as being directed at either: (i) unordered or free trees or (ii) ordered or unordered trees. An example of the first is HybridTreeMiner and an example of the second is CMTreeMiner.

HybridTreeMiner (Chi *et al*., 2004a) uses a BFCS representation. In the enumeration tree, each node represents an unordered tree in BFCF. For a node v in the enumeration tree, all the children of v may be generated using either an extension or a join operation. The join operation is applied to pairs of sibling nodes with a height (depth) of h, resulting in a BFCF tree with the same height. The extension operation is applied by extending a new leaf at the bottom level of the BFCF tree with a height of h, resulting in a BFCF tree with a height of $(h + 1)$. This hybrid enumeration strategy was further extended to handle the free tree case. Reported experimental results demonstrated that HybridTreeMIner was faster than FreeTreeMiner, and that its memory usage was also much less than that required by FreeTreeMiner.

CMTreeMiner was introduced to mine both *closed* and *maximal* frequent subtrees in collections of labelled ordered or unordered trees (Chi *et al*., 2004b). By using pruning and heuristic techniques, the enumeration tree was grown only on the branches that can potentially produce closed or maximal frequent subtrees, thus avoiding the computational overhead associated with finding all frequent subtrees. The advantage offered by CMTreeMiner is that it directly mines closed and maximal frequent subtrees without first generating all frequent subtrees. Experimental results showed that: (i) for an ordered tree, database CMTreeMiner outperformed FREQT and (ii) for an unordered tree, database CMTreeMiner ran faster than HybridTreeMiner.

4.5 Summary of frequent subtree mining algorithms

From the foregoing it can be seen that many different methods, techniques, and strategies have been proposed to achieve FTM. From the perspective of applications, these algorithms can be divided into three main domains:

(a) *Web access analysis*: Examples are SLEUTH, RootedTreeMiner, TreeMiner, IMB3-Miner, Chopper, XSpanner, TRIPS, TIDS, CMTreeMiner, and HybridTreeMiner.
(b) *IP multicast analysis*: Examples are FreeTreeMiner and CMTreeMiner.
(c) *Chemical compound analysis*: Examples are FreeTreeMiner, FTMiner, F3TM, CFFTree, and HybridTreeMiner.

From the perspective of the traversing strategy employed in the search space, FTM algorithms can be categorized into two groups:

(a) *BFS strategy*: The BFS strategy has the advantage of performing full pruning, which, however, requires significant memory usage. Examples are RootedTreeMiner, AMIOT, FreeTreeMiner, and HybridTreeMiner.
(b) *DFS strategy*: The DFS strategy has the disadvantage of weak pruning. However, the memory usage is smaller than that required for BFS. Examples are uFreqT, SLEUTH, FREQT, TreeMiner, IMB3-Miner, TIDS, FTMiner, and CMTreeMiner.

Table 4 Summary of popular FTM algorithms and their candidate generation and support computation mechanisms

Algorithm	Candidate generation	Support computation
TreeFinder	Apriori itemset generation	Clustering techniques
uFreqT	Rightmost path expansion	Maximum bipartite matching
SLEUTH	Equivalence class extension	Scope-lists
cousinPair	Cousin distance	Lookup table
RootedTreeMiner	Enumeration tree	Occurrence list
FREQT	Rightmost path expansion	Occurrence list
TreeMiner	Equivalence class extension	Scope list join
Chopper XSpanner	n/a	n/a
AMIOT	Right-and-left tree join	Occurrence list
IMB3-Miner	TMG	Occurrence list
TRIPS	Leftmost path extension	Hash table
TIDES	Rightmost path extension	Hash table
FreeTreeMiner	Self-join	Subtree isomorphism
FTMiner	Extension tables	Support sets
F3TM CFFTree	Enumeration tree + extension Frontier	Ullmann's backtracking Algorithm
CMTreeMiner	Enumeration tree	n/a
HybridTreeMiner	Extension + join	Occurrence list

FTM = Frequent subtree mining; TMG = Tree Model Guided.

Table 4 lists the main techniques used for candidate generation and support counting with respect to the algorithms described in this section. Generally, each frequent subtree mining algorithm has its strengths and weaknesses. There is no universally applicable frequent subtree mining algorithm. In terms of FTM efficiency, the following techniques are considered to offer the best performance:

- DFS sequence and its variants for tree representation.
- DFS strategy for traversing the search space.
- Enumeration tree growth with right-most path expansion in candidate generation phase.
- Occurrence list for support counting.

Examples of algorithms containing at least three of these techniques are SLEUTH, FREQT, TreeMiner, and IMB3-Miner. Among these, FREQT and TreeMiner are usually chosen as base algorithms for comparison with others. TreeMiner is an Apriori-like FTM algorithm, while FREQT is a right-most path expansion style algorithm. These two styles represent two streams within the realm of FTM. Although subtree isomorphism can be solved in $\mathcal{O}\left(\frac{k^{1.5}}{log\, k}n\right)$ time, very few frequent subtree mining algorithms adopted it directly for support counting; occurrence lists are more frequently adopted. The main reason for this is that occurrence list counting is much more straightforward to implement.

5 Frequent subgraph mining algorithms

As was indicated in Figure 1b, FGM algorithms find substantial application in chemical informatics and biological network analysis. There are a variety of FGM algorithms reported in the literature. As in the case of FTM, candidate generation and support counting are key issues. Since subgraph isomorphism detection is known to be *NP*-complete, a significant amount of research work has been directed at various approaches for effective candidate generation. The mechanism employed for candidate generate is the most significant distinguishing feature of such algorithms. An exploration of current well-known FSM algorithms is provided in this section. Interested readers should note that a good review of the theoretical foundation of FGM, prior to 2003, can

be found in Washo and Motoda (2003). A more recent review of mining frequent patterns, including itemsets, subsequence, and subgraphs, appears in Han *et al*. (2007).

For discussion purposes, the FGM algorithms examined in this section are categorized into 'general purpose' and 'pattern dependent' FGM. The distinction is that in the latter case the nature of the patterns to be discovered is in some way specialized or limited because of the nature of the application domain (e.g. we are only interested in subgraphs satisfying some specific constraints). Consequently, knowledge of the nature of these special patterns allows for a reduction of the search space.

5.1 General purpose frequent subgraph mining

In this subsection a number of general purpose FGM algorithms are considered. To aid the discussion the algorithms are categorized according to three criteria: (i) the completeness of the search (exact search or inexact search), (ii) the type of input (transactions graphs or one single graph), and (iii) the search strategy (BFS or DFS).

5.1.1 Inexact frequent subgraphs mining

Inexact search-based FGM algorithms use an approximate measure to compare the similarity of two graphs, that is, any two subgraphs are not required to be entirely identical to contribute to the support count, instead a subgraph may contribute to the support count for a candidate subgraph if it is in some sense similar to the candidate. Inexact search is of course not guaranteed to find all frequent subgraphs, but the nature of the approximate graph comparison often leads to computational efficiency gains. There are only a few examples of inexact FSM algorithms in the literature. However, one frequently quoted example is the SUBDUE algorithm (Cook & Holder, 1994, 2000). SUBDUE uses the minimum description length principle to compress the graph data; and a heuristic beam search method, that makes use of background knowledge, to narrow down the search space. Although the application of SUBDUE shows some promising results in domains such as image analysis and CAD circuit analysis, the scalability of the algorithm is an issue, that is, the run time does not increase linearly with the size of the input graph. Furthermore, SUBDUE tends to discover only a small number of patterns.

Another inexact search-based FGM algorithm is GREW (Kuramochi & Karypis, 2004b). However, GREW is directed at finding connected subgraphs, which have many vertex-disjoint embeddings[13] in single large graphs. GREW uses a heuristic-based approach that is claimed to be scalable, because it employs ideas of *edge contraction* and *graph rewriting*. GREW deliberately underestimates the frequency of each discovered subgraph in an attempt to reduce the search space. Experiments on four benchmark data sets showed that GREW significantly outperformed SUBDUE with respect to: run time, number of patterns found, and size of patterns found.

To the best knowledge of the authors, the two most recent inexact search-based FGM algorithms are gApprox (Chen *et al*., 2007a) and RAM (Zhang & Yang, 2008). The gApprox algorithm uses the notion of an upper-bound for support counting, and an approximation measure to discover frequent approximately connected subgraphs in very large networks. Empirical studies based on protein–protein interaction networks indicated that gApprox is efficient and that the discovered patterns were biological meaningful. RAM is founded on a formal definition of frequent approximate patterns in the context of biological data represented as graphs, where the edge information tended to be inaccurate. Reported experiments showed that RAM can discover some important patterns that cannot be found by exact search-based mining algorithms.

5.1.2 Exact frequent subgraphs mining

Exact FGM algorithms are much more common than inexact search-based FGM algorithms. They can be applied in the context of graph transaction-based mining or single graph-based

[13] Two embeddings in a graph G are *vertex-disjoint*, if they do not share any vertexes in G.

mining. A fundamental feature for exact search-based algorithms is that the mining is complete, that is, the mining algorithms are guaranteed to find all frequent subgraphs in the input data. As noted in Kuramochi and Karypis (2004b), such complete mining algorithms perform efficiently only on sparse graphs, with a large amount of labels for vertexes and edges. Due to this completeness restriction, these algorithms undertake extensive subgraph isomorphism comparison, either explicitly or implicitly, resulting in a significant computational overhead.

We will commence the discussion of exact FGM algorithms by considering graph transaction-based FGM, the mining of collections of relatively small graphs; single graph-based FGM will be considered at the end of this subsection. With respect to graph transaction mining, the algorithms can be divided into two groups: BFS and DFS, according to the traversing strategy adopted. BFS tends to be more efficient in that it allows for the pruning of infrequent subgraphs (at the cost of high I/O and memory usage) at an early stage in the FGM process, whereas DFS requires less memory usage (in exchange for less efficient pruning). We will consider the BFS algorithms first.

As in the case of ARM algorithms, such as Apriori (Agrawal & Srikant, 1994), BFS-based FGM algorithms utilize the DCP, that is, a $(k + 1)$ subgraph cannot be frequent if its immediate parent k subgraph is not frequent. Using BFS, the complete set of k candidates is processed before moving on to the $(k + 1)$ candidates, where k refers to the expansion unit for growing the candidates, which can be expressed in terms of vertexes, edges, or disjoint paths. Four well-established exact FGM algorithms are itemized below:

- AGM (Inokuchi *et al.*, 2000) is a well-established algorithm used to identify frequent induced subgraphs. AGM uses an adjacency matrix to represent graphs and a level-wise search to discover frequent subgraphs. AGM assumes that all vertexes in a graph are distinct. The evaluation of AGM on chemical carcinogenesis data demonstrated that it was more efficient than an inductive logic programming-based approach combined with a level-wise search. AGM discovers not only connected subgraphs, but also unconnected subgraphs with several isolated graph components. A more efficient version of AGM, called AcGM, has also been developed to mine only frequent connected subgraphs (Inokuchi *et al.*, 2002). AcGM uses the same principles and graph representation as AGM. Experimental results indicate that AcGM is significantly faster than AGM and FSG (see below). Inokuchi *et al.* have further extended their original work to mine frequent induced subgraphs from general graph databases that can contain directed (or undirected), labelled (or unlabelled) graphs and even loops (Inokuchi *et al.*, 2003).
- FSG (Kuramochi & Karypis, 2001, 2004a) is directed at finding all frequent connected subgraphs. FSG uses the BFS strategy to grow candidates, whereby pairs of identified frequent k subgraphs are joined to generate $(k + 1)$ subgraphs. FSG uses a canonical labelling method for graph comparison and computes the support of the patterns using a vertical *transaction list* data representation, which has been used extensively in FTM. Experiments show that FSG does not perform well when graphs contain many vertexes and edges that have identical labels because the *join* operation used by FSG allows multiple automorphism[14] of single or multiple cores[15].
- The FSG algorithm is directed at graph databases consisting of a two-dimensional arrangement of vertexes and edges in each graph (sometimes referred to as topological graphs). However, in chemical compound analysis, users are often interested in graphs that have coordinates associated with the vertexes in two- or three-dimensional space (sometimes referred to as geometric graphs). gFSG (Kuramochi & Karypis, 2002) extends the FSG algorithm to discover frequent geometric subgraphs with some degree of tolerance among geometric graph transactions. The extracted geometric subgraphs are rotation, scaling, and translation invariant.

[14] *Automorphism* is a graph isomorphism to itself via a non-identity mapping.

[15] In the candidate generation phase, a *core* is a common $(k - 1)$ subgraph shared by two frequent k subgraphs. Two frequent k subgraphs are eligible for joining only if they contain the same core (Kuramochi & Karypis, 2001).

gFSG shares the approach of candidate generation with FSG. In order to speed up the computation of geometric isomorphism, a number of topological properties and geometric transform invariants are used in the matching process. In the process of support counting, geometric transform invariants (such as an edge-angle list)[16] and transaction lists are used to facilitate the computation. Experimental evaluation was performed using a chemical database with more than 20 000 chemical compounds to show that gFSG operated well, with low support values and scaled linearly with respect to data size.

- DPMine (Vanetik *et al*., 2002; Gudes *et al*., 2006) uses edge-disjoint paths as the expansion units for candidate generation. Use of a large expansion unit reduces the number of candidates that are generated. First, DPMine identifies all frequent paths; second, it finds all subgraphs with two paths; and third, it merges pairs of frequent subgraphs with $(k-1)$ paths, which have $(k-2)$ paths in common, in order to obtain subgraphs with k paths. Experimental results indicated that the support computation was the most significant contributor to the overall computation time. Gudes *et al*. (2006) also suggested that reducing the support computation overhead is more important than reducing the candidate generation computation overhead. (DPMine can operate on both graph transaction-based and single graph-based data.)

FGM algorithms that adopt a DFS strategy tend to need less memory because they traverse the lattice of all possible frequent subgraphs in a DFS manner. Five well-known example algorithms are listed below:

- MoFa (Borgelt & Berthold, 2002) is directed at mining frequent connected subgraphs describing molecules. The algorithm stores the embedding list of previously found subgraphs, and the extension operation is restricted only to these embeddings. MoFa also uses structural pruning and background knowledge to reduce support computation. However, MoFa still generates many duplicates, resulting in unnecessary support computation.
- gSpan (Yan & Han, 2002) uses a canonical representation, M-DFSC, to uniquely represent each subgraph. The algorithm uses DFS lexicographic ordering to construct a tree-like lattice over all possible patterns, resulting in a hierarchical search space called a DFS code tree. Each node of this search tree represents a DFS code. The $(k+1)$-th level of the tree has nodes which contain DFS codes for k subgraphs. The k subgraphs are generated by one edge expansion from the k-th level of the tree. This search tree is traversed in a DFS manner, and all subgraphs with non-minimal DFS codes are pruned so that redundant candidate generations are avoided. Instead of keeping the embedding list, gSpan only preserves the transaction list for each discovered pattern; subgraph isomorphism detection only operates on the graphs within the list. In comparison with embedding list-based algorithms, the gSpan algorithm saves on memory usage. Experiments show that gSpan outperforms FSG by an order of magnitude. gSpan is arguably the most frequently cited FSM algorithm.
- ADI-Mine (Wang *et al*.,, 2004b) addresses the issue of mining large disk-based graph data sets. ADI-Mine uses a general index structure, *ADI*. Experiments have indicated that ADI-Mine can mine graph data sets with one million graphs, while gSpan could only mine databases with 300 000 graphs.
- FFSM (Huan *et al*., 2003) is directed at graphs that are large and dense with a small number of labels. For example, protein structure mining. FFSM adopted the CAM representation. Thus, a tree-like structure, a suboptimal CAM tree, was constructed to include all possible patterns. Each node in this suboptimal CAM tree can be enumerated by either a join or an extension operation. FFSM records embedding lists for each discovered pattern to avoid explicit subgraph isomorphism testing in the support counting phase. Performance evaluation, using several chemical data sets, indicated that FFSM outperformed gSpan.

[16] An edge-angle list is a multi-set where each element represents the angle formed by two distinct edges sharing the same end points.

- GASTON integrates frequent path, subtree, and subgraph mining into one algorithm, due to the observation that most frequent sub-structures in molecular databases are free trees (by Nijssen & Kok, 2004). The algorithm provided a solution by splitting up the FSM process into path mining, then subtree mining, and finally subgraph mining. Consequently, the subgraph mining is only invoked when needed. Thus, GASTON operates best when the graphs are mainly paths or trees, because the more expensive subgraph isomorphism testing is only encountered in the subgraph mining phase. GASTON records the embedding list so as to grow only patterns that actually appear, thus saving on unnecessary isomorphism detection. Experiments show that GASTON is at a competitive level with a wide range of other FGM algorithms.

Because of the diversity of FGM algorithms, it is difficult to enumerate the strong and weak points of the various algorithms. However, Wörlein *et al*. (2005) presented a detailed comparison of four DFS-based miners: MoFa, gSpan, FFSM, and GASTON, with respect to their performance on various chemical data sets. In the experiments, they found that the use of embedding lists did not offer significant gains at the expense of memory usage. They also confirmed that using canonical representations for duplicate detection required less computation than explicit subgraph isomorphism detection. By utilizing the two main distinguishing features of molecular data, 'symmetries in molecules' and 'non-uniform frequency distribution of atom and bond types', Jahn and Kramer (2005) optimized the performance of gSpan with respect to the mining molecular databases.

We complete this subsection by considering *Single graph*-based exact FGM algorithms, where the frequency of a pattern is determined by occurrence-based counting (we have already noted that DPMine can operate on both graph transaction-based and single graph-based data). A fundamental issue regarding single graph-based mining is how to define the support of the pattern. The DCP, often used to prune the search space when using transaction-based counting, does not hold in the case of occurrence-based counting. Thus, occurrence-based support measures that satisfy the DCP are desirable. One well-established type of occurrence-based support measure that maintains the DCP is founded on the concept of *overlap graphs*[17]. By building a overlap graph for each pattern, the occurrence-based support measure is defined as the size of the Maximum Independent Set (MIS) of vertexes in the overlap graph. The MIS measure was first introduced in Vanetik (2002) and Kuramochi and Karypis (2004c, 2005). In Vanetik *et al*. (2006), the formal definitions were provided together with proofs for the sufficient and necessary conditions required for occurrence-based support measures to maintain the DCP. Their work was further extended to introduce a new occurrence-based support measure, which maintained the DCP, and was computable in polynomial time (Calders *et al*., 2008).

Kuramochi and Karypis (2004c, 2005) proposed two algorithms: HSIGRAM and VSIGRAM to find all frequent subgraphs in a large sparse graph. These two algorithms used BFS and DFS strategies, respectively, and the support of each pattern was determined by the overlap graph-based MIS measure (Vanetik *et al*., 2006). Several variations of the MIS measures, including exact and approximate MIS measures, were implemented. Experiments demonstrated that both algorithms scaled well when mining large graphs, although VSIGRAM was faster than HISGRAM. The reason for the performance advantage of the VSIGRAM algorithm is that it keeps track of the embeddings of the frequent subgraphs along the DFS path, resulting in less subgraph isomorphism checking. In comparison with SUBDUE, the results indicated that SUBDUE performed worse than both HSIGRAM and VSIGRAM; SUBDUE tends to focus on small subgraphs with high frequency and consequently tends to miss significant patterns. Kuramochi and Karypis' work was further extended by Schreiber and Schwöbbermeyer (2005) to mine frequent patterns of a given size, but considering alternative frequency concepts. This frequency-based algorithm, FPF, was applied to two different

[17] An *overlap graph* for a given pattern with a set of all embeddings (occurrences) is a constructed graph where each vertex represents a non-identical embedding of the pattern; two vertexes are connected if the corresponding embeddings overlap (Kuramochi & Karypis, 2005).

biological networks to discover network motifs[18]. Surprisingly, a comparison of the number of frequent patterns found using the alternative frequency concepts demonstrated that the frequency of a pattern alone was not sufficient to identify network motifs, and that it was not clear whether frequent patterns could play functional roles in the biological network.

5.2 Pattern dependent frequent subgraph mining

In FSM, users are usually interested in a certain type of pattern, rather than the complete set of patterns, that is, some subset of the set of all frequent subgraphs. Such 'special patterns' are diagnosed according to their topology and/or some specific constraint (limitation) on the nature of the patterns. Pattern dependent FGM algorithms can be grouped according to nature of the patterns they are directed at: (i) relational patterns, (ii) maximal and closed pattens, (ii) cliques, and (iv) other constrained patterns. Each is discussed in more detail below.

5.2.1 Relational pattern mining

Relational graphs are suitable for modelling large scale networks such as biological or social networks. Yan *et al.* (2005a) indicated that relational pattern mining has three features, which serve to differentiate it from general purpose FSM: (i) the data have distinct vertex labels, (ii) the data comprise very large graphs, and (iii) a focus on frequent patterns with certain *connectivity constraints* (e.g. the minimum degree[19] of a pattern). Thus, relational graph mining aims to identify all frequent patterns displaying a specified connectivity constraint.

CLOSECUT and SPLAT, both proposed by Yan *et al.* (2005a), are directed at mining (closed) frequent subgraphs with connectivity constraints. CLOSECUT uses a pattern growth approach to integrating connectivity constraints, together with graph condensation and decomposition techniques. The SPLAT algorithm uses a pattern reduction approach to integrating the graph decomposition technique. Experiments indicated that CLOSECUT performed better than SPLAT on patterns with low connectivity when using a high support threshold value; however, SPLAT performed better than CLOSECUT on patterns with high connectivity when using a low support threshold value. The results, with respect to biological data, showed that both algorithms could find interesting patterns with strong biological meanings.

5.2.2 Mining maximal and closed patterns

The number of possible frequent subgraphs increases exponentially with the size of the graph, that is, for a frequent k-graph, the number of its frequent subgraphs can be as large as 2^k. In Yan and Han (2003), it was observed that about 1 000 000 frequent graph patterns were generated from 422 chemical compounds (using a support threshold of 5%); among these many were found to structurally repetitive. Therefore, both *closed* and *maximal* FGM approaches have been proposed as mechanisms to limit the number of frequent subgraphs generated. These approaches are discussed further in this subsection. The following notation is used: MFS denotes the set of maximal frequent subgraphs, CFS denotes the set of closed frequent subgraphs, and FS denotes the set of all frequent subgraphs in the graph database. Thus, $MFS \subseteq CFS \subseteq FS$.

Let $MFS = \{g | g \in FS \wedge \neg(\exists h \in FS \wedge g \subset h)\}$. The task of maximal FSM is to find all graphic patterns that belong to MFS. Maximal frequent subgraphs encode the maximal common structures; in the case of biological networks, these are deemed to be the most interesting patterns (e.g. Koyutürk *et al.*, 2004). However, the frequency of non-maximal subgraphs is not produced. Two example maximal FGM algorithms are SPIN and MARGIN.

SPIN (Huan *et al.*, 2004) is a spanning tree-based FSM algorithm designed to discover only maximal frequent subgraphs with the intention of reducing the overall computation cost.

[18] *Network Motifs* are defined as 'patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks' (Milo *et al.*, 2002).

[19] The *minimum degree* of a pattern g is the minimum of the degree of v, for all $v \in V(g)$ (Yan *et al.*, 2005a).

The concept of *tree-based equivalence classes* is introduced by the notion of a *canonical spanning tree*[20]. In SPIN, the graph partitioning method utilizes such tree-based equivalence classes together with three pruning techniques. The algorithm has two main phases: (i) identification of all frequent subtrees within the input data using an appropriate frequent subtree mining algorithms and (ii) the detection of all frequent subgraphs whose canonical spanning tree is isomorphic to each discovered frequent subtree. The desired maximal frequent subgraphs are generated by optimized post-processing. The performance of SPIN has been compared with gSpan and FFSM. Results demonstrated that SPIN displayed significantly better performance than gSpan and FFSM, with respect to both synthetic and chemical data.

MARGIN (Thomas *et al.*, 2006) was founded on the observation that the set of potential maximally frequent subgraphs is included in the set of frequent k subgraphs that have infrequent $(k+1)$ supergraphs. Consequently, the search space of MARGIN is significantly reduced by pruning the lattice around the set of potential maximally frequent subgraphs. The set of candidates is recursively discovered by the core algorithm, *ExpandCut*, and the maximal frequent subgraphs are then found by the post-processing operation. Experimental results showed that MARGIN was computationally faster than gSpan when applied to some databases. However, the efficiency of MARGIN largely relies on the initial *cut*[21].

Let $CFS = \{g | g \in FS \wedge \neg(\exists h \in FS \wedge g \subset h \wedge sup(g) = sup(h))\}$. The task of closed FSM is to find all patterns that belong to *CFS*. These closed patterns have some biological meaning, because generally, a biochemist is only interested in the largest structures with certain properties (Fischer & Meinl, 2004). CLOSECUT and SPLAT are two example closed FGM algorithms that have already been considered (see Subsection 5.2.1). Another example is CloseGraph (Yan & Han, 2003), which is founded on gSpan. CloseGraph uses an equivalent occurrence-based early termination to prune the search space. For the case where the early termination fails and cannot be applied, the detection of the failure of early termination is implemented. Experimental results demonstrated that CloseGraph performed better than gSpan and FSG.

5.2.3 Mining cliques

A clique (or quasi-clique) is a subset of one subgraph with a fixed topology. The first algorithm directed at detecting cliques was proposed by Harary and Ross (1957). Since then, many more algorithms have been devised directed at a variety of the clique detection problems (Bomze *et al.*, 1999; Gutin, 2004). More recently, it has been found that discovering frequent cliques from a set of graph transactions is useful in domains such as communication, finance, and bioinformatics. Example applications where the mining of cliques, or quasi-cliques, has been applied includes: community mining (Abello *et al.*, 2002), gene expression mining (Pei *et al.*, 2005), and the discovery of highly correlated stocks from stock market graphs (Wang *et al.*, 2006). General purpose FGM algorithms can be used to discover such 'special patterns'; however, the computation can be made more efficient if the special properties of cliques is taken into account. Two example clique mining algorithms, CLAN and Cocain, are discussed in the following paragraphs.

CLAN (Wang *et al.*, 2006) is directed at mining frequent closed cliques[22] from large dense graph databases. The algorithm utilized the properties of the clique structure to facilitate clique or sub-clique isomorphism testing by introducing a canonical representation of a clique. Wang *et al.* also devised several pruning techniques to effectively reduce the search space. The experimental results showed that CLAN can efficiently mine large and dense graph data sets. However, the

[20] A *canonical spanning tree* of a graph is defined as the lexicographically maximal spanning tree of the graph (Huan *et al.*, 2004).

[21] A *cut* between two nodes in the lattice is defined as an ordered pair (p, c), where node p represents the parent of node c and p is infrequent while c is frequent (Thomas *et al.*, 2006).

[22] Let $V(g)$ denote the set of vertexes in a graph g, a subset $s \subseteq V(g)$ is a *clique* if the subgraph induced on s is a complete graph.

reported evaluation only used high support threshold values, and scalability was demonstrated using only a small and sparse graph data set.

Extending the work of CLAN, Zeng *et al.* (2006) introduced a general form of clique mining algorithm, Cocain, to mine closed γ-quasi-cliques[23] from large and dense graph data sets. In Cocain, cliques are required to satisfy a user-specified parameter, γ. Cocain utilized the properties of quasi-cliques to prune the search space, combined with a closure checking scheme to speed up the discovery process. However, the reported evaluation of Cocain was only directed at US stock market data.

5.2.4 Constrained pattern mining

The main idea of user constraint-based frequent pattern mining is to integrate constraints into the mining process in order to prune the search space. Zhu *et al.* (2007) presented a framework, called gPrune, to incorporate various constraints into the FSM process. In gPrune, the search spaces of both data and patterns were examined and a new concept, *pattern-inseparable-data-anti-monotonicity* (D-anti-monotonicity), introduced to support effective pruning of the search space. However, an empirical study showed that the benefit of such D-anti-monotonicity pruning was coupled tightly with the speed of the corresponding constraint measure function. Furthermore, experiments indicated that the effectiveness of integrating constraints into the FSM process is influenced by many aspects, including the properties of the data and the pruning cost. Therefore, constraint-based mining algorithms need to take into account the trade-off between the pruning cost and any potential benefit.

5.3 Summary

Table 5 summarizes the approaches to canonical representation, candidate generation, and support counting used by the FGM algorithms described in this section. With respect to the FGM algorithms listed in the Table, SUBDUE, AGM, FSG, MoFa, gSpan, FFSM, and GASTON are the most frequently cited. Among these algorithms, SUBDUE is used more widely than others. However, one frequently quoted disadvantage of SUBDUE is that the algorithm tends to find only small-size patterns, consequently it may miss interesting larger patterns. AGM and FSG are two representative BFS-based miners. MoFa is a specialized miner for molecular data and is able to mine directed graphs. FFSM and GASTON cannot be used in the context of directed graphs, while gSpan, with some minor changes, can accommodate directed graphs.

One common feature for the majority of algorithms in the table is that the search space is usually modelled as a tree-like lattice over all possible patterns, which are ordered lexicographically. Each node in the lattice represents a pattern, and the relationship between patterns at levels $(k + 1)$ and k will only differ by one vertex or edge (i.e. there is a 'parent–child' relation). Therefore, search strategies comprise a traversal of the lattice and storing all patterns that satisfy some threshold. Either a BFS or DFS strategy can be used to traverse the lattice. BFS strategy-based miners offer the advantage over DFS strategy-based miners that they can obtain a much 'tighter' upper bound for the support of k subgraphs from the support associated with the complete set of identified $(k - 1)$ subgraphs. Knowledge of this upper bound can be employed to limit the number of candidate subgraphs that are generated. DFS strategy-based miners typically derive an upper bound for K candidates based only on a single $k - 1$ parent frequent subgraph.

As indicated in Wörlein *et al.* (2005), an efficient FGM algorithm usually displays three distinct features:

- *Restrictive extension*: The extension of a subgraph is valid only when the extension exists in the graphs within the subgraph's occurrence list. Examples of such operations are right-most path expansion as used by gSpan, and right-most path extension as used by MoFa.

[23] A *γ-quasi-clique*($0 \leqslant \gamma \leqslant 1$) is a k subgraph ($k \geqslant 1$), g, where $\forall v \in V(g)$, $degree(v) \geqslant \lceil \gamma(k-1) \rceil$ (Zeng *et al.*, 2006).

Table 5 Summary of popular FGM algorithms and their candidate generation and support computation mechanisms

Algorithm	Representation	Candidate generation	Support computation
AGM/AcGM	CAM	Level-wise join	Database scan
FSG	CAM	Level-wise join	Transaction list
gFSG	n/a	Level-wise join	Edge-angle list Transaction list hybrid
DPMine	n/a	level-wise join	n/a
MoFa	n/a	Extension	Embedding list
gSpan	M-DFSC	Rightmost path extension	Transaction list
ADI-Mine	M-DFSC	Rightmost path extension	Transaction list
FFSM	CAM	Join + extension	Embedding list
GASTON	n/a	Path, tree, and graph enumeration	Embedding list
HSIGRAM	CAM	Level-wise join	Various MIS measures
VSIGRAM	CAM	Extension	Various MIS measures
FPF	CAM	Extension	MIS measure
DPMine	n/a	Level-wise join	n/a
CLOSECUT	M-DFSC	Rightmost path extension	Transaction list
SPLAT	n/a	n/a	n/a
SPIN	n/a	Join	Embedding set
MARGIN	n/a	ExpandCut	n/a
CloseGraph	M-DFSC	Rightmost path extension	Transaction list
CLAN	vertex label sequence	DFS-based extension	n/a
Cocain	vertex label sequence	DFS-based extension	n/a
gPrune	M-DFSC	Rightmost path extension	Transaction list

FGM = frequent subgraph mining; CAM = canonical adjacency matrix; M-DFSC = minimum DFS code; DFS = depth first search; MIS = maximum independent set.

- *Efficient candidate generation*: This operation is achieved by using a canonical graph representation. Such representation can facilitate the filtering out of candidate duplicates before performing graph isomorphism testing. The two main canonical representations are: (i) CAM, used by AGM, FSG, and FFSM; and (ii) M-DFSC, used by gSpan.
- *Essential subgraph isomorphism*: When computing the support of a pattern, a trade-off needs to be sought between using explicitly subgraph isomorphism and keeping embeddings of the pattern. Examples of keeping embeddings are FFSM and GASTON, and instances of using subgraph isomorphism are FSG and gSpan.

Although distributed algorithms can offer a distinct advantage with respect to excessively large databases, very few researchers have used such algorithms for FGM. One example, proposed by Fatta and Berthold (2005), is an extension of MoFa to accommodate the distributed computation of mining frequent patterns with respect to data sets representing large molecular compounds.

6 Discussion and conclusion

A view of the 'state of the art' of current FSM, referencing especially those algorithms most frequently referred to in the literature, has been presented. The most computationally expensive aspects of FSM algorithms are candidate generation and support computation, with the latter being the most computationally expensive. Broadly, the distinguishing feature of the mining algorithms considered in this survey is how they efficiently address candidate generation and support counting.

With reference to the literature, many different mining strategies have been proposed with respect to many different types of graph, to produce many different kinds of patterns. So as to

impose some structure on the wide range of FSM algorithms featured in the literature, we have adopted a categorization whereby FSM algorithms are considered according to: (i) candidate generation strategy, (ii) search strategy, and (iii) approach to frequency counting. Generally, FTM algorithms cannot be directly applied to graphs, while FGM algorithms can be applied to both graphs and trees. FTM and FGM algorithms have been developed differently for different purposes. Thus, in this survey, we describe these two types of algorithms separately. For FTM algorithms, common applications are web usage and XML mining, while FGM algorithms tend to be directed at chem- and bio-informatics. Although there are abundant research publications on FGM applications, many important issues remain to be addressed.

First, can we discover a compact and meaningful set of frequent subgraphs instead of a complete set of frequent subgraphs? A lot of research effort has been directed at reducing the resultant set of frequent subgraphs; for example, the use of maximal frequent subgraphs, closed frequent subgraphs, approximate frequent subgraphs, and discriminative frequent subgraphs. However, there is no clear understanding of what kind of frequent subgraphs are the most compact and representative for any given application. In many cases, the resultant set of frequent subgraphs are too large to be analyzed individually and many of the identified frequent subgraphs are often found to be structurally repetitive. Research work focusing on how to significantly reduce the size of the resultant set of frequent subgraphs is much in demand.

Second, can we achieve better classification using frequent subgraph-based classifiers than other approaches? Can we integrate *feature selection* techniques deeply into the FSM process and directly identify the most discriminative subgraphs which are effective for classification? There is still much room for researchers to utilize classic data mining techniques and integrate them into the FSM process.

Third, as many researchers have noted, exact frequent subgraphs are not very helpful with respect to many real application. Can we therefore devise more efficient algorithms to generate *approximate frequent subgraphs*? Little work has been conducted in the context of approximate frequent subgraphs mining with the notable exception of the well-known SUBDUE algorithm.

Finally, in domains like: document image classification, work-flow mining, social network mining, single graph-based mining, and so on, there is still a lot of work that can be done to improve the mining task. There is always a trade-off between the combinatorial complexity of FSM algorithms and the utility of the frequent subgraphs discovered by them. Much work is needed to circumvent this issue. Is the *frequency* of a subgraph really a good measure for discovering interesting subgraphs? Can we devise other interestingness metrics for subgraph discovery, rather than adopting those from the domain of ARM?

References

Abello, A., Resende, M. G. C. & Sundarsky, S. 2002. Massive quasi-clique detection. In *Proceedings of the 5th Latin America Symposium on Theoretical Informatics*, Cancun, Mexico, 598–612.

Agrawal, R. & Srikant, R. 1994. Fast algorithm for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 487–499.

Agrawal, R. C., Aggarwal, C. C. & Prasad, V. V. V. 2001. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing* **61**(3), 350–371.

Alm, E. & Arkin, A. P. 2003. Biological Networks. *Current Opinion in Structural Biology* **13**(2), 193–202.

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H. & Arikawa, S. 2002. Efficient substructure discovery from large semi-structured data. In *Proceedings of the 2nd SIAM International Conference on Data Mining*, Fukuoka, Japan, 158–174.

Asai, T., Arimura, H., Uno, T. & Nakano, S. 2003. Discovering frequent substructures in large unordered trees. In *Proceedings of the 6th International Conference on Discovery Science*, Fukuoka, Japan, 47–61.

Bayardo, R. J. Jr 1998. Efficiently Mining Long Patterns from Databases. In *Proceedings of the 1998 International Conference on Management of Data*, 85–93.

Bomze, I. M., Budinich, M., Pardalos, P. M. & Pelillo, M. 1999. The maximum clique problem. In *Handbook of Combinatorial Optimization*, Do, D-Z. & Pardalos, P. M. (eds). Kluwer Academic Publishers, **4**, 1–74.

Borgelt, C. & Berthold, M. 2002. Mining molecular fragments: finding relevant substructures of molecules. In *Proceedings of International Conference on Data Mining*, 211–218.

Borgwardt, K. M. & Kriegel, H. P. 2005. Shortest-path kernels on graphs. In *Proceedings of the 2005 International Conference on Data Mining*, 74–81.

Brin, S. & Page, L. 1998. The anatomy of a large-scale hyper-textual web search engine. In *Proceedings of the 7th International World Wide Web Conference*, 107–117.

Bunke, H. & Allerman, G. 1983. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters* **1**(4), 245–253.

Calders, T., Ramon, J. & van Dyck, D. 2008. Anti-monotonic overlap-graph support measures. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, 73–82.

Chakrabarti, S., Dom, B., Gibson, D., Kleinberg, J., Kumar, R., Raghavan, P., Rajagopaln, S. & Tomkins, A. 1999. Mining the link structure of the world wide web. *IEEE Computer* **32**(8), 60–67.

Chen, M. S., Han, J. & Yu, P. S. 1996. Data mining: an overview from database perspective. *IEEE Transaction on Knowledge and Data Engineering* **8**, 866–883.

Chen, C., Yan, X., Zhu, F. & Han, J. 2007a. gApprox: mining frequent approximate patterns from a massive network. In *Proceedings of the 7th IEEE International Conference on Data Mining*, 445–450.

Chen, C., Yan, X., Yu, P. S., Han, J., Zhang, D. & Gu, X. 2007b. Towards graph containment search and indexing. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, 926–937.

Chen, C., Lin, C. X., Yan, X. & Han, J. 2008. On effective presentation of graph patterns: a structural representative approach. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 299–308.

Chi, Y., Yang, Y., Xia, Y. & Muntz, R. R. 2003. Indexing and mining free trees. In *Proceedings of the 2003 IEEE International Conference on Data Mining*, 509–512.

Chi, Y., Nijssen, S., Muntz, R. & Kok, J. 2004. Frequent subtree mining – an overview. *Fundamenta Informaticae, Special Issue on Graph and Tree Mining* **66**(1–2), 161–198.

Chi, Y., Yang, Y., Xia, Y. & Muntz, R. R. 2004a. HybridTreeMiner: an efficient algorithm for mining frequent rooted trees and trees using canonical forms. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, 11–20.

Chi, Y., Yang, Y., Xia, Y. & Muntz, R. R. 2004b. CMTreeMiner: mining both closed and maximal frequent subtrees. In *Proceedings of the 8th Pacific Asia Conference on Knowledge Discovery and Data Mining*, 63–73.

Chi, Y., Yang, Y., Xia, Y. & Muntz, R. R. 2005. Canonical forms for labelled trees and their applications in frequent subtree mining. *Journal of Knowledge and Information Systems* **8**(2), 203–234.

Christmas, W. J., Kittler, J. & Petrou, M. 1995. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(8), 749–764.

Chung, J. C. 1987. $\mathcal{O}(n^{2.5})$ time algorithm for subgraph homeomorphism problem on trees. *Journal of Algorithms* **8**, 106–112.

Conte, D., Foggia, F., Sansone, C. & Vento, M. 2004. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence* **18**(3), 265–298.

Cook, D. J. & Holder, L. B. 1994. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research* **1**, 231–255.

Cook, D. J. & Holder, L. B. 2000. Graph-based data mining. *IEEE Intelligent Systems* **15**(2), 32–41.

Cordella, L. P., Foggia, P., Sansone, C. & Vento, M. 2001. An improved algorithm for matching large graphs. In *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, 149–159.

Cordella, L. P., Foggia, P., Sansone, C., Tortorella, F. & Vento, M. 1998. Graph Matching: a fast algorithm and its evaluation. In *Proceedings of the 14th Conference on Pattern Recognition*, 1582–1584.

Cui, J. H., Kim, J., Maggiorini, D., Boussetta, K. & Gerla, M. 2005. Aggregated multicast – a comparative study. *Cluster Computing* **8**(1), 15–26.

Deshpande, M., Kuramochi, M., Wale, N. & Karypis, G. 2005. Frequent sub-structure-based approach for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering* **17**(8), 1036–1050.

Fan, W., Zhang, K., Cheng, H., Gao, J., Yan, X., Han, J., Yu, P. S. & Verscheure, O. 2008. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, USA, 230–238.

Fatta, G. D. & Berthold, M. R. 2005. High performance subgraph mining in molecular compounds. In *Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC'05)*, 866–877.

Fischer, I. & Meinl, T. 2004. Graph based molecular data mining – an overview. In *Proceedings of the 2004 IEEE International Conference on Systems,Man and Cybernetics*, 4578–4582.

Flake, G., Tarjan, R. & Tsioutsiouliklis, K. 2004. Graph clustering and minimum cut trees. *Internet Mathematics* **1**, 385–408.

Fortin, S. 1996. *The Graph Isomorphism Problem* Technical report, no. TR06-20, The University of Alberta.

Foggia, P., Genna, R. & Vento, M. 2001. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, 188–199.

Garey, M. R. & Johnson, D. S. 1979. *Computers and intractability – a guide to the theory of NP-completeness*. W.H. Freeman and Company.

Gärtner, T., Flach, P. & Wrobel, S. 2003. On graph kernels: hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Learning Theory (COLT'03)*, 129–143.

Getoor, L. & Diehl, C. 2005. Link mining: a survey. *ACM SIGKDD Explorations Newsletter* **7**(2), 3–12.

Gibbons, A. 1985. *Algorithmic Graph Theory*. Cambridge University Press.

Greco, G., Guzzo, A., Manco, G., Pontieri, L. & Saccá, D. 2005. Mining Constrained Graphs: the case of workflow systems, constraint based mining and inductive databases, Lecture Notes in Computer Science, 155–171. Springer.

Gudes, E., Shimony, S. E. & Vanetik, N. 2006. Discovering frequent graph patterns using disjoint paths. *IEEE Transaction on Knowledge and Data Engineering* **18**(11), 1441–1456.

Gutin, G. 2004. 5.3 Independent sets and cliques. In *Handbook of Graph Theory, Discrete Mathematics & Its Applications*, Gross, J. L. & Yellin, J. (eds). CRC Press, 389–402.

Han, J. & Kamber, M. 2006. *Data Mining Concepts and Techniques*, 2nd edition. Morgan Kaufmann.

Han, J., Pei, J. & Yin, Y. 2000. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1–12.

Han, J., Cheng, H., Xin, D. & Yan, X. 2007. Frequent pattern mining: current status and future directions. *Journal of Data Mining and Knowledge Discovery* **15**(1), 55–86.

Harary, F. & Ross, I. C. 1957. A procedure for clique detection using the group matrix. *Sociometry* **20**(3), 205–215.

Hein, J., Jiang, T., Wang, L. & Zhang, K. 1996. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics* **71**(1–3), 153–169.

Hido, S. & Kawano, H. 2005. AMIOT:induced ordered tree mining in tree-structured databases. In *Proceedings of the 5th IEEE International Conference on Data Mining*, 170–177.

Hopcroft, J. E. & Tarjan, R. E. 1972. Isomorphism of planar graphs. In *Complexity of Computer Computations*, Miller, R. E. & Thatcher, J. W. (eds). IBM Research Symposia Series Plenum Press, 131–152.

Hu, H., Yan, X., Huang, Y., Han, J. & Zhou, X. 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* **21**(1), 213–221.

Huan, J., Wang, W. & Prins, J. 2003. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 2003 International Conference on Data Mining*, 549–552.

Huan, J., Wang, W., Prins, J. & Yang, J. 2004. SPIN: mining maximal frequent subgraphs from graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 581–586.

Huang, X. & Lai, W. 2006. Clustering graphs for visualization via node similarities. *Visual Language and Computing* **17**, 225–253.

Inokuchi, A., Washio, T. & Motoda, H. 2000. An Apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 13–23.

Inokuchi, A., Washio, T. & Motoda, H. 2003. Complete mining of frequent patterns from graphs: mining graph data. *Journal of Machine Learning* **50**(3), 321–354.

Inokuchi, A., Washio, T., Nishimura, K. & Motoda, H. 2002. *A Fast Algorithm for Mining Frequent Connected Subgraphs*. Technical report RT0448, IBM Research, Tokyo Research Laboratory, Japan.

Jahn, K. & Kramer, S. 2005. Optimizing gSpan for molecular datasets. In *Proceedings of the 3rd International Workshop on Mining Graphs, Trees and Sequences*, 509–523.

Kashima, H., Tsuda, K. & Inokuchi, A. 2003. Marginalized kernels between labelled graphs. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, 321–328.

Ke, Y., Cheng, J. &, Ng, W. 2007. Correlated search in graph databases. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 390–399.

Ke, Y., Cheng, J. & Yu, J. 2009. Efficient discovery of frequent correlated subgraph pairs. In *Proceedings of the 9th IEEE International Conference on Data Mining*, 239–248.

Kelley, B., Sharan, R., Karp, R., Sittler, E., Root, D., Stockwell, B. & Tdeker, T. 2003. Conserved pathways within bacteria and yeast as revealed by Global Protein Network alignment. In *Proceedings of the National Academy of Science of the United States of America (PNAS'03)* **100**(20), 11394–11399.

Kleinberg, J. M. 1998. Authoritative sources in a hyper-linked environment. In *Proceedings of ACM-SIAM Symposium Discrete Algorithms*, 668–677.

Kosala, R. & Blockeel, H. 2000. Web mining research: a survey. *ACM SIGKDD Explorations Newsletter* **2**(1), 1–15.

Koyutürk, M., Grama, A. & Szpankowski, W. 2004. An efficient algorithm for detecting frequent subgraphs in biological networks. *Journal of Bioinformatics* **20**(1), 200–207.

Kudo, T., Maeda, E. & Matsumoto, Y. 2004. An application to boosting to graph classification. In *Proceedings of the 8th Annual Conference on Neural Information Processing Systems*, 729–736.

Kuramochi, M. & Karypis, G. 2001. Frequent subgraph discovery. In *Proceedings of the International Conference on Data Mining*, 313–320.

Kuramochi, M. & Karypis, G. 2002. Discovering frequent geometric subgraphs. In *Proceedings of the IEEE International Conference on Data Mining*, 258–265.

Kuramochi, M. & Karypis, G. 2004a. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1038–1051.

Kuramochi, M. & Karypis, G. 2004b. GREW-A scalable frequent subgraph discovery algorithm. In *Proceedings of the 4th IEEE International Conference on Data Mining*, 439–442.

Kuramochi, M. & Karypis, G. 2004c. Finding frequent patterns in a large sparse graph. In *Proceedings of the SIAM International Conference on Data Mining*, 345–356.

Kuramochi, M. & Karypis, G. 2005. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery* **11**(3), 243–271.

Liu, B. 2008. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer.

Liu, T. L. & Geiger, D. 1999. Approximate tree matching and shape similarity. In *Proceedings of 7th International Conference on Computer Vision*, 456–462.

Matula, D. W. 1978. Subtree isomorphism in $\mathcal{O}(n^{5/2})$ *Annals of Discrete Mathematics* **2**, 91–106.

McKay, B. D. 1981. Practical graph isomorphism. *Congressus Numerantium* **30**, 45–87.

Messmer, B. T. & Bunke, H. 1998. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence* **20**(5), 493–504.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D. & Alon, U. 2002. Network motifs: simple building blocks of complex networks. *Science* **298**(5594), 824–827.

Miyazaki, T. 1997. The complexity of McKay's canonical labelling algorithm, *Groups and Computation II, DIMACS Series Discrete Mathematics Theoretical Computer Science*, American Mathematical Society, **28**, 239–256.

Newman, M. E. J. 2004. Detecting community structure in networks. *The European Physical Journal B – Condensed Matter and Complex Systems* **38**(2), 321–330.

Nijssen, S. & Kok, J. N. 2003. Efficient discovery of frequent unordered trees. In *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences*, 55–64.

Nijssen, S. & Kok, J. N. 2004. A quickstart in frequent structure mining can make a difference. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 647–652.

Ozaki, T. & Ohkawa, T. 2008. Mining correlated subgraphs in graph databases. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'08)*, 272–283.

Paul, S. 1998. *Multicasting on the Internet and Its Applications*. Kluwer Academic Publishers.

Pei, J., Jiang, D. & Zhang, A. 2005. On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, USA, 228–238.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. & Hsu, M. C. 2001. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of 12th IEEE International Conference on Data Engineering (ICDE 01)*, Heidelberg, Germany, 215–224.

Preiss, B. R. 1998. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. Wiley.

Prüfer, H. 1918. Neuer beweis eines satzes über permutationen. *Archiv für Mathematik und Physik* **27**, 742–744.

Read, R. C. & Corneil, D. G. 1977. The graph isomorph disease. *Journal of Graph Theory* **1**, 339–363.

Rückert, U. & Kramer, S. 2004. Frequent free tree discovery in graph data. In *Proceedings of Special Track on Data Mining, ACM Symposium on Applied Computing*, 564–570.

Schmidt, D. C. & Druffel, L. E. 1976. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM* **23**(3), 433–445.

Schreiber, F. & Schwöbbermeyer, H. 2005. Frequency concepts and pattern detection for the analysis of motifs in networks. *Transactions on Computational Systems Biology* **3**, 89–104.

Shamir, R. & Tsur, D. 1999. Faster subtree isomorphism. *Journal of Algorithms* **33**(2), 267–280.

Shapiro, L. G. & Haralick, R. M. 1981. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **3**, 504–519.

Shasha, D., Wang, J. T. L. & Giugno, R. 2002. Algorithms and applications of tree and graph searching. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles on Database Systems*, 39–52.

Shasha, D., Wang, J. & Zhang, S. 2004. Unordered tree mining with applications to phylogeny. In *Proceedings of the 20th International Conference on Data Engineering (ICDE 04)*, 708–719.

Sharan, R., Suthram, S., Kelley, R., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R. & Ideker, T. 2005. Conserved patterns of protein interaction in multiple species. In *Proceedings of the National Academy of Science of the United States of America (PNAS'05)*, **102**(6), 1974–1979.

Tan, H., Dillon, T. S., Feng, L., Chang, E. & Hadzic, F. 2005. X3-Miner: mining patterns from XML database. In *Proceedings of the 6th International Data Mining*, 287–297.

Tan, H., Dillon, T. S., Hadzic, F., Chang, E. & Feng, L. 2006. IMB3-Miner: mining induced/embedded subtrees by constraining the level of embedding. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 450–461.

Tatikonda, S., Parthasarathy, S. & Kurc, T. 2006. Trips and Tides: new algorithms for tree mining. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, 455–464.

Termier, A., Rousset, M. C. & Sebag, M. 2002. Treefinder: a first step towards XML data mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, 450–457.

Thomas, L. T., Valluri, S. R. & Karlapalem, K. 2006. MARGIN: maximal frequent subgraph mining. In *Proceedings of the 6th International Conference on Data Mining (ICDM 06)*, Hong Kong, 1097–1101.

Ullmann, J. R. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM* **23**(1), 31–42.

Valiente, G. 2002. *Algorithms on Trees and Graphs*. Springer.

Vanetik, N. 2002. *Discovery of Frequent Patterns in Semi-structured Data*. Department of Computer Science, Ben Gurion University.

Vanetik, N., Gudes, E. & Shimony, S. E. 2002. Computing frequent graph patterns from semi-structured data. In *Proceedings of the 2nd International Conference on Data Mining*, 458–465.

Vanetik, N., Shimony, S. E. & Gudes, E. 2006. Support measures for graph data. *Journal of Data Mining and Knowledge Discovery* **13**(2), 243–260.

Wang, C., Hong, M., Pei, J., Zhou, H., Wang, W. & Shi, B. 2004a. Efficient pattern-growth methods for frequent tree pattern mining. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 441–451.

Wang, C., Wang, W., Pei, J., Zhu, Y. & Shi, B. 2004b. Scalable mining of large disk-based graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 316–325.

Wang, J., Zeng, Z. & Zhou, L. 2006. CLAN: an algorithm for mining closed cliques from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA, 797–802.

Washo, T. & Motoda, H. 2003. State of the art of graph-based data mining. *SIGKDD Explorations* **5**, 59–68.

West, D. B. 2000. *Introduction to Graph Theory*, 2nd edition. Prentice Hall.

Wörlein, M., Meinl, T., Fischer, I. & Philippsen, M. 2005. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM and Gaston. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, Portugal, 392–404.

Xin, D., Cheng, H., Yan, X. & Han, J. 2006. Extracting redundancy aware top K patterns. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 444–453.

Yan, X. & Han, J. W. 2002. gSpan: graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining*, 721–724.

Yan, X. & Han, J. 2003. CloseGraph: mining closed frequent graph patterns. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., USA, 286–295.

Yan, X., Yu, P. S. & Han, J. 2004. Graph Indexing: a frequent structure-based approach. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, Paris, France, 335–346.

Yan, X., Zhou, X. & Han, J. 2005a. Mining closed relational graphs with connectivity constraints. In *Proceeding of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 324–333.

Yan, X., Yu, P. S. & Han, J. 2005b. Sub-structure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 766–777.

Yan, X., Zhu, F., Han, J. & Yu, P. S. 2006. Searching substructures with superimposed distance. In *Proceedings of the 22nd International Conference on Data Engineering*, 88–97.

Yan, X., Cheng, H., Han, J. & Yu, P. S. 2008. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, Vancouver, Canada, 433–444.

Zaki, M. J. 2002. Efficiently Mining Frequent Trees in a Forest. In *Proceedings of the SIGKDD 2002*. ACM, 71–80.

Zaki, M. J. 2005a. Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae* **66**(1–2), 33–52.

Zaki, M. J. 2005b. Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering* **17**(8), 1021–1035.

Zaki, M. J. & Aggarwal, C. C. 2003. XRules: an effective structural classifier for XML data. In *Proceedings of the 2003 International Conference on Knowledge Discovery and Data Mining*, 316–325.

Zeng, Z., Wang, J., Zhou, L. & Karypis, G. 2006. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA, 797–802.

Zhang, S. & Wang, J. T. L. 2006. Mining frequent agreement subtrees in phylogenetic databases. In *Proceedings of the 6th SIAM International Conference on Data Mining*, 222–233.

Zhang, S. & Yang, J. 2008. RAM: Randomized Approximate Graph Mining. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, 187–203.

Zhao, P. & Yu, J. 2006. Fast frequent free tree mining in graph databases. In *Proceedings of the 6th IEEE International Conference on Data Mining Workshop*, 315–319.

Zhao, P. & Yu, J. 2007. Mining closed frequent free trees in graph databases. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, Thailand, 91–102.

Zhu, F., Yan, X., Han, J. & Yu, P. S. 2007. gPrune: a constraint pushing framework for graph pattern mining. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 388–400.