Graph Mining: An Overview

Christian Borgelt

European Centre for Soft Computing
Campus Mieres, Edificio Científico-Tecnológico
c/ Gonzalo Gutiérrez Quirós s/n, E-33600 Mieres (Asturias), Spain
Tel.: +34 985 456545, Fax: +34 985 456699
E-Mail: christian.borgelt@softcomputing.es

1 Introduction

In the early years of data mining and knowledge discovery in databases, method development focused on rigidly and plainly structured data. Most often efforts were even confined to data that can be represented as a simple table, which describes a set of sample cases by attribute-value pairs. Recent years, however, have seen a constantly growing interest in the analysis of more complex data, with a less rigid and/or more sophisticated structure.

In this development, two foci of attention can be identified: *text document analysis* and *graph mining*. The former tries to extract the rich information available in text documents like news articles, scientific papers, or blog posts, in order to make this information more easily accessible, to group related documents and rank them, and even to generate digests for more effective and efficient browsing [17]. The main problems in this area stem from the need to deal with natural language, which, due to its amazing variability and flexibility of expressing similar or even the same meaning in vastly differing ways and with different words, poses severe obstacles to linking the subjects of different documents automatically. Nevertheless, statistical approaches, which model texts as a so-called *bags of words* and rely mainly on term frequencies and their distribution in a collection of documents, have led to impressive results. More sophisticated approaches exploit lexical databases like WordNet [18] to resolve ambiguities and to deal with hyponyms and hypernyms.

The other area, graph mining, enjoys an increasing popularity due to two reasons: in the first place, a lot of data naturally comes in the form of graphs. Whether we deal with molecules, protein interaction networks, metabolic networks, the Internet and world wide web, co-author networks, social networks, etc., the corresponding data clearly describes a graph with nodes and edges. Secondly, basically all n to m relationships in a relational database can fairly naturally be represented or interpreted as a graph, thus making graphs a natural tool to analyze single- as well as multi-relational data. Examples are customer and product databases (containing information about who bought what), which are easily cast as a bipartite graph, or collaborative (or social) tagging systems, the data of which can conveniently be modeled as tripartite graphs of users, resources, and tags.

In this paper I try to give an overview of some of the main tasks and techniques of different subareas of graph mining. However, although I try to cover the core ideas in each of the subareas, I will provide a somewhat more detailed treatment only for the special task of frequent subgraph mining, which is my personal area of expertise. Apart from frequent subgraph mining (Section 6), I review the tasks of identifying global graph patterns (Section 2), classifying graphs (Section 3), predicting the labels of nodes and edges (Section 4), and predicting new or yet unknown links (Section 5).

2 Global Graph Patterns

Global graph patterns are similar in spirit to the characteristic measures used in descriptive statistics (like mean, variance, skew, etc.) or to the state variables used in thermodynamics (like pressure, temperature, energy, entropy, etc.). That is, they serve the purpose of characterizing complex graphs with few quantities, mainly with one of the following three goals: (1) find properties that distinguish real-world graphs from random graphs, (2) detect anomalies in a given graph, and (3) generate synthetic, but realistic graphs.

The seminal work in this area is the small-world network model [49], which introduced formal properties (specifically average path length and clustering coefficient, see below) to capture the small-world phenomenon that was observed in social networks [38, 45]. These properties made it possible to contrast real-world graphs to random graphs built with the Erdös–Rényi model [16] (edges occur independently with equal probability). The most common global graph properties in use nowadays are the following [8]:

Power laws. In many real-world graphs, many nodes have few connections, while only few nodes have a large number of connections. This can be expressed formally with a power law distribution for the degree x of the nodes: $p(x) = ax^{-b}$ where $a > 0$, $b > 1$ (needed so that the density can be normalized), and $x \geq x_{\min}$. Graphs exhibiting a power law degree distribution are often called *scale-free*, because the power law $y(x) = ax^{-b}$ is invariant up to a multiplicative factor, that is, $y(c_1 x) = c_2 y(x)$. The goal of a power law analysis is usually to find the value of b in order to characterize the graph with a single number. Power laws have been found to be empirically valid for the degree distribution of the Internet graph and the in- and out-degree distribution of the World Wide Web. However, checking for a power law distribution is often done sloppily, e.g., by looking at a scatter plot. More rigorous testing methods have only recently been developed.

Diameter / average path length. In real-world graphs, two nodes are usually not far apart in the sense that one can reach the one from the other on a fairly short path. This property can be captured by several closely related measures: (1) The *effective diameter* or *eccentricity* is the minimum number of hops (hop: following an edge to an adjacent node) in which some fraction (e.g. 90%) of all connected pairs of nodes can reach each other. (2) The *characteristic path length* is computed by taking the average length of the shortest paths from a given node to every other node, and then taking the median of these values over all nodes. (3) The *average diameter* is defined in basically the same way, only that the mean is computed instead of the median. If the graph is not connected, the characteristic path length and average diameter are computed only for the largest component. All of these measures yield fairly small values for a large variety of real-world graphs.

Community structure. A *community* is a set of nodes where each node is closer to the other nodes in the community than to nodes outside it. A community exhibits transitivity in a graph: if a and b are adjacent and b and c are adjacent, then a and c are likely also connected. This property is formally measured with the *clustering coefficient*. For a single node u with n_u neighbors, between which k_i edges exist, the clustering coefficient $C(u)$ is defined as k_i/n_i if $n_i > 1$ and 0 otherwise. A global clustering coefficient is computed by simple averaging: $C = \frac{1}{|U|}\sum_{u \in U} C(u)$. Alternatively, the clustering coefficient can be defined by drawing on the insight that transitivity occurs if *triangles* exist in the graph. Therefore one may define $C = (3 \cdot$ number of triangles) / number of connected triples, where a *connected triple* is any subgraph with three nodes and two edges. In this form C measures the fraction of connected triples that actually form triangles.

3 Graph Classification

In machine learning, classification is generally the task of predicting the label of an input object, and therefore graph classification is the task of predicting the label of an input graph. For example, we may want to predict the function of a protein [6] or whether a given molecule will be bioactive (that is, for example, will inhibit the growth of cancer cells). In statistics, however, classification is the task of placing given objects into groups—a task commonly called *clustering* in machine learning—and thus graph classification may also refer to graph clustering. Here we consider both tasks.

The most straightforward approach to graph classification (in either of the abovementioned meanings) is to reduce it by feature extraction to the task of classifying or grouping vectors of attribute-value pairs. That is, rather than capturing the actual structure of a graph, it is described by attributes that are derived from its structure and/or from the labels that may be associated with its nodes or its edges or with both. For example, a graph may be described by how many triangles it contains, by how many edges between nodes having a given label exist in it, of how many connected components it consists etc. Depending on how many different node and/or edge labels as well as specific substructures are considered, a graph may thus be described a huge number of attributes. This is also the main drawback of this approach: it is usually difficult to find the right features, so that a classical attribute-value based classifier can yield good results.

As a consequence, an alternative approach has gained increasing popularity, which focuses on graph similarity as a central problem for clustering and classification. The core idea is that classifiers like, for example, support vector machines require only a similarity measure that satisfies certain properties, and many clustering algorithms are able to work on a mere distance or similarity matrix as input. In such an approach the similarity of two graphs is computed with so-called *graph kernels*, where a *kernel* is generally a measure of similarity that is symmetric and positive semi-definite. The basic idea of graph kernels is to compare substructures that are traversable in polynomial time (like walks, paths, cyclic patterns, trees, etc.), where this restriction is meant to ensure that the kernel function can be evaluated in reasonable time. In contrast to this, more rigid approaches based, for instance, on maximum common subgraphs suffer from being NP-hard.

Since the seminal work [21], many different types of graph kernels have been designed, including so-called marginalized graph kernels [28, 29], subtree kernels [46], and rational kernels [10]. Good overviews, providing also a generalized view, can be found in [23, 48]. Here I focus on the concepts underlying random walk kernels [22], which are based on the following idea: if we consider the possible walks (paths) between two nodes, it is plausible that the two graphs are similar if, for most pairs of nodes, many walks are matching, so we simply count the number of matching walks. However, in order to ensure convergence, it is advisable to discount longer paths (i.e., give them a lower weight).

Technically, not all possible walks (paths) are considered, but a random sample, hence the name *random walk kernel*. Technically, the kernel is computed on the product graph of two given graphs G_1 and G_2, which consists of pairs of identically labeled nodes and edges from G_1 and G_2. The reason is that a random walk on the product graph is equivalent to simultaneous random walks on the input graphs. The kernel is then defined as

$$k(G_1, G_2) = \frac{1}{|G_1||G_2|} \sum_k \frac{\lambda^k}{k!} \, \vec{e}^{\top} A_{\times}^k \vec{e},$$

where $|G_i|$ is the number of nodes of graph G_i, $\vec{e}$ encodes a probability distribution on the nodes, which may be chosen as the uniform distribution, $A_\times$ is the adjacency matrix of the product graph, and $\lambda^k/k!$ is the discounting factor. The main problem with such an approach is that the product graph, and thus its adjacency matrix, is huge. However, with certain technical tricks the computation can be made feasible and efficient [47].

4 Label Prediction

Label prediction is a semi-supervised learning problem, also called *within-network classification*, which consists in the task to classify the nodes (and maybe also the edges) of a partly labeled graph (that is, to assign labels to them). Solutions to this problem have applications in image processing (i.e. classify the nodes of an image graph), document and web page classification, classifying protein interaction and gene expression data, part-of-speech tagging, fraud detection, customer suggestions and many other areas.

Many existing methods for label prediction are proximity-based: missing labels are inferred based on the hypothesis that linked or nearby nodes are likely to have the same labels [7, 20]. This hypothesis is also known as *homophily* (i.e., love of the same) and describes the tendency of individuals to associate and bond with similar others. Thus it is not surprising that these approaches work fairly well in social networks.

A similar approach is based on random walk kernels, similar to those for graphs [31]: it is plausible that two nodes are similar if they are connected by many walks. Long walks are discounted (receive a lower weight) in order to deal with the fact that the number of walks goes to infinity if the graph has cycles. Formally, walks are computed by taking powers of adjacency matrix A: $[A^k]_{ij} = c$ means that there are c paths of length k between nodes i and j. The kernel can then be defined as

$$k(i,j) = \left[\sum_k \frac{\lambda^k}{k!} A^k\right]_{ij},$$

where $\lambda^k/k!$ is the discounting factor with which a lower weight is assigned to longer walks. As an alternative, the kernel may be computed from the Laplacian matrix instead of the adjacency matrix. In this case it is known as a *diffusion kernel* and has been shown to be a valid positive semi-definite kernel. In both cases, however, the kernel is not evaluated for all paths, but only for a random sample; hence the name *random walk kernel*.

Unfortunately, the *homophily* assumption fails for many types of graphs, including trading networks, molecules, metabolic networks, etc. As an alternative to a *homophily*-based approach, such graphs can be handled with approaches based on the similarity of nodes, which can also be computed with kernel-like approaches, for example, with a marginalized similarity kernel [14]. In this approach random walks from two given nodes are compared in order to determine their similarity. Similar to the approach to graph kernels, two nodes are considered to be the more similar, the more matching walks start at them.

In all of these approaches it is important that the node labels should be inferred simultaneously instead of individually—a technique known as *collective classification*. This can be achieved in different ways, for example by iterative classification, which crisply assigns labels, or by learning the joint probability distribution of the labels. The latter can be achieved exactly using Markov Random fields and their extensions, or approximately e.g. by Gibbs sampling, but also by relaxation labeling or loopy belief propagation.

5 Link Prediction

Link prediction is the problem to predict between which nodes, which are unconnected in the current state of a graph, a new link (edge) will emerge or will be discovered. Solutions to this problem have applications in social network analysis (e.g.: which authors are likely to write a paper together soon?), web page improvement (e.g.: which references between web pages—for example: in Wikipedia—are missing?), and biological networks (e.g.: which protein interaction is not yet known, but may be discovered soon?).

The seminal work in this area considered co-author networks and tried to predict from a snapshot of a co-author graph which people, who did not work together in the past, would co-author a paper soon after the snapshot was taken [35]. More specifically, it was investigated how well topological and proximity measures for the nodes of a co-author graph are suited to predict emerging links between authors. Formally, each measure produces a ranked list of edges. Given the number n of edges that actually emerged, it is determined how many of these edges are among the top n edges of the ranked list.

The employed measures draw on ideas that already showed up in previous sections: they are either based on the direct neighbors and specifically the *common neighbors* of two unconnected nodes under consideration, or on the existing *paths (walks) between nodes*. Among the former are: (1) the simple *number of common neighbors* $|N(x) \cap N(y)|$, where $N(\cdot)$ denotes the set of neighbors of a node; (2) the *Jaccard coefficient* of the common neighbors, that is, $J(x, y) = |N(x) \cap N(y)| / |N(x) \cup N(y)|$; (3) the *Adamic/Adar measure* [1], in which common neighbors z are weighted with $1/\log(|N(z)|)$, that is, common neighbors with many neighbors contribute little weight; and (4) *preferential attachment*, which encodes the principle that nodes with many neighbors are more likely to get another neighbor than nodes with few neighbors, formalized as $|N(x)||N(y)|$. Among the latter (i.e. measures based on paths between the nodes) are: (1) the (negated) *length of the shortest path*; (2) the *unweighted Katz measure* [30], which sums over all paths between two nodes, discounting or dampening longer paths with a factor β^k where k is the path length; (3) the *weighted Katz measure* [30], which instead of simply counting the edges to determine the path length weights them with the number of existing co-authorships; (4) the *hitting time*, which is the expected number of steps of a random walk from one node to the other; (5) the *commute time*, which is the symmetrized hitting time; (6) the *rooted PageRank* [41], which is the stationary distribution weight of the node y under the following random walk: with probability α, jump to node x and with probability $1-\alpha$, go to random neighbor of the current node; and (7) the *SimRank* [27], which is recursively defined as two nodes being similar to the extent that they are joined to similar neighbors.

Experiments showed that basically all measures improve significantly over a random prediction, in which each non-existent edge has the same probability of emerging. Especially the Katz measures perform consistently well, but also simple measures like the number of common neighbors and the Adamic/Adar measure yield surprisingly good results.

The idea has been transferred to biological networks, where the task is to predict links that exist, but are not yet known [43]. The prediction is evaluated by comparing the predicted links to links that were discovered (with other, i.e., biological means) later.

Newer approaches to link prediction are based on the algebraic spectrum of a graph, which generalizes many graph kernels [33]. Note also that label prediction approaches that are based on proximity or node similarity can easily be transferred to link prediction.

6 Frequent Subgraph Mining

In analogy to the well-known task of frequent item set mining, with which item sets are found that are contained in a sufficiently large number of transactions of a given database (as specified by a user-provided minimum support), frequent subgraph mining tries to find (sub)graphs that are contained in a sufficiently large number of (attributed or labeled) graphs of a given graph database. Since the advent of this research area around the turn of the millennium, several clever algorithms for frequent subgraph mining have been developed. Some of them rely on principles from inductive logic programming and describe the graph structure by logical expressions [19]. However, the vast majority transfers techniques developed originally for frequent item set mining. Examples include MolFea [32], FSG [34], MoSS/MoFa [3], gSpan [50], CloseGraph [51], FFSM [26], and Gaston [39, 40]. A related, but slightly different approach is used in Subdue [11], which is geared towards graph compression with common subgraphs rather than frequent subgraph mining. An overview of several methods and related problems can be found in [12].

6.1 Motivation: Molecular Fragment Mining

Developing a new drug can take ten to twelve years (from the choice of the target to the introduction into the market) and the duration of the development process even increases continuously. At the same time the number of substances under development has gone down drastically. The reasons for these trends are raised safety standards for (new) drugs and the fact that due to rising research investments pharmaceutical companies must secure their market position and competitiveness by only a few, highly successful drugs (like Aspirin). As a consequence the chances for the development of drugs for target groups with rare diseases or with special diseases that are most frequent in developing countries (like AIDS) are considerably reduced, since the expected revenue from such drugs is low. In order to improve this situation, considerable efforts are devoted to significantly reduce the development time, by which one hopes to mitigate or even reverse this trend.

One way in which one tries to reduce the drug development time is to try to improve the discovery and optimization of candidate substances (so-called *pharmacophores*) and in particular to enhance the evaluation of high-throughput screening experiments. In such experiments a large number of potentially useful molecules are tested for activity w.r.t. some chosen target, for example, whether they are able to protect a human cell against a certain virus. The result is a database of chemical compounds together with activity information. An example of such a database is the DTP AIDS Antiviral Screen Database of the National Cancer Institute [15], in which about 40,000 chemical compounds are recorded together with a measurement of their ability to protect human CEM cells against an HIV-1 infection. An (adapted) excerpt from this database is shown in Table 1, where CI means "confirmed inactive", CM means "moderately active" (provided reproducibly 50% protection), and CA means "confirmed active" (provided reproducibly 100% protection). Molecules are described in the SMILES language (Simplified Molecular Input Line Entry System), which is a fairly popular description languages for molecules.

Molecular fragment mining is a very useful analysis method for such databases, which is based on frequent subgraph mining. The main goal is to identify molecular substructures that are frequent in the active, but rare in the inactive molecules, hoping that such fragments provide insights into what causes the activity. As an example, Figure 1 shows four

```
737,   CI,CN(C)C1=[S+][Zn]2(S1)SC(=[S+]2)N(C)C
2018,  CI,N#CC(=CC1=CC=CC=C1)C2=CC=CC=C2
19110,CI,OC1=C2N=C(NC3=CC=CC=C3)SC2=NC=N1
20625,CA,NC(=N)NC1=C(SSC2=C(NC(N)=N)C=CC=C2)C=CC=C1.OS(O)(=O)=O
22318,CI,CCCCN(CCCC)C1=[S+][Cu]2(S1)SC(=[S+]2)N(CCCC)CCCC
24479,CI,C[N+](C)(C)C1=CC2=C(NC3=CC=CC=C3S2)N=N1
55917,CI,O=C(N1CCCC[CH]1C2=CC=CN=C2)C3=CC=CC=C3
64054,CA,CC1=C(SC[C-]2N=C3C=CC=CC3=C(C)[N+]2=O)C=CC=C1
64055,CM,CC1=CC=CC(=C1)SC[C-]2N=C3C=CC=CC3=C(C)[N+]2=O
64057,CA,CC1=C2C=CC=CC2=N[C-](CSC3=NC4=CC=CC=C4S3)[N+]1=O
66151,CI,[O-][N+](=O)C1=CC2=C(C=NN=C2C=C1)N3CC3
...
```

Table 1: A fraction of the NCI DTP AIDS Antiviral Screen Database in a format in which the molecules are described in SMILES (Simplified Molecular Input Line Entry System). Each row consists of an identifier, an activity indicator, and a molecule description.

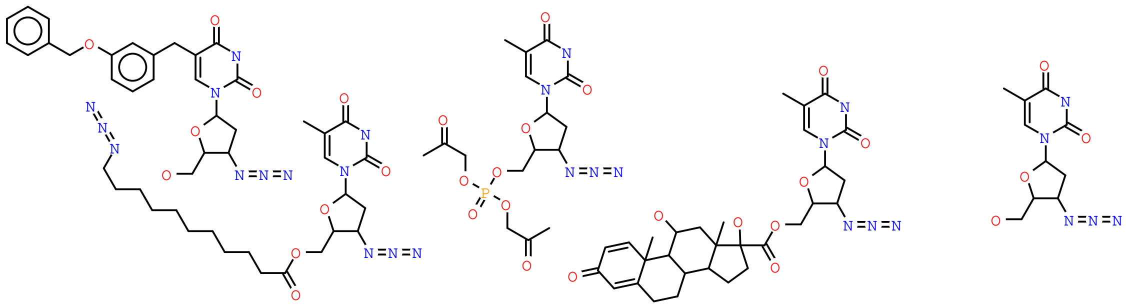


Figure 1: Four confirmed active molecules from the NCI DTP AIDS Antiviral Screen Database (left) and a molecular fragment that is part of all four of them (right).

confirmed active molecules from the DTP AIDS Antiviral Screen Database as well as a fragment that is common to all of them. This fragment is the characteristic substructure of a class of substances known as AZT, which are currently used as AIDS medication.

6.2 Notation, Presuppositions, and Problem Definition

Formally, frequent subgraph mining works on a database of *labeled graphs* (also called *attributed graphs*). A labeled graph is a triple $G = (V, E, l)$, where V is the set of vertices, $E \subseteq V \times V - \{(v, v) \mid v \in V\}$ is the set of edges, and $l : V \cup E \rightarrow L$ assigns labels from some label set L to vertices and edges. The graphs we consider are undirected and simple (that is, there is at most one edge between two given vertices) and contain no loops (that is, there are no edges connecting a vertex to itself). However, graphs without these restrictions (that is, directed graphs, graphs with loops and/or multiple edges) can be handled as well with properly adapted methods. Note also that several vertices and edges may carry the same label, which is one core reason for the complexity of the problem.

The *support* $s_{\mathcal{G}}(S)$ of a (sub)graph S w.r.t. a given graph database $\mathcal{G}$ is the number of graphs $G \in \mathcal{G}$ it is contained in. What is meant by a graph being contained in another is made formally precise by the notion of a *subgraph isomorphism*. Given two labeled graphs $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$, a subgraph isomorphism of S to G is

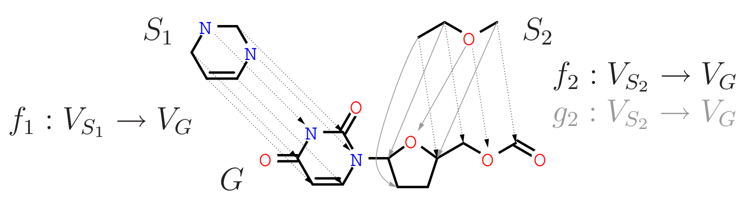


Figure 2: Examples of subgraph isomorphisms with two fragments and a molecule that contains them in different ways.

an injective function $f : V_S \to V_G$ satisfying (1) $\forall v \in V_S : l_S(v) = l_G(f(v))$ and (2) $\forall (u, v) \in E_S : (f(u), f(v)) \in E_G \wedge l_S((u, v)) = l_G((f(u), f(v)))$. That is, the mapping f preserves the connections between vertices and the labels of both vertices and edges.

Note that there may be several ways to map a labeled graph S to a labeled graph G, so that connections and labels are preserved. For example, G may possess several subgraphs that are isomorphic to S. It may even be that S can be mapped in several different ways to the same subgraph of G. This is the case if there exists a subgraph isomorphism of S to itself (a *graph automorphism*) that is not the identity. Examples of subgraph isomorphism w.r.t. a molecule G and two molecular fragments S_1 and S_2, all of which are modeled as labeled graphs, are shown in Figure 2. Fragment S_2 is contained in several different ways in molecule G, two of which are indicated. Here, however, we neglect that there may be several subgraph isomorphism and consider only whether there exists at least one or not.

Given a database $\mathcal{G}$ of labeled graphs and a user-specified minimum support $s_{\min} \in \mathbb{N}$, a (sub)graph S is called *frequent* in $\mathcal{G}$ iff $s_{\mathcal{G}}(S) \geq s_{\min}$. The task of frequent subgraph mining is to identify all subgraphs that are frequent in a given graph database $\mathcal{G}$. However, the output is usually restricted to *connected subgraphs* for two reasons: in the first place, connected subgraphs suffice for most applications. Secondly, restricting the result to connected subgraphs considerably reduces the search space, which otherwise is so huge that searching it becomes infeasible even for small graph databases.

6.3 Search Space

In order to search for frequent subgraphs, we consider, in analogy to frequent item set mining, the semi-lattice of subgraphs of the graphs of a given database. That it is only semi-lattice is due to the fact that there is no natural largest element, since there is no largest labeled graph for any given label set L. Therefore we use the database graphs as maximal elements, which, however, are usually not comparable. A *Hasse diagram* of an example semi-lattice, for a simple database of three molecule-like graphs (no chemical meaning attached, to be seen at the bottom) is shown in Figure 3 on the left: two subgraphs are connected if one possesses an additional edge (and an additional vertex) compared to the other. The frequent subgraphs are located at the top of this semi-lattice, thus suggesting a top-down search, just as it is used for frequent item set mining.

Therefore the search for frequent subgraphs consists in growing subgraphs into the graphs of the given database, step by step adding edges and vertices, until the support falls below the threshold. Like in frequent item set mining, a core problem of the search is that the same graph can be reached on different paths, as is easily visible in the subgraph semi-lattice shown in Figure 3. Hence we face the task to avoid redundant search. The solution principle is to assign a unique parent to each subgraph, which turns the subgraph semi-lattice into a subgraph tree. This is illustrated in Figure 3 on the right (the principle underlying the assignment of unique parents is discussed below, see Section 6.4).

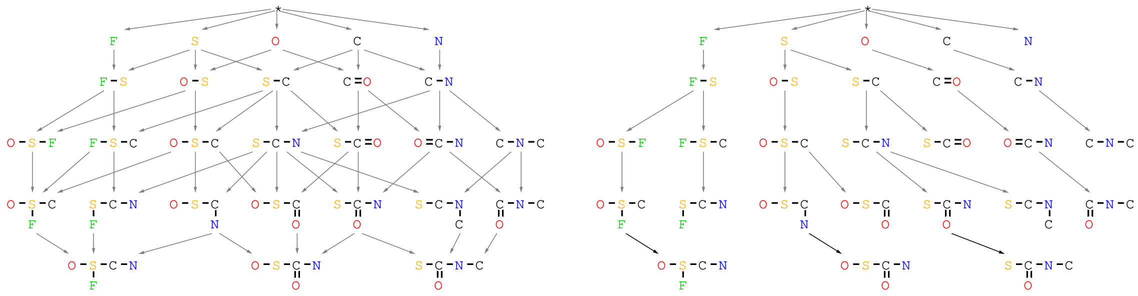


Figure 3: A semi-lattice of subgraphs for three molecule graphs, to be seen at the bottom (left) and an assignment of unique parents, which turns the semi-lattice into a tree (right).

With a unique parent for each subgraph, we can carry out the search for frequent subgraphs according to the following simple recursive scheme: in a base loop, all possible vertex labels are traversed (their unique parent is the empty graph). All vertex labels (and thus all single vertex subgraphs) that are frequent are processed recursively. A given frequent subgraph S is processed recursively by forming all possible extensions R of S by a single edge (also adding a vertex except when the edge closes a cycle), for which S is the chosen unique parent. All such extensions R that are frequent (that is, for which $s_{\mathcal{G}}(R) \geq s_{\min}$) are processed recursively, while infrequent extensions are discarded.

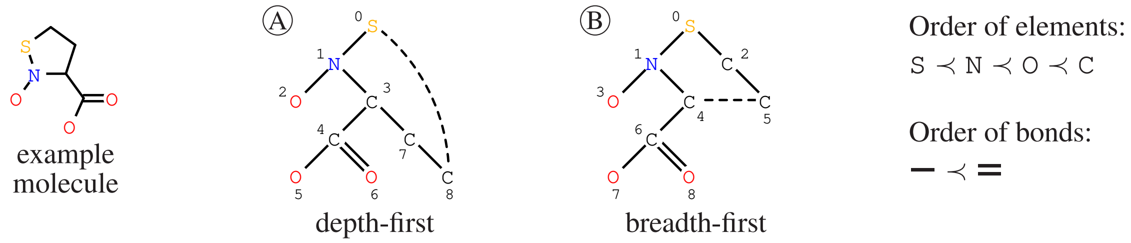
6.4 Canonical Forms of Graphs

In the search for frequent item sets it is trivial to assign unique parents, namely by choosing an (arbitrary, but fixed) order of the items and defining the parent of an item set as the set that results if its maximum element w.r.t. this chosen order is removed. Unfortunately, in the search for frequent subgraphs ordering the labels in the set L, though also necessary, is not enough. The reason is mainly that several vertices (and several edges) may carry the same label. Hence the labels do not uniquely identify a vertex, thus rendering it impossible to describe the graph structure solely with these labels. We rather have to endow each vertex with a unique identifier (usually a number), so that we can unambiguously specify the edges of the graph and which of them has to be removed to obtain the parent.

Given an assignment of unique vertex identifiers, we can describe a graph with a *code word*, which specifies the vertex and edge labels and the connection structure, and from which the graph can be reconstructed. Of course, the form of this code word depends on the employed numbering of the vertices: each numbering leads to a different code word. In order to single out one of these code words as the so-called *canonical code word*, we simply select the lexicographically smallest (or largest) code word.

With canonical code words, we can easily define unique parents: the canonical code word of a (sub)graph S is obtained from a specific numbering of its vertices and thus also fixes (maybe with some additional stipulation) an order of its edges. By removing the last edge in this order, which is not a bridge or which is incident to at least one vertex with degree 1 (which is then also removed), we obtain a connected graph that is exactly one level up in the semi-lattice of subgraphs and thus may be chosen as the unique parent of S.

Technically, canonical code words for graphs can either be based on a systematic way of constructing a spanning tree [50, 4] or on (extended) adjacency matrices [36]. Here we



Code words: A: `S 10-N 21-O 31-C 43-C 54-O 64=O 73-C 87-C 80-C`
B: `S 0-N1 0-C2 1-O3 1-C4 2-C5 4-C5 4-C6 6-O7 6=O8`

Figure 4: Spanning trees for an example graph, vertex numberings that have been derived from them with depth-first or breadth-first traversal, and the corresponding code words.

consider only the former: starting at an arbitrary vertex, a spanning tree of a given graph is formed either by a depth-first or by a breadth-first search. The vertices of the graph are numbered in the order in which they are visited and are thus endowed with unique identifiers. A code word is then formed by appending edge descriptions to a single letter stating the label of the root vertex, thus yielding the following general code words:

depth-first:	$a\,(i_d\,\underline{i_s}\,b\,a)^m$	
breadth-first:	$a\,(i_s\,b\,a\,i_d)^m$	(or $a\,(i_s\,i_d\,b\,a)^m$),

where i_s and i_d are the source and destination index of the incident vertices (using the convention that the incident vertex with the smaller index is the source), b is an edge label, a is a vertex label, and m is the number of edges. The edge descriptions are generally listed in the order in which the edges are visited in the spanning tree traversal. However, for both depth-first and breadth-first this can also be described by a sorting order on the edge descriptions, specified by a precedence order of their elements for comparisons, which is given in the above regular expressions by the order in which the elements are listed. All elements are compared ascendingly, with the exception of the source index in the depth-first case, which is compared descendingly (indicated by the underscore).

As an illustration, Figure 4 shows an example molecule and two spanning trees, in which the vertices are numbered in the order in which they are visited by a depth-first (A) or breadth-first search (B). The resulting code words are shown at the bottom of the figure. It is not difficult to verify that these code words are actually the lexicographically smallest code words that can result from different choices of the root vertex and different ways of forming the spanning trees. (Even though the general traversal order is fixed, there is no a-priori rule in which order to visit the neighbors of a vertex.) Therefore they are the canonical code words for these graphs, for these two specific code word construction methods. As a consequence, we see that the parent of the example molecule w.r.t. a depth-first code word is the graph in which the edge between sulfur and carbon is missing, while the parent w.r.t. a breadth-first code word is the graph in which the double bond is missing.

Constructing a canonical code word, or testing whether a given code word is canonical, is done by recursively enumerating (prefixes of) all possible code words (see [4] for details). However, the above canonical forms possess the *prefix property*: any prefix of a canonical code word is a canonical code word itself. This property simplifies the search: code words of extensions can be formed by simply appending the description of the added edge to the canonical code word of the (sub)graph that is extended. If the result is a canonical code word, the extension was created from the correct parent, otherwise it is to be discarded. Furthermore, this possibility enables us to determine in many cases with very simple checks whether the result of an extension is canonical and thus whether it has been created from the correct parent [4]. Unfortunately, though, these *simple rules* are reliable only if they state the the result is not canonical. Otherwise a full canonical form test is needed in order to be sure that an extension needs to be processed.

6.5 Search Algorithms

The basic search scheme was already outlined in Section 6.3. Nevertheless many different concrete approaches are possible, for example, depending on the order in which the two properties are tested that an extended graph must have in order to be processed recursively: it must have been created from the correct parent (which is determined with the help of a canonical form test) and it must have at least the minimum support. Which property is easier or faster to test may determine the order in which they are tested.

In addition, one can consider two possible ways of forming possible extensions of a given graph. The first method records all occurrences of a subgraph to extend and checks what extensions are actually possible in the database graphs. In this case the support of each extension is readily available and thus checked first against the minimum support. Only for those extensions that satisfy the minimum support requirement a canonical form test is carried out. Alternatively, the possible extensions may be determined from general information about the database of graphs, like which edge labels occur together with which vertex labels. In this case the support is not readily available, as one must check in which database graphs a created extension occurs. As a consequence, it may be advantageous to check for canonical form first and to compute the support (by subgraph isomorphism tests on the database graphs) only for those extensions that are canonical. Generally the first scheme is advantageous if the database graphs have many different labels, because in this case using only general database information yields too many extensions. In database with few or even no labels, however, the second scheme is preferable, because it avoids the storage and computation overhead of maintaining lists of all occurrences [13].

6.6 Other Techniques

Since they are for general graphs, the methods described above are, of course, also applicable to trees and sequences (modeled as chains). However, for these restricted classes of graphs there exist specialized canonical forms that allow for perfectly reliable *simple rules*, so that a costly backtracking check for canonical form can be avoided [9, 40]. Such a canonical form is even known for the more general class of outerplanar graphs [25].

In the specific area of molecular fragment mining, special methods have been developed to deal with rings [24, 5] (which chemists often treat as irreducible building blocks), carbon chains [37] and wild cards [24]. The goal of these approaches is to make the output of a frequent subgraph mining algorithm easier to interpret and more useful for a chemist.

An interesting alternative to a canonical form based search is an enumeration scheme that stores part of the subgraph lattice together with certain transformations in order to avoid redundant search and to achieve polynomial delay in the enumeration [42].

7 Outlook

Similar to recent developments in other areas of data mining, there is currently a strong shift of focus towards dynamic aspects of graphs, that is, towards change mining on graph databases, which are observed over time. This includes, for example, global graph patterns in evolving graphs like the densification power law and shrinking diameters [8] as well as graph evolution rules [2], but also link prediction (see Section 5).

For applications, it appears to be mandatory to include (more) background knowledge into the mining process, especially in frequent subgraph mining. Without problem-specific chemical, biological, social etc. background knowledge, it seems infeasible to restrict the huge search space to the relevant part and to reduce the output from simply frequent subgraphs to the actually interesting and useful ones.

References

[1] L.A. Adamic and E. Adar. Friends and Neighbors on the Web. *Social Networks* 25(3):211–230. Elsevier Science, Amsterdam, Netherlands 2003

[2] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis. Mining Graph Evolution Rules. *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2009, Bled, Slovenia)*, LNAI 5781:115–130. Springer, Heidelberg, Germany 2009

[3] C. Borgelt and M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proc. IEEE Int. Conf. on Data Mining (ICDM 2002, Maebashi, Japan)*, 51–58. IEEE Press, Piscataway, NJ, USA 2002

[4] C. Borgelt. On Canonical Forms for Frequent Graph Mining. *Proc. 3rd Int. Workshop on Mining Graphs, Trees and Sequences (MGTS'05, Porto, Portugal)*, 1–12. ECML/PKDD 2005 Organization Committee, Porto, Portugal 2005

[5] C. Borgelt. Combining Ring Extensions and Canonical Form Pruning. *Proc. 4th Int. Workshop on Mining and Learning with Graphs (MLG 2006, Berlin, Germany)*, 109–116. ECML/PKDD Organization Committee, Berlin, Germany 2006

[6] K. Borgwardt, C. Ong, S. Schönauer, S.V.N. Vishwanathan, A. Smola, and H.-P. Kriegel. Protein Function Prediction via Graph Kernels. *Bioinformatics* 21(1):47–56. Oxford University Press, Oxford, United Kingdom 2005

[7] J. Callut, K. Francoisse, M. Saerens, and P. Dupont. Semi-supervised Classification from Discriminative Random Walks. *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2008, Antwerp, Belgium)*, LNAI 5211:162-ÂŰ177. Springer, Heidelberg, Germany 2008

[8] D. Chakrabarti and C. Faloutsos. Graph Mining: Laws, Generators and Algorithms. *ACM Computing Surveys* 36(1):article 2. ACM Press, New York, NY, USA 2006

[9] Y. Chi, S. Nijssen, R.R. Muntz, and J.N. Kok. Frequent Subtree Mining - An Overview. *Fundamenta Informaticae* XXI:1001-1038. IOS Press, Amsterdam, Netherlands 2001

[10] C. Cortes, P. Haffner, and M. Mohri. Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research* 5:1035–1062. Massachusetts Institute of Technology, Cambridge, MA, USA 2004

[11] D.J. Cook and L.B. Holder. Graph-Based Data Mining. *IEEE Trans. on Intelligent Systems* 15(2):32–41. IEEE Press, Piscataway, NJ, USA 2000

[12] D.J. Cook and L.B. Holder. *Mining Graph Data*. J. Wiley & Sons, Chichester, United Kingdom 2007

[13] C. Desrosiers, P. Garnier, P. Hansen, and A. Hertz. Improving Frequent Subgraph Mining in the Presence of Symmetry. *Proc. 5th Int. Workshop on Mining and Learning with Graphs (MLG 2007, Florence, Italy)*, 25–30. MLG 2007 Organization Committee, Florence, Italy 2007

[14] C. Desrosiers and G. Karypis. Within-Network Classification Using Local Structure Similarity. *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2009, Bled, Slovenia)*, LNAI 5781:260–275. Springer, Heidelberg, Germany 2009

[15] *DTP AIDS Antiviral Screen (HIV Data Set) — Subset from 2001*. Developmental Therapeutics Program (DTP), National Cancer Institute, USA 2001 `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

[16] P. Erdös and A. Rényi. On Random Graphs. *Publ. Math. Debrecen* 6:290–297. Institute of Mathematics, University of Debrecen, Hungary, 1959

[17] R. Feldman and J. Sanger. *The Text Mining Handbook*. Cambridge University Press, Cambridge, United Kingdom 2006

[18] C. Fellbaum (ed.) *WordNet — An Electronic Lexical Database*. MIT Press, Cambridge, MA, USA 1998

[19] P.W. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacore Discovery Using the Inductive Logic Programming System PROGOL. *Machine Learning*, 30(2-3):241–270. Kluwer, Amsterdam, Netherlands 1998

[20] B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos. Using Ghost Edges for Classification in Sparsely Labeled Networks. *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 256-ÂŰ264. ACM Press, New York, NY, USA 2008

[21] T. Gärtner. Exponential and Geometric Kernels for Graphs. *Proc. NIPS*2002 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*. NIPS*2002 organization committee, Vancouver, Canada 2002

[22] T. Gärtner, P. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. *Proc. Annual Conf. Computational Learning Theory*, 129–143. Springer, New York, NY, USA 2003

[23] T. Gärtner. *Kernels for Structured Data*. World Scientific, Hackensack, NJ, USA 2008

[24] H. Hofer, C. Borgelt, and M.R. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Intelligent Data Analysis*, 8:495–504. IOS Press, Amsterdam, Netherlands 2004

[25] T. Horvath, J. Ramon, and S. Wrobel. Frequent Subgraph Mining in Outerplanar Graphs. *Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 197–206. ACM Press, New York, NY, USA 2006

[26] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. *Proc. 3rd IEEE Int. Conf. on Data Mining (ICDM 2003)*, 549–552. IEEE Press, Piscataway, NJ, USA 2003

[27] G. Jeh and J. Widom. SimRank: A Measure of Structural Context Similarity. *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2002, Edmonton, Canada)*, 538–543. ACM Press, New York, NY, USA 2002

[28] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized Kernels between Labeled Graphs. *Proc. Int. Conf. on Machine Learning*, 321–328. Morgan Kaufmann, San Mateo, CA, USA 2003

[29] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels on Graphs. In: K. Tsuda, B. Schölkopf, and J. Vert (eds.) *Kernels and Bioinformatics*, 155–170. MIT Press, Cambridge, MA, USA 2004

[30] L. Katz. A New Status Index Derived from Sociometric Analysis. *Psychometrika* 18(1):39–43. Psychometric Society, Greensboro, NC, USA 1953

[31] I.R. Kondor amd J.D. Lafferty. Diffusion Kernels on Graphs and Other Discrete Structures. *Proc. Int. Conf. on Machine Learning*, 315–322. Morgan Kaufmann, San Mateo, CA, USA 2002

[32] S. Kramer, L. de Raedt, and C. Helma. Molecular Feature Mining in HIV Data. *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2001, San Francisco, CA)*, 136–143. ACM Press, New York, NY, USA 2001

[33] J. Kunegis and A. Lommatzsch. Learning Spectral Graph Transformations for Link Prediction. *Proc. 26th Int. Conference on Machine Learning (ICML'09, Montreal, Canada)*, 1–8. ACM Press, New York, NY, USA 2009

[34] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. *Proc. 1st IEEE Int. Conf. on Data Mining (ICDM 2001, San Jose, CA)*, 313–320. IEEE Press, Piscataway, NJ, USA 2001

[35] Liben-Nowell and J. Kleinberg. The Link Prediction Problem for Social Networks. *Proc. 12th Annual ACM Int. Conf. on Information and Knowledge Management (CIKM'03)*, 556–559. ACM Press, New York, NY, USA 2003

[36] B. McKay. Practical Graph Isomorphism. *Congressus Numerantium* 30:45-âĂŞ87. Utilitas Mathematica Publishing, Winnipeg, Canada 1981

[37] T. Meinl, C. Borgelt, and M.R. Berthold. Mining Fragments with Fuzzy Chains in Molecular Databases. *Proc. 2nd Int. Workshop on Mining Graphs, Trees, and Sequences (MGTS 2004 at PKDD 2004, Pisa, Italy)*, 49–60. ECML/PKDD Organization Committee, Pisa, Italy 2004

[38] S. Milgram. The Small World Problem. *Psychology Today* 1:61–67. Sussex Publishers, New York, NY, USA 1967

[39] S. Nijssen and J.N. Kok. A Quickstart in Frequent Structure Mining Can Make a Difference. *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD2004, Seattle, WA)*, 647–652. ACM Press, New York, NY, USA 2004

[40] S. Nijssen and J.N. Kok. The Gaston Tool for Frequent Subgraph Mining. *Electronic Notes in Theoretical Computer Science* 127(1):77-87. Elsevier Science, Amsterdam, Netherlands 2005

[41] L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report, Stanford University, Palo Alto, CA, USA 1999

[42] J. Ramon and S. Nijssen. Polynomial-Delay Enumeration of Monotonic Graph Classes. *Journal of Machine Learning Research* 10:907–929. MIT Press, Cambridge, MA, USA 2009

[43] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link Discovery in Graphs Derived from Biological Databases. *Proc. 3rd Int. Workshop on Data Integration in the Life Sciences (DILS'06, Hinxton, United Kingdom)*, LNBI 4705:35–49. Springer, Heidelberg, Germany 2006

[44] L.T. Thomas, S.R. Valluri, and K. Karlapalem. MARGIN: Maximal Frequent Subgraph Mining. *Proc. 6th IEEE Int. Conf. on Data Mining (ICDM 2006)*, 1097–1101. IEEE Press, Piscataway, NJ, USA 2006

[45] J. Travers and S. Milgram. An experimental study of the Small World Problem. *Sociometry* 32(4):425ÂŰ443. American Sociological Association, Washington, DC, USA 1969

[46] S.V.N. Vishwanathan and A. Smola. Fast Kernels for String and Tree Matching. *Advances in Neural Information Processing Systems 15*, 569–576. MIT Press, Cambridge, MA, USA 2003

[47] S.V.N. Vishwanathan, K.M. Borgwardt, and N.N. Schraudolph. Fast Computation of Graph Kernels. *Advances in Neural Information Processing Systems 19*, 1449–1456. MIT Press, Cambridge, MA, USA 2007

[48] S.V.N. Vishwanathan, K.M. Borgwardt, I.R. Kondor, and N.N. Schraudolph. Graph Kernels. *Journal of Machine Learning Research* (submitted). Massachusetts Institute of Technology, Cambridge, MA, USA 2008 (available at arXiv:0807.0093)

[49] D.J. Watts and S.H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature* 393(6684):440-442. Nature Publishing Group, New York, NY, USA 1998

[50] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. *Proc. 2nd IEEE Int. Conf. on Data Mining (ICDM 2003, Maebashi, Japan)*, 721–724. IEEE Press, Piscataway, NJ, USA 2002

[51] X. Yan and J. Han. Closegraph: Mining Closed Frequent Graph Patterns. *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2003, Washington, DC)*, 286–295. ACM Press, New York, NY, USA 2003