Chapter 1

Graph and network pattern mining

1.1 Introduction

During the last decade, ever growing databases have been constructed by both users and automated processes. In both cases, the structure of the databases is often complex, describing relations between many different objects. Therefore, data is usually represented using relational databases, graphs

or similar formalisms. Over the last decade, graph mining has emerged as a branch of data mining focusing on such graph-structured data.

One can distinguish two major types. First, in transactional graph mining one considers a collection of instances each represented by a separate isolated graph. One example is a database of molecules. Every molecule is represented with a graph, every node representing an atom and every edge representing a bond. Other examples include drawings, scene descriptions and local neighborhoods in a larger world.

The second setting can be called the single network setting. In this setting, all data is contained in one large graph. Instances of interest are vertices in this large graph. A typical example is the Internet. We can represent web pages with vertices and hyperlinks with edges. Other examples are social networks, forums and citation networks.

Graph mining research considers both algorithmic and statistical challenges. First, algorithmic challenges arise from the fact that many graph problems are intractable, e.g., subgraph isomorphism is NP-complete. Second, in a network, examples connected by edges are not (statistically) independent. E.g., web pages connected with hyperlinks are more likely to have similar topics. Therefore, the assumption of many machine learning algorithms that observations are independently and identically distributed (i.i.d.) does not hold. This poses challenges, e.g. with respect to the estimation of the statistical power of a training sample.

Next to data mining methods, also other approaches can provide useful insight in network structured data. One line of research which is increasingly being adopted by the graph mining community originated from statistical physics. It studies amongst others the assymptotic properties of networks where all participants behave in a similar way.

In this chapter, we survey graph mining methods. We focus on graph pattern mining, but also discuss a number of related topics such as generative models and the patterns emerging from them. The chapter is organized as follows. First, we review basic definitions and notations. Then, we discuss graph pattern mining in the transactional graph mining setting. In Section 1.4 we then consider mining in a single large network. Finally, we conclude and provide some pointers for further reading.

1.2 Basic concepts

We will first briefly review some basic terminology. A more in-depth introduction to graph theory and terminology can be found in [25].

Definition 1 (directed graph) *A directed graph is a tuple* (V, E, λ) *where* V *is a finite set of vertices (also called nodes),* $E \subseteq V \times V$ *is a set of arcs,*

and $\lambda : V \cup E \to \Sigma$ is a labeling function mapping every vertex and arc on a member of some label alphabet Σ. For a graph G, we will denote its vertex set with $V(G)$, its arc set with $E(G)$ and its labeling function with λ_G.

Many different types of graphs are considered in the literature depending on what is useful in a specific situation. For instance, unlabeled graphs are graphs where Σ is a singleton. They are often represented with pairs (V, E). In some settings, edges between nodes carry a continuous-valued weight, expressing the strength of a relationship.

Undirected graphs are graphs whose arcs, called edges, do not have a direction. Often edges of such graphs are defined as sets of two vertices, i.e. $E \subseteq \{e \subseteq V | \# e = 2\}$. For a directed graph, the underlying undirected graph is the graph obtained by removing the orientation of all arcs. A simple graph is a graph having at most one arc (edge) between any two vertices, and no loops (arcs connecting a node to itself). A multigraph may have several arcs between the same pair of vertices, these are then called parallel arcs. Hypergraphs are graphs where edges have one, two or more elements. Here too, there are variants with ordered and unordered edges. For simplicity and uniformity of explanation, we will use directed labeled graphs unless explicitly stated otherwise. The theory and algorithms presented here are very similar for undirected graphs, unlabeled graphs and hypergraphs.

As an example of how graphs are used to represent real-world data, consider a social network such as Facebook. We can represent users with nodes and friendship relations with edges. If we want to model more of Facebook, we could label the user nodes with label 'person', and add also 'group' nodes and 'event' nodes. Accordingly, we could add edges between persons and groups representing memberships and edges between persons and events indicating participation. Some relations are not binary. E.g. an invitation has a sender, a receiver and an event to which the receiver is invited by the sender. We could distinguish between different roles in several ways. First, the node representing the event to which the receiver is invited has the label 'event' which hence determines its role. To distinguish between sender and receiver, we could either use directed arcs from the sender to the invitation to the receiver, or we could label the edges with 'receiver' and 'sender', or we could insert additional nodes labeled 'receiver' and 'sender' and connect the sender, receiver and invitation with them. Many alternatives are equivalent and the specific choice often depends on what is easiest (e.g. for implementing algorithms) and clearest (to understand the semantics of the data).

Definition 2 (incident, adjacent, degree) *An edge $\{v, w\}$ is said to be incident with the vertices v and w. v and w themselves are said to be adjacent because they are connected by an edge. The degree of a vertex is the number of edges incident with it.*

An essential concept in pattern mining is pattern matching, i.e. the matching of a small graph, called the pattern, in a larger database graph. We will

first introduce subgraph isomorphism and homomorphism, the most popular matching operators.

Definition 3 (subgraph) *A graph H is a subgraph of a graph G if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and for all $x \in V(H) \cup E(H)$, $\lambda_H(x) = \lambda_G(x)$.*

Definition 4 (isomorphism) *Two graphs H and G are isomorphic, denoted $H \cong_i G$, iff there is a bijection $\pi : V(G) \to V(H)$ such that for all $x, y \in V(G)$, $(x, y) \in E(G) \Leftrightarrow (\pi(x), \pi(y)) \in E(H)$ and such that for all $x \in V(G) \cup E(G)$, $\lambda_G(x) = \lambda_H(\pi(x))$. Such a bijection π is called an isomorphism between the two graphs H and G. A graph H is subgraph isomorphic to a graph G, denoted $H \preceq_i G$, iff H is isomorphic to a subgraph of G.*

Definition 5 (homomorphism) *A graph H is homomorphic to a graph G, denoted $H \preceq_h G$ iff there is a (not necessarily injective) mapping, called homomorphism, $\pi : V(H) \to V(G)$ such that for all $x, y \in V(H)$, $(x, y) \in E(H)$ implies $(\pi(x), \pi(y)) \in E(G)$ and such that for all $x \in V(H) \cup E(H)$, $\lambda_H(x) = \lambda_G(\pi(x))$. Two graphs G and H are homomorphically equivalent, denoted $H \cong_h G$, iff G is homomorphic to H and H is homomorphic to G.*

As can already be expected from the notation, both relations induce a partial order and corresponding equivalence relation. Subgraph isomorphism and homomorphism is sometimes also called OI-subsumption (subsumption under object identity) and θ-subsumption (e.g. in the field of inductive logic programming [64]). Table 1.1 summarizes the terminology. Every subgraph isomorphism mapping is a homomorphism, but not every homomorphism is a subgraph isomorphism mapping. Figure 1.1 gives an example of (a) a subgraph isomorphism mapping and (b) a homomorphism.

Mapping	isomorphism	homomorphism
Equivalence	isomorphic ($\cong_i$)	equiv. under homomorphism ($\cong_h$)
Partial order	subgraph isomorphic ($\preceq_i$)	homomorphic ($\preceq_h$)
logic-based	OI-subsumption	θ-subsumption

TABLE 1.1: Isomorphism and homomorphism terminology

It is possible that a graph is subgraph isomorphic or homomorphic to another graph in several ways. The embedding and image concepts allow one to disambiguate between several such ways.

Definition 6 (embedding) *An embedding under isomorphism (resp. homomorphism) of a graph H in a graph G is a subgraph isomorphism mapping (resp. a homomorphism) from H to G. We denote the set of all embeddings of H in G with $Emb_i(H, G)$ (resp. $Emb_h(H, G)$).*

Definition 7 (image) *An image of a pattern H under some mapping π is*

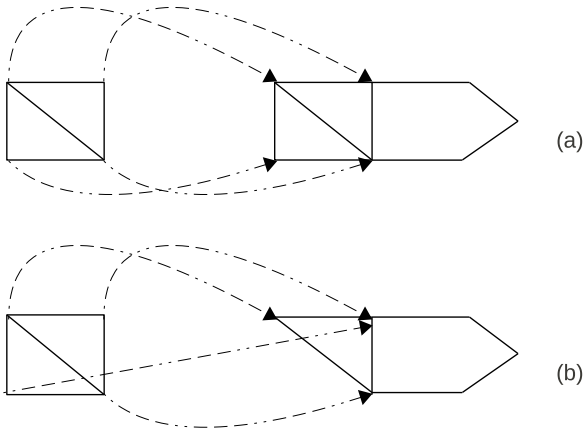

FIGURE 1.1: Isomorphism and homomorphism: (a) a subgraph isomorphism mapping (which is also a homomorphism), (b) a subgraph homomorphism mapping (but not isomorphism)

the graph formed by the set of images of all vertices and all edges under π*, i.e.* $\pi(H) = (\pi(V(H)), \pi(E(H)))$*. An image of* H *under isomorphism (resp. homomorphism) in some graph* G *is an image* $\pi(H)$ *of* H *under some subgraph isomorphism (resp. homomorphism)* π *from* H *to* G*.*

It is possible that several embeddings of H in G correspond to the same image of H in G (see e.g. Figure 1.2). The reverse does not hold: every embedding corresponds to only one image.

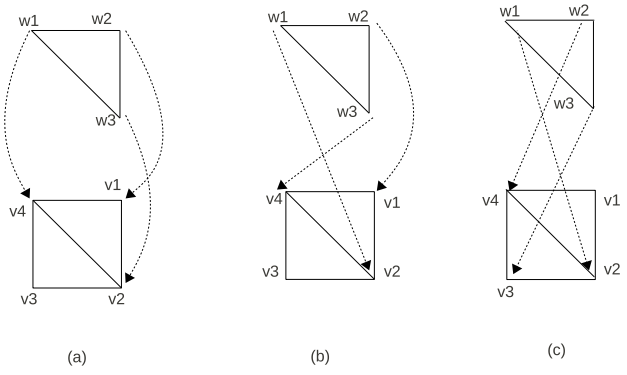

FIGURE 1.2: Embedding: (a) and (b) are two embeddings under isomorphism corresponding to the same image, c) shows a different image.

1.3 Transactional Graph Pattern mining

In the transactional graph mining setting, one considers a set of transactions, each represented with a graph. We don't assume dependencies between the transactions (in the terminology of learning theory, the transactions are independently and identically drawn from some distribution), so compared to the propositional machine learning setting and the itemset mining setting, the main additional challenge is the more complex structure of the graphs. This setting has many applications, and a good understanding of the transactional setting is a valuable prerequisite to study the more complex setting where all data is represented with a single network.

In this section, we will focus on graph pattern mining. Pattern mining is a basic task to discover patterns of interest. Once useful patterns have been generated, they can be used for a wide range of purposes. One application of pattern mining is association rule generation, where one wants to see whether certain patterns are correlated with certain target variables. Another popular application is feature generation, where every pattern can give raise to a feature which is 1 for the transactions where the pattern occurs and 0 for the other transactions). Also, if the set of generated patterns is of managable size, manual inspection can produce useful insight in the dataset.

1.3.1 The graph pattern mining problem

Depending on the situation, different settings of graph pattern mining may be considered. We therefore first identify the most important ingredients of a graph pattern mining problem.

Definition 8 (graph pattern mining setting) *A graph pattern mining setting is a triple* $(L_d, L_p, \leq)$ *where*

- L_d *is a class of graphs, called the database language,*
- L_p *is a subclass of* L_d*, called the pattern language*
- $\leq$ *is a partial order on* L_d*, called the matching operator*

A database for the graph pattern mining setting $(L_d, L_p, \leq)$*, or shortly a database for* L_d*, is a multiset of graphs of* L_d*. For two patterns* P_1 *and* P_2*, if* $P_1 \leq P_2$ *then we call* P_1 *more general than* P_2 *(since* P_1 *will occur in all graphs where* P_2 *occurs) and* P_2 *more specific than* P_1*.*

Figure 1.3 shows an example database of three undirected graphs over the alphabet $\{a, b, c\}$.

Definition 9 (graph pattern mining problem) *Let* $(L_d, L_p, \leq)$ *be a graph*

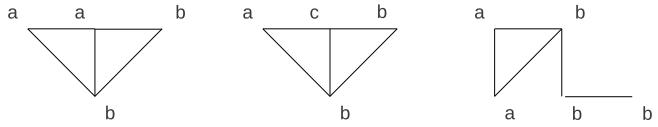

FIGURE 1.3: An example database of three undirected graphs over the alphabet $\{a, b, c\}$

pattern mining setting. An interestingness predicate ι for $(L_d, L_p, \leq)$ is a predicate mapping every pair (D, P), where D is a database for L_d and P is a pattern from L_p, to either true *or* false*.*

The problem of ι-interesting graph pattern mining for $(L_d, L_p, \leq)$ is the problem where the input is an input database D for $(L_d, L_p, \leq)$ and the task is to list all elements $P \in L_p$ for which $\iota(D, P)$ holds.

The most popular interestingness predicate is a minimal frequency constraint:

Definition 10 (pattern frequency) *Let $(L_d, L_p, \leq)$ be a graph pattern mining setting and D be a database for it. Let $P \in L_p$ be a pattern. The frequency of P in D, denoted $freq(D, P)$ is defined as*

$$freq(D, P) = |\{T \in D \mid P \leq T\}| \tag{1.1}$$

P is frequent in D w.r.t. some frequency threshold t if $freq(D, P) \geq t$

Consider e.g. the graph pattern depicted in Figure 1.4. The frequency of this graph pattern in the database of Figure 1.3 is 2.

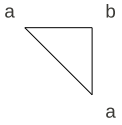

FIGURE 1.4: An example graph pattern.

Definition 11 (frequent graph pattern mining problem) *Let $(L_d, L_p, \leq)$ be a graph pattern mining setting. The problem of frequent graph pattern mining for $(L_d, L_p, \leq)$ is the problem where the input is a database D for $(L_d, L_p, \leq)$ and a minimal frequency threshold t, and the task is to list all elements $P \in L_p$ for which $freq(D, P) \geq t$ holds.*

Other interestingness predicates can be considered, e.g. [13] studies the constraint that patterns should have a minimal correlation with a given target

attribute. The pattern mining problem is also related to combinatorial enumeration problems such as the enumeration of all molecules satisfying some specific properties [37].

A useful property for interestingness predicates is anti-monotonicity. Most of standard pattern mining algorithms rely on this property to prune their search.

Definition 12 (anti-monotone predicates) *Let $(L_d, L_p, \leq)$ be a graph pattern mining setting and let ι be an interestingness predicate for $(L_d, L_p, \leq)$. ι is said to be anti-monotone iff for all databases D for L_d and for all graphs G and H in L_p , $G \leq H$ and $\iota(D, H)$ implies $\iota(D, G)$.*

Note that while the term anti-monotone is more common in the field of pattern mining, different conventions exist. For example, in the field of graph theory the terms "monotone graph class" and "monotone predicate" are more commonly used.

1.3.2 Basic pattern mining techniques

The task of pattern mining is an enumeration task in the sense that there are several solutions to a pattern mining problem and the task is to list each of them.

Most approaches perform a *search*, starting at some most general element, and then incrementally specializing it. In such a general-to-specific search, one can exploit the anti-monotonicity property. E.g., the frequency of a more specific pattern will never be larger than the frequency of a more general pattern. Accordingly, if a pattern is not frequent, it is not needed to investigate any pattern which is more specific.

In order to perform a search, it is necessary to get from one candidate solution to another one. For this, an extension operator (also called refinement operator) is commonly used.

Definition 13 (minimal element) *Let $L, \leq$ be an ordered set. An element x of L is minimal iff for all $y \in L$, $y \leq x$ implies $x \leq y$.*

Definition 14 (extension operator) *Let $L_p, \leq$ be a partially ordered set. An extension operator ρ is a mapping $\rho : L_p \to L_p$ such that $\forall G \in L_p : \rho(G) > G$. We denote $\rho^1 = \rho$, $\rho^{i+1} = \rho \circ \rho^i$ and $\rho^*(x) = \cup_{i \in \mathbb{N}} \rho^i(x)$. We call ρ a complete extension operator iff $L_p, \leq$ has a finite set of minimal elements $P_\perp$ and $L_p = \rho^*(P_\perp)$, i.e. by applying ρ recursively to $P_\perp$ all patterns can be generated in a finite number of steps.*

Pseudocode for a generic frequent graph pattern mining algorithm is shown in Algorithm 1. As can be seen, to perform a search in practice, the generic algorithm needs a starting point (the minimal elements $P_\perp$ of $(L_p, \leq)$) and a complete extension operator ρ. These, together with the frequency counting operator $freq(\cdot, \cdot)$ are important topics in graph pattern mining research.

Algorithm 1 is a breadth-first algorithm. We will discuss alternatives (mainly depth-first) and advantages of each search strategy later in this chapter.

Algorithm 1 Algorithm MineFrequentPatterns($(L_d, L_p, \leq), P_\perp, \rho, D, t$)

```
1: Require. a graph pattern mining setting (L_d, L_p, ≤), the set P_⊥ of all minimal elements of L_p under ≤, a complete extension operator ρ, a database D and a frequency threshold t.
2: Ensure. all frequent patterns
3: S_0 ← {H ∈ P_⊥ | freq(D, H) ≥ t}
4: k ← 0
5: while S_k ≠ {} do
6:     k ← k + 1
7:     S_k ← {}
8:     C_k ← {}
9:     for all G ∈ S_{k−1} do
10:        print(G)
11:        for all H ∈ ρ(G) do
12:            if H ∉ C_k then
13:                C_k ← C_k ∪ {H}
14:                if freq(D, H) ≥ t then
15:                    S_k ← S_k ∪ {H}
16:                end if
17:            end if
18:        end for
19:    end for
20: end while
21: return ∪_k S_k
```

Consider again the database in Figure 1.3. Suppose we mine all patterns under subgraph isomorphism with frequency at least 2. Figure 1.5 shows the resulting patterns.

In general, many approaches can be seen as *generate-and-test* approaches, where first the task is to *orderly enumerate* all candidate patterns (in Algorithm 1 the patterns which are stored in C_k), and the second step is to check for each enumerated pattern whether it is *frequent* (in Algorithm 1, these patterns are added to S_k and later printed). Of course, both steps can be interleaved as in Algorithm 1. For more complex data structures, orderly enumeration may become a non-trivial problem. It is needed to enumerate all patterns, but enumerating the same pattern several times may be inefficient or produce redundant output. Usually, one tries to either define a canonical form of the patterns, or a canonical form of generating a pattern. We will present an example later when discussing concrete graph mining systems.

One can distinguish between two major types of pattern mining algorithms: breadth-first search algorithms and depth-first search algorithms. The *breadth-*

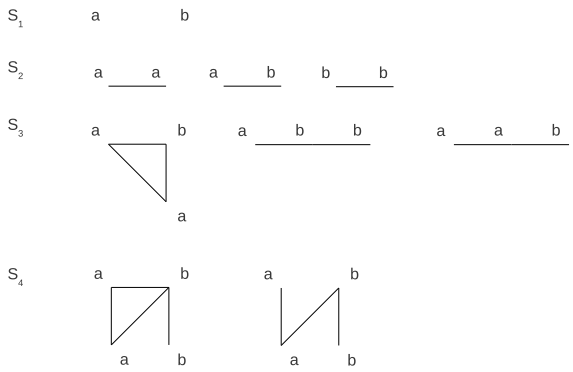

FIGURE 1.5: 2-frequent patterns (i.e. patterns occurring with frequency at least 2) in the database of Figure 1.3.

first algorithms define a size function on the pattern language and generate the patterns in order of size. For example, the size of a graph could be defined as the number of edges or as the number of vertices. When a graph is generated, then one knows that all its subgraphs (which have a smaller size) have already been considered. The breadth-first algorithms therefore have the advantage that they can exploit the anti-monotonicity property better, since when a pattern is generated the frequency of all patterns which are more general is already counted. Therefore, there is a maximal opportunity to find out when a pattern which is more general than the one under consideration, is infrequent.

On the other hand, this also means that the breadth-first algorithms store a lot of data in quickly accessible storage space. The *depth-first* approaches have the advantage that when a branch of the search is finished, one can discard all data structures concerning that branch. As a result, only a minimum of memory is required to store patterns, and available memory can be used for other optimizations. For example, when one stores for some node in the search tree all transactions matched by the associated pattern, then when counting the frequency of the more specific children one has only to check these transactions. Indeed, transactions not matched by a parent node will not be matched by its children. One example of a depth-first algorithm is FP-growth [38].

1.3.3 Graph mining settings

There are several significant factors influencing the characteristics of a pattern mining problem. A first factor is the *graph class*. One can either consider the fully general graph mining problem, allowing any graph as transaction or pattern, or one can restrict graphs to some subclass such as star graphs, paths or trees. Several settings considering restricted graph classes have been

considered in the literature or are equivalent to well-studied pattern mining settings:

- Star graphs are graphs consisting of a central vertex and a number of leaf vertices connected to this central one. When considering star graphs where leaf vertices are labeled with items from some set I, mining frequent star graph patterns of at least 2 vertices is equivalent to itemset mining [1] where the set of items is I and the transactions are the sets of leaf labels of individual star graphs. Even though problems only involving itemset mining can be handled more efficiently with a special-purpose itemset mining system, star graphs may be used to represent itemsets in problems involving also more complex graphs. In that case, star graphs make all data uniform so that a graph mining system can be used.

- A (directed) path is a graph P with $V(P) = \{v_1, v_2, \ldots, _v n\}$ and $E = \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\}$. Labeled paths can be seen as sequences. Finding frequent subgraph isomorphic patterns is equivalent to finding frequent subsequences. Many subsequence mining algorithms have been proposed in the literature, amongst others [2], [67] and [4].

- An undirected cycle is a graph C with $V(C) = \{v_1, v_2, \ldots, v_n\}$ and $E = \{\{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$. A tree is a graph that does not contain a cycle as a subgraph. In the data mining literature, several types of trees are considered. Free trees are undirected trees. Rooted trees (also called rooted unordered trees) are trees where one vertex acts as the root and all edges are directed away from this root. For a vertex v of a rooted tree T, the vertices w for which $(v, w) \in E(T)$ are called children of v, and v is called their parent. Rooted ordered trees are rooted trees where for each vertex a total order is specified on its children. An attribute tree is a rooted tree for which for each vertex all its children have a distinct label. A huge amount of tree mining algorithms have been proposed in the literature [18]. Common applications include mining parse trees (in programming languages or natural language processing), XML and many other tree-structured file formats.

- Some approaches consider specifically molecular graphs or classes of a subset of the molecular graphs. Molecular graphs do not have a very formal definition, the main idea is that all molecules occurring in practice are considered. Atoms are represented with vertices and bonds with edges. Naturally, there are some restrictions to these graphs, for example due to the limited valency of atoms, a constraint which translates to a limited degree of vertices.

Usually, one chooses a graph class which is sufficiently expressive to represent all data in the application, but which is otherwise as restricted as possible

in order to be able to exploit the structural properties of the graphs. For example, texts are normally represented with strings and HTML web pages with rooted trees.

A second parameter is the *graph type*. One can use directed and/or undirected edges, ordinary graphs or hypergraphs, possibly allow loops (edges with the same vertex at both endpoints), and possibly allow parallel graphs (where the edge set is a multiset, making it possible to have several edges between the same pair of vertices). Often, the choice here will depend on what is most natural for the application at hand, and what types of graphs the available algorithms can process. There exist transformation methods to transform certain types of graphs into other types of graphs without loss of information and often without significant loss in efficiency. Consider for example a graph representing relationships between humans. It is possible that two persons (vertices) p and q are both friends and colleagues. One can represent this in a parallel graph with drawing an edge labeled "friend" between p and q, and by also drawing an edge labeled "colleague" between p and q. Alternatively, one can avoid the complexity of parallel graphs by introducing a new vertex for every relationship. One would then add vertices x (labeled "friend") and y (labeled "colleague") and connect them to both p and q.

A final important parameter of a graph mining problem is the *matching operator*. Depending on the application at hand, some matching operator may be preferred to another. The most common graph matching operator is the subgraph isomorphism operator. The homomorphism operator has been studied extensively in the field of Inductive Logic Programming [63], where it is known as the θ-subsumption operator. Other graph matching operators exist, such as the subgraph homeomorphism operator [56], but these are less frequently used in the graph mining literature. So when to use isomorphism and when to use homomorphism? It often happens that vertices represent objects in the application and edges represent relations between them. When using subgraph isomorphism, one wants that different objects (vertices) of the pattern to correspond to different objects (vertices) in the matching transactions. For example, when considering molecules, saying that a molecule contains a chain of 3 carbon atoms naturally means that there are three different carbon atoms in the object. On the other hand, when using homomorphism, objects can play several roles. Consider for instance graphs representing relationships between people. A pattern could express that some persons p_1 and p_2 are brothers, that p_1 has a boss b_1 and that p_2 has a boss b_2 (by drawing suitable colored edges between p_1, b_1, p_2 and b_2). However, if the brothers would work in the same company they may have the same boss. In that case, one would like the pattern to match the transaction (a triangle between the boss and the two brothers). In such a situation one may prefer homomorphism.

1.3.4 Complexity

Graph mining may be very demanding for storage and computation resources. In typical data mining experiments, one attempts to mine thousands to millions of patterns, while the size of databases is constantly increasing. It is therefore important to have a better understanding of the factors determining the complexity of pattern mining, both from a theoretic and from a practical point of view. In this section, we will discuss techniques to analyze the complexity of a pattern mining algorithm and techniques to optimize it w.r.t. a naive algorithm.

1.3.4.1 Enumeration complexity

We start with a discussion on complexity analysis. While such analysis is not necessarily predictive for the behavior of an algorithm in practice, it can provide insight and hints at improvements. For example, one may want to exploit the structure of data when it is known that this allows for a lower time or space complexity, or one may decide to restrict the representation in such a way that an efficient technique can be applied. In some cases, such decisions can make the difference between a polynomial time assymptotic complexity or solving a $\#P$-hard problem, while in practice the running time can be improved by several orders of magnitude.

The number of solutions (frequent patterns) to a frequent pattern mining problem may be huge. Even for the itemset mining problem, in worst case there are $2^{|I|}$ frequent itemsets, where $|I|$ is the number of items in the database. In the general case for graphs, we get roughly 2^{n*n} possible patterns with n the number of vertices, not even taking into account multiple label values. As a consequence, one can not hope to obtain a polynomial time algorithm. To perform a more refined analysis, the following complexity classes are usually considered in the literature [48]. They are based on the idea that a listing algorithm (here a pattern mining algorithm) lists its solutions (frequent patterns) one by one, and that one can study the delay between outputting two consecutive solutions. At the start, the algorithm gets a certain amount of time on the clock, for every new solution it also gets "payed" a certain amount of new time on the clock, and its goal is to never run out of time.

Definition 15 (listing complexity) *Let S be a set of cardinality N. Then its elements, say $s_1, \dots s_N$, are listed with polynomial delay if the time until printing s_1, the time between printing s_i and s_{i+1} for every $i = 1 \dots N$, and the termination time after printing s_N is bounded by a polynomial of the size of the input, in incremental polynomial time if the time between printing s_i and s_{i+1} for every $i = 1 \dots N$ (resp. the termination time after printing s_N) is bounded by a polynomial of the combined sizes of the input and $s_1, \dots, s_i$ (resp. $s_1, \dots, s_N$), in output polynomial time (or polynomial total time) if S is printed in a time bounded by the combined sizes of the input and the entire set S.*

Clearly, polynomial delay implies incremental polynomial time, which, in turn, implies output polynomial time. We also note that, in contrast to incremental polynomial time, an output polynomial time algorithm may have in the worst case a delay time exponential in the size of the input before printing the i-th element for any $i \geq 1$.

1.3.4.2 Complexity results

For several settings, one has investigated whether efficiency with respect to one of the above classes could be obtained. As a simple example, in the case of mining general graph patterns under subgraph isomorphism, one can use the following simple argument to show that (unless P=NP) no algorithm exists to list all frequent subgraph patterns in output polynomial time [40]. Consider a database with two transactions. The first one is a cycle with length n, and the second transaction is an arbitrary graph on n vertices. Let the minimal frequency threshold be 2. There are at most $n + 1$ frequent patterns (the paths of length 0 to $n - 1$ and the cycle of length n). Suppose that there is an algorithm which lists all frequent subgraph patterns in output-polynomial time. Then, given that we can bound the output size by a polynomial in the input size $n + 1$, we could bound the total running time by a polynomial in the input size. This would mean that we could decide whether there is a cycle of length n in the arbitrary graph of size n in polynomial time. However, this problem, known as the Hamiltonian cycle problem, is known to be NP-complete. Therefore, unless P=NP we reach a contradiction.

On the other hand, if the pattern matching operator $\leq$ can be executed in polynomial time, Algorithm 1 will run with polynomial delay if the candidate generation can be performed efficiently (i.e. ρ and the test in line 10 of Algorithm 1 takes only polynomial time). This has been shown to be the case for itemsets [2], paths and trees. For larger classes of graphs, the situation depends on some other factors.

Consider e.g. the class of bounded treewidth graphs. Treewidth is a graph property measuring how much a graph resembles a tree.

Definition 16 (treewidth) *A tree decomposition of a graph G is a pair (T, X) where T is a rooted tree and $X = \{X_x\}_{x \in V(T)}$ is a family of subsets of $V(G)$ satisfying (i) $\cup_{x \in V(T)} X_x = V(G)$, (ii) for every $\{u, v\} \in E(G)$, there is a $x \in V(T)$ such that $u, v \in X_x$ and (iii) $X_x \cap X_y \subseteq X_z$ for every $x, y, z \in V(T)$ such that z in on the path between x and y. The treewidth of (T, X) is $max_{x \in V(T)} |X_x| - 1$ and the treewidth of G is the minimal treewidth over all tree decompositions of G.*

Let w be a constant. If G has treewidth at most w and H is connected and has bounded degree, then one can decide whether H is a subgraph of G in polynomial time (the constant w appearing in the exponent). However, in general subgraph isomorphism between connected graphs of treewidth at most w (for w at least 2) is NP-complete [59]. Despite this negative result, it

can be shown that frequent connected subgraphs of treewidth at most w can be listed in incremental polynomial time [42]. The homomorphism matching operator has a lower time complexity. For (not necessarily connected) graphs of bounded treewidth, not necessarily of bounded degree, homomorphism can be decided in polynomial time and hence mining is possible with polynomial delay.

The graph class and the matching operator are not the only parameters that have an influence on the complexity. The search strategy may imply or prohibit that some computations can be shared. For example, the incremental polynomial time result for graphs [42] uses a breadth-first approach. For a depth-first approach, incremental polynomial time can not be reached.

1.3.4.3 Optimization techniques

There are a large number of common strategies to improve the computational complexity of graph pattern mining. Some techniques affect the asymptotic behavior, others only affect the practical running time. In our discussion, we will focus on the frequency counting since for sufficiently large databases, this forms the dominating cost.

Restricting the graph classes

A first technique consists in restricting the class of the data graphs or of the pattern graphs. Such restriction may be possible due to several reasons. First, in many applications, one may have prior knowledge on the structure of the data. This may include hard rules (e.g. molecules have bounded valency, mRNA molecules can be represented as outerplanar graphs [41], ...) or general knowledge to which there are exceptions (for example. 96% of the molecules are outerplanar graphs [41], traffic networks are almost planar graphs).

Second, one may feel that only patterns from a particular subclass are sufficiently valuable, because other patterns are too complex to interprete afterwards. For example, domain experts will often prefer free patterns rather than closed patterns because free patterns are smaller and hence easier to understand (see below for a definition of 'free pattern').

In each of these cases, one can make assumptions on the graph class of the data and the patterns. This may allow one to use specialized algorithms which exploit the known structure and are therefore more efficient.

Storing information to avoid recomputation

Typically, several computations need to be repeated. One strategy which has been explored is to store some of these results to avoid recomputation. However, since memory is limited, one usually has to make an intelligent choice regarding which information to store and which information to recompute. Information which is stored often includes the following:

- the identifiers of transactions matching smaller patterns, since only those have to be considered when checking superpatterns;
- embeddings of smaller patterns, since to obtain an embedding of a superpattern one can start from an embedding of a smaller pattern;

- association rules already discovered, since these allow one to derive in certain cases the frequency of patterns from the frequency of smaller patterns[24],

Heuristics
Solving hard problems may be unavoidable, and in that case intelligent search can help. Consider for example the problem of pattern matching. Subgraph isomorphism checking is the most expensive part of most large-scale graph mining processes. Even though it is NP-complete in general, different heuristic strategies may vary strongly in efficiency. The most common strategy is to map the pattern in the database graph, node by node, backtracking when no solution can be found. A classical search heuristic is to first map the node for which the number of available alternatives is the smallest [76].

Prioritizing the more important patterns
Even though it does not influence the total cost, delivering the most important patterns first may be more desirable than treating all patterns equally. This is even more true in the case where the number of patterns is so large that the mining process is expected to be terminated before outputting all answers. Also, when the time needed to find patterns diverges strongly, finding the easier-to-find patterns first may be called for. One example is described in [65] where paths are mined first, then trees and only at the end, cyclic graphs.

Hierarchical approaches
In some cases, sets of nodes can be grouped together into larger entities, and working with such groups is more efficient than working with individual nodes. For example, in molecules, atoms are often grouped together in functional groups such as rings and chains [41], [11], [23]. Also, in traffic networks, streets may be grouped into districts or cities, an approach which is typically used in routing applications [33].

1.3.5 Condensed representations

When performing pattern mining, one is usually only interested in the most valuable patterns. E.g., patterns may be processed afterwards, either by a human or an algorithm performing a next step of data mining. In both cases, it is undesirable to have a too large set of mined patterns. Therefore, a huge amount of literature has considered the question of how to remove from the set of all patterns those which are redundant according to certain criteria. In this section, we will review two directions of research. First, we will consider free and closed patterns as a way to represent all information using fewer patterns. Next, we will discuss different criteria which can be used to select the most useful patterns, thereby discarding part of the information.

1.3.5.1 Free and closed patterns

Recall that the language L_p of patterns is equipped with a partial order $\leq$, and that the frequency measure is anti-monotonic w.r.t. this order. Consider

two patterns P and Q such that $P < Q$, In some cases, Q may (informally stated) occur everywhere where P occurs. For example, the pattern Q in Figure 1.6 matches in all graphs of the database in Figure 1.3 where pattern P from Figure 1.6 matches. In such case, it is not necessary to output both patterns with their frequency, as they are identical. We will call the pattern P from Figure 1.6 a frequency-free (f-free for short) pattern, as the pattern description does not contain redundant information. Pattern Q in Figure 1.6 is not free, as removal of an edge gives a pattern with the same frequentcy. On the other hand, pattern Q in Figure 1.6 is a maximal pattern with frequency 2. We will therefore call Q a f-closed pattern. More formally:

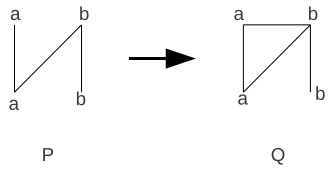

FIGURE 1.6: In the database of Figure 1.3, the association rule $P \rightarrow Q$ holds, i.e. whenever P is a subgraph, also Q is a subgraph. Therefore, Q is a closed pattern but P is not.

Definition 17 (f-free pattern) *Let D be a database and let $P \in L_p$ be a pattern. P is a f-free w.r.t. D iff there is no pattern $Q < P$ such that $freq(D, Q) = freq(D, P)$.*

Definition 18 (f-closed pattern) *Let D be a database and let $P \in L_p$ be a pattern. P is a f-closed w.r.t. D if there is no pattern $Q > P$ such that $freq(D, Q) = freq(D, P)$.*

The notion of closed pattern is most interesting when it is related to a so called closure operator. An operator $cl : L_p \rightarrow L_p$ is a closure operator if it satisfies the following properties:

- idempotency: $\forall P \in L_p : cl(cl(P)) = cl(P)$
- extensivity: $\forall P \in L_p : cl(P) \geq P$
- monotonicity: $\forall P, Q \in L_p : P \leq Q \Rightarrow cl(P) \leq cl(Q)$

A pattern P is called cl-closed if $cl(P) = P$.

The importance of closure operators becomes clear when one considers the complexity of mining all closed patterns. In particular, for a closure operator cl, under reasonably weak assumptions, it is possible to mine all cl-closed patterns in output polynomial time.

For itemsets and a number of other settings such as attribute trees [5], there is a closure operator such that the corresponding closed patterns coincide

with the definition of f-closed patterns above. For more complex graph classes however, the situation is more complicated.

[32] showed that when every pair of patterns in L_p has a unique least upper bound, a closure operator exists and one can mine all f-closed patterns efficiently. This holds in particular for graph patterns under homomorphism. The least upper bound of two graph patterns under homomorphism is their product graph. In fact, this least upper bound is well-known in the field of Inductive Logic Programming as the least general generalization of two logical conjunctions [68]. Unfortunately, the size (as well as the treewidth) of the least upper bound of a set of graph patterns under homomorphism is in general not bounded by a polynomial in their total size. As a consequence, computing the least upper bounds and checking their frequency prohibits efficient mining of closed graph patterns under homomorphism.

Nevertheless, the relation between homomorphism and logical subsumption is an important one which allows for using very expressive languages. For example, [24] describes how to combine mining of closed logical conjunctions with a background theory which may represent prior domain knowledge of the user.

For a closure operator cl, one can also define cl-free patterns (these are patterns P such that for all $Q < P$, $cl(Q) \neq cl(P)$). However, even though the set of free patterns is a subset of the set of all patterns, the number of closed patterns is smaller than the number of free patterns. This may explain why closed patterns got more attention in the literature on condensed representations.

When a notion of closed patterns is used which does not correspond to some closure operator, the so-called "early termination" technique, the usual pruning strategy for closed pattern mining algorithms, can not be used. In the general case, this may mean that one has to mine all frequent patterns, and then to filter out those which are closed. Given that the number of frequent patterns may be exponential in the number of closed patterns, one can not expect such an approach to run in output polynomial time.

If it is not needed that the solution set is a lossless compression of the set of all patterns with their frequency, then stronger compression is possible in the sense that smaller sets of patterns can be generated from which one can not reconstruct the frequency of all patterns. One possible approach is to mine δ-closed patterns. A pattern P is δ-closed if each of its superpatterns has a frequency which is smaller than $freq(P) - \delta$. The notion of δ-closed patterns can also be extended to graphs [24].

1.3.5.2 Selection of informative patterns

Closed patterns are a powerful concept to reduce the number of patterns without losing information. However, often the number of patterns is still huge. Moreover, in a number of settings there are no known strategies to skip

the non-closed patterns (achieving output-polynomial time), and one has to revert anyway to a generate-and-test strategy.

Therefore, another research direction is to select patterns or association rules on the basis of some useful quality criteria. These criteria include:

- correlation of the pattern or association rules with some target attribute [83], [13]
- association rule quality (e.g., lift, confidence, leverage),
- additional information provided by the pattern and its frequency compared to a set of already selected patterns.

Typically, one can start from the set of all frequent patterns and then filter them using the given selection criteria. In some cases, one can perform some pruning in the search space. For example, [83] describes a method to prune when searching for patterns correlating with the target attribute.

1.3.6 Transactional graph mining systems

One of the first graph mining systems was AGM [45], mining induced subgraphs, and its variants such as AcGM. This system used a matrix-based canonical form, and introduced operations to join the canonical form of smaller patterns into the canonical form of a larger pattern. In particular, patterns are represented with their adjacency matrices. Given a graph G with an ordered set of vertices $V(G) = \{v_1, v_2, \ldots, v_n\}$, the adjacency matrix A of G is an $n*n$ matrix with $A_{i,j} = 1$ iff $(v_i, v_j) \in E(G)$ and $A_{i,j} = 0$ otherwise. For example, consider Figure 1.7. On the right-hand side one can see the adjacency matrix for the pattern on the left-hand side (with the vertices ordered 1,2,3,4,5). AGM defines the code for an adjacency matrix A as follows:

$$code(G) = A_{1,1}A_{2,1}A_{2,2}A_{3,1}A_{3,2}A_{3,3}\ldots A_{n,n-1}A_{n,n}$$

For example, in Figure 1.7, the code is shown below the adjacency matrix of the pattern. The canonical code of a graph is the minimal code which can be obtained from an adjacency matrix of that graph over all orderings of its vertices. The interesting property of a canonical form is that whenever two graphs are isomorphic, they have the same canonical code. This property can be used to ensure that every pattern is generated exactly once. The code in Figure 1.7 is canonical.

AGM is a breadth-first pattern miner. A new pattern of size n is generated from two patterns of size $n-1$ sharing a common $(n-2)*(n-2)$ block in the top left of their canonical adjacency matrix. In particular, if

$$A = \begin{bmatrix} X & y \\ y^T & z \end{bmatrix}$$

and

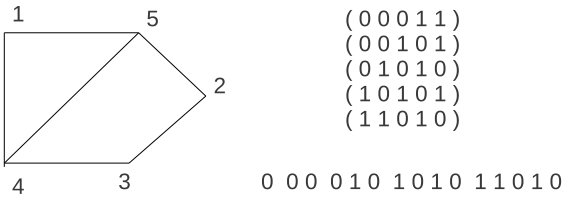

FIGURE 1.7: A pattern and its canonical adjacency matrix and code.

$$B = \begin{bmatrix} X & u \\ u^T & v \end{bmatrix}$$

then the newly generated pattern is of the form

$$C = \begin{bmatrix} X & y & u \\ y^T & z & w \\ u^T & w & v \end{bmatrix}$$

where w is either 0 or 1. Several subsequent algorithms also exploited this idea of only generating new patterns from the combination of two smaller patterns which differ only at one vertex. This limits the number of new candidate patterns to consider. AcGM [44] extends AGM by allowing for both induced subgraph isomorphism and normal subgraph isomorphism, and by considering hierarchies of labels.

gSpan [82] is one of the most popular graph mining systems. It uses a depth-first algorithm. This system is also based on an orderly generation of patterns using a canonical form, called depth-first search (DFS) code. New patterns are generated by extending a single smaller pattern, in such a way that every pattern is generated from exactly one parent pattern. The depth-first approach has been shown to have advantages on not too large datasets as it allows for storing information on embeddings of the parent pattern in the database graphs, which can then be exploited when counting the frequency of larger patterns.

The Gaston system [65] has been shown to be one of the most efficient graph pattern mining systems. One of its most important features is that it first mines all paths and trees before mining cyclic structures. The main advantage of this approach is that in a range of practical datasets, such as molecule datasets, most frequent patterns are acyclic. Gaston also employs a depth-first search algorithm.

Some approaches add more than one vertex at a time, making larger steps in the search space. Examples are the MoSS system [39] and the FOG system [41], both of which treat cyclic fragments separately from linear fragments, an approach which turns out to be useful in chemo-informatics applications [72].

Some researchers consider special purpose graph mining systems. For ex-

ample, [55] describes gFSG, aimining at mining geometric patterns, i.e. patterns where vertices have spatial coordinates.

1.4 Single network mining

Up to now, we considered the transactional setting of graph mining, where a database consists of separate, independent transaction graphs. In practice, not all graph data can be structured in this way. Consider for instance the Internet, news forums, traffic networks, protein interaction networks and citation networks. In all these cases, the data is made up by one large graph or network, where instances are the nodes. E.g. in a social network, the users may be represented by nodes and their interactions with edges. One may be interested both in global properties of the networks or in properties of individual nodes. We will discuss both of them in the following sections.

Global properties include parameters such as the average distance between nodes, the connectivity, the occurrence of clusters of highly connected nodes, the degree distribution, etc. The field of complex systems, originating from statistical physics, has produced a large body of literature on this topic.

When considering the properties of individual nodes, pattern mining questions similar to those in the transactional setting can be considered. However, there are several additional challenges compared to that simpler setting. First, since the neighborhoods of the nodes overlap heavily, they are not independent. In order to perform correct statistics, it is important that examples are independent and hence generalizing over nodes of a network is a nontrivial task. A second problem is that in the large network case, the graph at hand is usually much larger than the graphs considered in the transactional setting, and hence more care is needed w.r.t. computational complexity.

1.4.1 Network models

While in the transactional setting a lot of data consists of descriptions of static objects, many large networks are evolving. Even more, often one assumes that the evolution in all parts of the network is somehow similar. E.g., consider a social network. Motivations to become friends, such as recommendation of other friends and common interests, are relevant for each of the users in the network, and hence to some extent two users with similar properties in a similar context may act similarly.

Network models, of which we will discuss some simple examples in Section 1.4.1.2, have been studied with methods originating from statistical physics (where particles too are assumed to behave all similarly). Interestingly, when a fixed local process is repeated a large number of times, certain patterns and

structures arise in the network. In Section 1.4.1.1 we will discuss a number of properties commonly used to measure the characteristics of networks.

When no model of the network is available, it is possible to learn a model from data. Here, one can consider both predictions of future evolutions in the dynamic network and the learning of statistical patterns in a static snapshot of the network. In both cases, pattern mining, i.e. the discovery of interesting local patterns, is a first step. We will discuss pattern matching and pattern mining in later sections.

1.4.1.1 Network property measures

One of the most popular statistics for networks is the diameter. It is most commonly defined as the longest shortest path in the graph. This definition is problematic for real-world networks, as they often contain several components, i.e. for each vertex of the graph, there exist vertices that are not reachable via the edges. In such situations one usually considers the giant connected component of the graph, i.e. the connected component having the biggest number of nodes. In many applications, the diameter of the network has been shown to be surprisingly small. From a pattern mining point of view, the diameter is an indication for the diameters of 'local patterns' which make sense or could be efficiently matched against the network.

There has been significant interest in the degree distribution of networks, which is the distribution of the number of direct neighbors (vertex degree) of the network's nodes. While straightforward to collect, this statistic gives important insight into the underlying network generation process.

Finally, Watts and Strogatz introduce the clustering coefficient as a measure of "cliquishness of a friendship circle", which they define as the average over the rate of existing to possible edges among the direct neighbors of every node [80]. The clustering coefficient has a direct relation to the number of triangles in the graph, a topic which has researched in the data mining community as we discuss below.

While many more statistics over a network are possible, the ones discussed above are the most common in the literature concerning statistical models of graphs and are well studied.

1.4.1.2 Network models

Erdős-Rényi model

The simplest model from a conceptual point of view is the random-graphs model, attributed to Erdős and Rényi, in which all missing edges have the same (uniform) probability of appearing (in contrast with the Bernoulli-model first introduced by Gilbert in [35] for which the ER-model is often confused [10]; interestingly though, many properties are common for graphs from either graph probability space [9]). In a series of papers ([28, 29]) they study this model thoroughly and provide asymptotic bounds on many of its properties (degree distribution, probability of the all the nodes to belong to the giant

component, number of edge-choosing steps until the graph becomes fully connected). A number of graph properties have received significantly more attention, as to the researchers' surprise empirical observations strongly disagreed with the predictions and it is both what spurred a new wave of interest in the area of statistical models of networks and serve as a basis for classifying the networks as not being random. These properties are the degree distribution, the clustering coefficient and the diameter of the graph.

Erdős and Rényi proved [29] that the degree distribution of a graph generated by the ER-model follows a Poisson distribution. The bounds on the diameter are significantly more involved, and we will only mention that several small ($\sim log\ n$) values for the graph diameter are possible, depending on the value of model parameters [9].

Watts-Strogatz model
A distinguishing feature of many real-world networks is that, while having short diameters akin to random graphs, their clustering coefficient (and hence the frequency of triangles, and in fact also of other dense patterns) is significantly higher than what the ER-model predicts. This observation has lead Watts and Strogatz [80] to formally introduce the small-world model.

An interesting property of this model is that it allows one to vary the degree of "randomness" of the network it generates. The generation process is extremely simple and proceeds as follows: a regular graph (e.g. a ring lattice with fixed node degree d) on n nodes is generated, after which each edge is rewired with probability p [80]. This procedure has the effect of significantly reducing the diameter of the graph without having significant effect on the clustering coefficient (in a particular range of p).

While the WS-model represents a step forward in the understanding of real-world graphs, it captures a small number of statistical properties of interest.

Barabási-Albert model
The observation that a number of networks do not match the predictions of the ER-model for the degree distribution has been reported by Barabási and Reka in [6]. The authors suggest that two important processes occurring in real-world large-scale networks - network growth and preferential attachment to existing nodes - is not being accounted for and propose a model that incorporate said processes.

Since its introduction, the scale-free property has been confirmed in a large number of networks from a wide range of domains: biology, technological networks, citation networks, networks of chemical interactions etc. A detailed analysis of the network of scientific collaborations [7] has prompted the authors to introduce further refinements to this model, namely that "internal links" creation is also governed by the "rich get richer" scheme.

1.4.2 Pattern matching in a single network

We now continue our discussion of pattern mining, building on the insights gained in the transactional graph mining setting; and keeping in mind the statistical properties described in the previous section.

Given a database graph (a large network) D, the frequent subgraph pattern mining in D aims to find the set S of all the patterns P whose support is not less than a predefined threshold σ, that is,

$$S = \{P | f(D, P) \geq \sigma\},$$

where f is the support of the pattern P in the database graph D.

This problem of pattern mining poses two main chanllenges, pattern matching and frequecy counting. The first is related to deciding where a pattern occurs in the data, while the latter concerns the summarization of the occurrences in one signle number which makes sense statistically. First, in this section, we will consider the pattern matching problem. In the next section we survey work on support measures for single network pattern mining.

In particular, we need to find embeddings of a pattern P in the database graph G. The studied approaches can be divided into three main categories: *small pattern matching*, *exact matching* and *approximative matching*.

1.4.2.1 Matching small patterns

Due to the huge size of the networks, many studies only consider small patterns and either apply brute force or devise more efficient methods.

Triangles

Counting triangles has received quite some interest recently in the field of data mining, as it is the smallest non-trivial pattern matching problem, and is related to the calculation of the clustering coefficient. Furthermore, many interesting graph mining tasks are based on counting the number of triangles in the graph. For example, in [8] a method based on triangles was proposed for detecting spamming.

Both exact and approximate triangle counting algorithms have been proposed. In this section, we assume that G is an undirected loop-free graph with $n = |V(G)|$ and $m = |E(G)|$.

Exact triangle counting. The brute-force method, checking for every triple of vertices whether it forms a triangle. runs in $O(n^3)$ time.

The most efficient algorithms for counting triangles are based on matrix multiplication. The asymptotic time complexity of the fastest existing (theoretical) method for matrix multiplication takes $O(n^{2.37})$ time and $\Theta(n^2)$ space [21]. For sparse graphs, in [3], an $O(m^{1.41})$ time and $\Theta(n^2)$ space algorithm, NodeIterator, is proposed. It computes for each node its neighborhood and then checks how many edges exist among its neighbors. In [46], Itai and Rodeh exploited the NodeIterator idea to present an algorithm that counts the number of triangles in $O(m^{\frac{3}{2}})$ time. This algorithm computes spanning trees

of the graph and removes edges while checking that every triangle is listed exactly once.

Approximate triangle counting. [75] proposes a sampling approach. In particular, the proposed algorithm samples edges independently with probability p, counts the triangles in the resulting sparser graph, and divides the result by p^3. If p is not too small and triangles are sufficiently uniformly distributed, one can show that the result is a close approximation.

In [15], the authors present estimators for the number of triangles in the graph when the input is provided as a stream which is too large to store.

Network motifs

Network motifs are subgraphs which occur much more often than they occur in random networks [74]. Several researchers have studied in moderately large application domains motifs which are sufficiently small to match without significant computational challenges.

The authors of [74] found that much of a gene regulation network is composed of repeated appearances of three highly significant motifs. They showed that each network motif has a specific function in determining gene expression. Similarly, [58] presented a model of signaling pathways in hippocampal CA1 neurons, and found a high fraction of positive and negative feedback loop motifs.

In [62], the authors found network motifs in networks from biochemistry, neurobiology, ecology, and engineering. They saw that the motifs shared by ecological food webs were distinct from the motifs shared by the genetic networks of Escherichia coli or from those found in the World Wide Web.

1.4.2.2 Exact pattern matching

The subgraph isomorphism problem has been shown to be #P-complete in general, and almost all practical testing procedures are based on search with backtracking. An old but still widely used such backtracking algorithm is described in [76]. In [57] the search method exploits a heuristic derived from *constraint satisfaction* to reduce the cost. VF and its successer VF2 [22] are two search algorithms implementing a number of efficiently computable hueristics, and have been used in several transactional graph mining systems. There are algorithms using other strategies, e.g., decision tree based techniques [66]. The reader is referred to [61] for a short review of subgraph isomorphism algorithms. When the database graph is large and has a high average degree, these matching algorithms become intractable even for reasonably small patterns.

1.4.2.3 Approximative algorithms for pattern matching

When exact pattern matching for pattern mining is too expensive, one can resort to approximative algorithms. One option is to exploit statistical regularity in the graph. E.g. for certain pattern classes, there exist efficient algorithms which provide good approximations on almost all random graphs [31]

On the other hand, recent work on fixed parameter tractability has shown that there are algorithms, often randomized ones, whose assymptotic complexity is exponential in the pattern size but only polynomial (e.g. linear) in the network size. This is especially appealing when mining patterns, which are usually small, in large networks.

For instance if the patterns are trees, the subgraph isomorphism problem still remains #P-complete. However, based on recent advances in parameterized complexity theory, [50] proposed a randomized algorithm for mining rooted trees in large networks. This method finds all the homomorphisms first, and then with high probability removes those which are not isomorphisms exploiting properties of a specific algebraic structure [53]. This algorithm can mine all frequent rooted trees with delay linear in the size of the network and only mildly exponential in the size of the patterns.

1.4.2.4 Algorithms for approximate pattern matching

Inexact matching is also called error-tolerant graph matching. The major motivations of inexact matching algorithms are that 1) the collected data may have noise, and 2) patterns may have several variations. The tree search based strategy is also widely used in inexact matching and most of the algorithms are based on the *edit distance* (e.g. [14] and [71]). In order to compute the edit distance of two graphs, we first have to define a set of allowed operations with different costs on graphs. Usually, insertions, deletions and substitutions of vertices and edges are standard choices, and according to the applications, we also use other operations, like merging and spliting of vertices. Using these operations, a graph can be transformed to another graph, and different paths may exist. The graph edit distance of two graphs is the minimum cost path, and in most applications, to compute the optimal solution is extremely expensive. Practical approaches just find the suboptimal solution by relaxation [49]. Other strategies include continuous optimization , spectral methods , artificial neural networks, relaxation labeling and so on. We do not list all the algorithms here. See [20] for a comprehensive survey of pattern matching.

1.4.3 Pattern mining support measures

Pattern matching, discussed in the previous section, allows one to list all embeddings (or images) of a pattern P in a network G. A support measure (or frequency measure) is a function summarizing the embeddings of P into a single number reflecting how often P occurs in G.

The support of a pattern in the transactional setting is the number of graphs in the database which are supergraphs of that pattern. However, defining an appropriate support measure in a single graph is more challenging because two or more images of the pattern may overlap. If we count all the embeddings (or images) the support may not be anti-monotonic, which is problematic as usually the antimonotonicity allows for the most pruning in

the pattern search space. In fact, only anti-monotonicity is not enough for a practical support measure. For example, a support measure just returning a constant number is anti-monotonic, but not informative. Therefore, we often require a support measure to be *normalized*, i.e., it should return the number of images if there are only non-overlapping images.

An important class of normalized anti-monotonic support measures relies on the *overlap graph*. The nodes of an overlap graph G_P^D are the images of the pattern P in the database graph D, and two nodes are adjacent if the corresponding images overlap, i.e., they share at least a common edge (*edge-overlap*) or a common vertex (*vertex-overlap*).

Vanetik et al. [77] introduced the maximum independent set support measure (MIS), measuring the size of the maximum independent set of the overlap graph. Fiedler and Borgelt [30] proved that the MIS measure is anti-monotonic and claim that some cases of overlap can be ignored without affecting the anti-monotonicity of resulting support measures. Given a pattern $P = (V(P), E(P))$, there exists a *harmful overlap* of embeddings ϕ and ϕ' if there is a vertex $v \in V(P)$ such that $\phi(v), \phi'(v) \in \phi(V(P)) \cap \phi'(V(P))$. Vanetik et al. [77] gave a necessary and sufficient condition for anti-monotonicity of overlap graph based support measures. We cannot take the MIS support measure into practice directly because it is NP-hard to compute, and remains so even when the degree of the overlap graph is bounded. Kuramochi and Karypis [54] designed two practical mining algorithms using the MIS support measure. In their algorithms, the MIS support is computed approximately.

Bringmann and Nijssen [12] examined the expense of computing the MIS of overlap graphs, and then described another support measure *minimum image based support* which does not use overlap graphs. Given a pattern $P = (V(P), E(P))$, the minimum image based support is defined as,

$$minImage(D, P) = \min_{v \in V(P)} |\{\phi_i(v) | \phi_i \text{ is an embedding of } P \text{ in } D\}|.$$

This support measure is anti-monotonic, and it can be computed very efficiently. However, compared to overlap graph based support measures, the minImage overestimates the statistical evidence. For example, consider the following embeddings of some pattern: $(1, 11)$, $(2, 11)$, $(3, 11)$, $(4, 11)$, $(5, 11)$, $(6, 12)$, $(6, 13)$, $(6, 14)$, $(6, 15)$ and $(6, 16)$. The minImage returns 6 as the support of the pattern while MIS returns 2. From a statistical point of view, all embeddings either depend on vertex 6 or on vertex 11, so there are only two independent observations (if we consider embeddings which do not overlap as independent observations of some phenomenon). Therefore, the notion of independence has some advantages.

[16] generalized the conditions for anti-monotonicity of overlap graph based support measures. They showed that the conditions can be used whenever the matching operator is isomorphism, homomorphism or heomomorphism. Besides, the authors proposed two normalized anti-monotonic support measures. One is the MCP that the minimum clique partition of the overlap

graphs, which is also NP-hard to compute. Another is the Lovász theta value ϑ which can be computed with semidefinite programming, and hence in polynomial time. However, existing methods are still very expensive to compute ϑ [51, 47, 43]. We point out that the Schrijver theta value ϑ' [73] can be also used as a normalized anti-monotonic support measure. It is very similar to the ϑ, and we always have $MIS \leq \vartheta' \leq \vartheta \leq MCP$.

Wang and Ramon [78] observed that those images which share a common vertex (vertex-overlap) or a common edge (edge-overlap) build a clique in the overlap graph, and proposed the overlap hypergraph whose nodes are the images and hyperedges are these cliques. They introduced an overlap hypergraph based normalized anti-monotonic support measure `s` that is the solution of a (usually sparse) linear program which can be solved very efficiently using recently interier-point methods. More recently, they showed that this measure also has a natural statistical interpretation [79].

1.4.4 Applications

The literature has a huge amount of articles describing the analysis of large networks in a wide range of application domains. Here we only give a few examples:

- In *co-author networks*, two people are connected if they published a paper together [60].
- In *citation networks*, articles are represented with vertices and citations with edges. One example of such network is DBLP[1].
- In *molecule interaction networks*, two molecules are connected if they interact during at least one experiment. E.g., [17] describes a protein interactio network.
- In *communication networks*, two people are connected if they had at least one contact during a certain observation period. This can concern phone calls, SMS messages [27], emails, etc.

Several other types of networks can be found in the SNAP repository[2]. Often such networks are represented with undirected graphs, even though in many cases (e.g. when sending email from one person to another one) the relationship is directed.

Next to tasks described in detail in this chapter, typical other tasks include the discovery of communities (clusters of vertices being strongly connected while the connection with other parts of the network is relatively weak), and classification of vertex properties (e.g. prediction of the area of research based on the areas of co-authors).

[1] `http://www.informatik.uni-trier.de/ley/db/`
[2] `http://snap.stanford.edu/data/`

1.5 Concluding remarks

In this chapter, we have provided an introduction to and an overview of graph pattern mining techniques. An important issue in graph mining is the computational cost. We outlined strategies for analyzing and improving the complexity. Our discussion was illustrated with a few of the many applications of graph mining.

Clearly, many challenges are remaining in the field of graph mining. First, despite the significant progress during the last decades, graph mining is still computationally demanding. More algorithmic insights are needed to handle the challenge of generating in reasonable time good (approximative) results. A second challenge lies in the transformation of the gained information in terms of patterns and statistics into insights in the applications at hand. It is not because we can predict the actions of users that we understand his motivations and will still be able to predict well in a changing context. We anticipate that the exploration of expert advice and causal inference may be interesting paths in future work.

1.6 Additional reading

This chapter focused on graph pattern mining. [70] surveys prediction problems in graphs. Graphs from a probabilistic model point of view are studied in the fields of statistical relational learning [34].

The field of graph mining is related to many other fields in the literature which deserve reading. First, there are the fields of graph theory [26], [81] and algorithmic graph theory [52]. Many, often old, results provide excellent inspiration for improving graph mining algorithms. A work providing references classified by problem type is ([36]). Second, there is a large literature on techniques investigating large graphs and networks. Recent work in this direction can easily be found in the proceedings of recent editions of major data mining conferences such as KDD, ICDM, SDM and PKDD.

Part of the work on large networks is based on properties of the adjacency matrix. One example of this is [69]. The field investigating the properties of the adjacency matrix of graphs is called spectral graph theory [19].

1.7 Glossary

Association rule: a rule representing a correlation between two patterns, the antecedent and the consequent

Assymptotic complexity: an expression indicating how the cost of a method depends on the input parameters (usually size) for very large inputs

Canonical form: the assignment of a unique string representation to patterns, such that all equivalent patterns obtain the same string

Complexity: the cost of a method in terms of time or space.

Frequent: a property is called frequent when it occurs often in a database graph

Graph: a mathematical object consisting of a set of vertices and a set of edges between some of these vertices

Mining: discovering interesting patterns in data

Motif: see pattern

Network: the term network usually refers to a very large graph

Pattern: a motif or substructure which can occur in a database

Powerlaw: dependency of the form $f(x) = bx^a$ for some constants a and b.

1.8 Acknowledgements

This work has been supported by the ERC StG 240186 project "MiGraNT: Mining Graphs and Networks, a Theory-based approach".

Bibliography

[1] H. Agrawal, R.and Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[2] R. Agrawal and R. Srikant. Mining generalised association rules. In *Proceedings of the 21th VLDB Conference*, pages 407–419, 1995.

[3] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[4] C. Antunes and A. Oliveira. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In *Machine Learning and Pattern Mining in Data Recognition*, pages 239–251. Springer-Verlag, 2003.

[5] H. Arimura and T. Uno. An output-polynomial time algorithm for mining frequent closed attribute trees. In *Proceedings of the 15th International Conference on Inductive Logic Programming*, pages 1–19, 2005.

[6] A. L. Barabási and A. Reka. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[7] A. L. Barabási, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(3–4):590–614, aug 2002.

[8] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of ACM KDD*, 2008.

[9] B. Bollobás. *Random Graphs*. Cambridge University Press, 2001.

[10] B. Bollobás and P. Erdős. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 80, pages 419–427, 1976.

[11] C. Borgelt. Combining ring extensions and canonical form pruning. In *MLG*, 2006.

[12] B. Bringmann and S. Nijssen. What is frequent in a single graph? In *Proceedings of the 12th Pacific-Asian Conference on Knowledge Discovery and Data Mining*, pages 858–863, 2008.

[13] B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 55–66, 2006.

[14] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253, 1983.

[15] L. S. Buriol, G. Frahling, S. Leonardi, A. M. Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS*, 2006.

[16] T. Calders, J. Ramon, and D. Van Dyck. All normalized anti-monotonic overlap graph measures are bounded. *Data Mining and Knowledge Discovery*, 23:503–548, 2011.

[17] J. Chen, W. Hsu, M. Lee, and S. Ng. Nemofinder: Dissecting genome wide protein-protein interactions with repeated and unique network motifs. In *Proceedings of the 12th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 106–115, 2006.

[18] Y. Chi, R. R. Muntz, and J. N. Nijssen, S.and Kok. Frequent subtree mining - an overview. *Fundam. Inform.*, 66(1-2):161–198, 2005.

[19] F. Chung. *Spectral Graph Theory*. AMS Press, 1997.

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

[21] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM conference on Theory of computing (STOC)*, pages 1–6, 1987.

[22] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, October 2004.

[23] K. De Grave and F. Costa. Molecular graph augmentation with rings and functional groups. *Journal of Chemical Information and Modeling*, 50(9):1660–1668, 2010.

[24] L. De Raedt and J. Ramon. Condensed representations for inductive logic programming. In D. Dubois, C. A. Welty, and M. Williams, editors, *Proceedings of 9th International Conference on the Principles of Knowledge Representation and Reasoning*, pages 438–446. AAAI Press, 2004.

[25] R. Diestel. *Graph Theory.* Springer-Verlag, 2000.

[26] R. Diestel. *Graph Theory.* Springer-Verlag, 2010.

[27] N. Du, C. Faloutsos, B. Wang, and L. Akoghi. Large human communication networks: patterns and a utility-driven generator. In *Proceedings of 15th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 107–115, 2009.

[28] P. Erdős and A. Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[29] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[30] C. Fiedler, M.and Borgelt. Support computation for mining frequent subgraphs in a single graph. In *Proceedings of the fifth Workshop on Mining and Learning with Graphs (MLG'07)*, 2007.

[31] M. Fürer and P. K. Shiva. Approximately counting embeddings into random graphs. In *Proceedings of the 11th international workshop, APPROX 2008, and 12th international workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, APPROX '08 / RANDOM '08, pages 416–429, Berlin, Heidelberg, 2008. Springer-Verlag.

[32] G. C. Garriga, R. Khardon, and L. De Raedt. On mining closed sets in multi-relational data. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 804–809, 2007.

[33] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th Workshop on Experimental Algorithms*, pages 319–333, 2008.

[34] L. Getoor and B. Taskar. *An Introduction to Statistical Relational Learning.* MIT Press, 2007.

[35] E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, December 1959.

[36] J. L. Gross and J. Yellen. *Handbook of Graph Theory.* CRC Press, 2004.

[37] R. Gugisch and C. Rucker. Unified generation of conformations, conformers and steroisomers: a discrete mathematics approach. *MATCH Communications in Mathematics and Computer Chemistry*, 61:117–148, 2009.

[38] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMID International Conference on Management of Data*, pages 1–12, 2000.

[39] H. Hofer, C. Borgelt, and M. Berthold. Large-scale mining of molecular fragments with wildcards. In *Advances in Intelligent Data Analysis V*, pages 380–389, 2003.

[40] T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded treewidth. In *Proceedings of Principles and Practice of Knowledge Discovery in Databases 2008*, volume 5211, pages 520–535, Antwerp, Belgium, September 2008. Springer-Verlag.

[41] T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. *Knowledge Discovery and Data Mining*, (to appear), 2010.

[42] Tamás Horváthh and Jan Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411:2784–2797, 2010.

[43] T.-H. Hubert Chan, K. L. Chang, and R. Raman. An SDP primal-dual algorithm for approximating the lovász-theta function. In *ISIT*, pages 2808–2812. IEEE, 2009.

[44] A. Inokuchi. Mining generalized substructures from a set of labeled graphs. In *ICDM*, pages 415–418. IEEE Computer Society, 2004.

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

[46] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[47] G. Iyengar, D. J. Phillips, and C. Stein. Approximating semidefinite packing problems. Technical report, IEOR Department, Columbia University, New York, June 2009.

[48] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[49] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, August 2006.

[50] A. M. Kibriya and J. Ramon. Nearly exact mining of frequent trees in large networks. In *Proceedings of ECML/PKDD 2012*, pages 426–440. Springer, 2012.

[51] P. N. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. Technical Report CS-96-07, Department of Computer Science, Brown University, January 1996.

[52] W. Kocay and D. L. Kreher. *Graphs, algorithms and optimization.* Chapman & Hall, 2004.

[53] I. Koutis. Faster algebraic algorithms for path and packing problems. In *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[54] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3):243–271, 2005.

[55] M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. *Information Systems*, 32(8):1101–1120, 2007.

[56] A. S. LaPaugh and R. L. Rivest. The subgraph homeomorphism problem. In *STOC '78*, pages 40–50, New York, NY, USA, 1978. ACM Press.

[57] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4): 403–422, 2002.

[58] A. Maayan, S. L. Jenkins, S. Neves, A. Hasseldine, E. Grace, B. Dubin-Thaler, N. J. Eungdamrong, G. Weng, P. T. Ram, J. J. Rice, A. Kershenbaum, G. A. Stolovitzky, R. D. Blitzer, and R. Iyengar. Formation of regulatory patterns during signal propagation in a mammalian cellular network. *Science*, 309(5737):1078–1083, 2005.

[59] J. Matousek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108:343–364, 1992.

[60] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. A machine learning approach to building domain-specific search engines. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 662–667. Morgan Kaufmann, 1999.

[61] B. Messmer and H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Trans. on Knowledge and Data Engineering*, 12(2):307–323, 2000. Copyright by IEEE, http://www.ieee.org.

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

[63] S. Muggleton and C. D. Page. A learnability model for universal representations. In S. Wrobel, editor, *Proceedings of the Fourth International Workshop on Inductive Logic Programming*, pages 139–160, Sankt Augustin, Germany, 1994. GMD.

[64] S.-H. Nienhuys-Cheng and R. De Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence.* Springer-Verlag, New York, NY, USA, 1997.

[65] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 647–652, 2004.

[66] A. Pande, M. Gupta, and A. K. Tripathi. A decision tree approach for design patterns detection by subgraph isomorphism. In *ICT*, volume 101 of *Communications in Computer and Information Science*, pages 561–564. Springer, 2010.

[67] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pages 215–224, 2001.

[68] G. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, 1971.

[69] B. A. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. E. Faloutsos. Surprising patterns and scalable community chipping in large graphs. In *Proceedings of the14th Pacific Asian Conference on Knowledge Discovery and Data Mining*, volume 2, pages 435–448, 2010.

[70] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville. Transforming graph data for statistical relational learning. *Journal of Artificial Intelligence Research*, 45:363–441, 2012.

[71] A. Sanfeliu and K. S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13:353–363, 1983.

[72] L. Schietgat, F. Costa, J. Ramon, and L. De Raedt. Effective feature construction by maximum common subgraph sampling. *Machine Learning*, 2010. To appear.

[73] A. SCHRIJVER. A compafrison of the Delsarte and Lovász bounds. *IEEE Trans. Infor. Theory*, IT-25:425–429, 1979.

[74] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nat. Genet.*, 31(1): 64–68, 2002.

[75] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15:703–726, 2011.

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

[77] N. Vanetik, S. E. Shimony, and E. Gudes. Support measures for graph data. *Data Min. Knowl. Discov.*, 13(2):243–260, 2006.

[78] Y. Wang and J. Ramon. An efficiently computable support measure for frequent subgraph pattern mining. In *Proceedings of ECML/PKDD 2012*, pages 362–379. Springer, 2012.

[79] Y. Wang, J. Ramon, and T. Fannes. An efficiently computable and statistically motivated subgraph pattern support measure. *Data Mining and Knowledge Discovery*, 2013.

[80] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, June 1998.

[81] D. B. West. *An introduction to graph theory.* Prentice Hall, 2001.

[82] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 721–724, Japan, 2002. IEEE Computer Society.

[83] A. Zimmermann and L. De Raedt. Corclass: Correlated association rule mining for classification. In *Proceedings of the 7th International Conference on Discovery Science*, pages 60–72. Springer, 2004.