UNIVERSITÄT BONN ML AI lab

Master's Thesis

Frequent Approximate Subgraph Mining with Polynomial Delay

written and submitted by

Richard Palme
born April 20, 1991 in Troisdorf

1st Examiner:	Dr. Pascal Welke University of Bonn
2nd Examiner:	Prof. Dr. Petra Mutzel University of Bonn
Advisor:	Till Hendrik Schulz University of Bonn

December 20, 2021

Abstract

Generating the set of frequent subgraphs of a graph database is a fundamental problem that has many applications, not least as a feature generation step that enables the application of standard machine learning methods to graph databases.
We propose an algorithm that generates frequent subgraphs using a depth-first search while exploiting the Apriori Principle. We empirically show that our algorithm is often an order of magnitude faster than a baseline DFS algorithm that does not apply the Apriori Principle.
Further, we propose a generalization of the frequent subgraph mining problem that generates subgraphs that are approximately frequent. We show that our problem formulation guarantees that the number of frequent approximate subgraphs is bounded in terms of the number of frequent subgraphs.
Finally, we show that algorithms for frequent subgraph mining can be adapted to generate a subset of the set of frequent subgraphs with polynomial delay, which means that the time between printing two consecutive frequent subgraphs, as well as the time before printing the first, is bounded by a polynomial in the size of the database.

Acknowledgments

I would like to express my gratitude to Till Schulz and Dr. Pascal Welke for making many useful suggestions and for providing invaluable insights into the field of graph mining during numerous interesting conversations. A special thank you goes to Till Schulz for reading a draft version of this thesis and for providing valuable feedback. Moreover, I would like to thank Dr. Tamàs Horvàth, Prof. Dr. Petra Mutzel, Till Schäfer, and Florian Seiffarth for their feedback and for asking helpful questions. Most of all, I want to thank my parents for always supporting and encouraging me.

Statement of Authorship

I hereby confirm that the work presented in this master thesis has been performed and interpreted solely by myself except where explicitly identified to the contrary. I declare that I have used no other sources and aids than those indicated. This work has not been submitted elsewhere in any other form for the fulfillment of any other degree or qualification.

Bonn, December 20, 2021 _____

Contents

Acronyms

BFS	Breadth-first search.
DFS	Depth-first search.
FASM	Frequent approximate subgraph mining (Definition 5.6).
FCSM	Frequent connected subgraph mining (Definition 3.14).
GED	Graph edit distance (Definition 3.27).
GI	Graph isomorphism (Definition 3.6).
SGED	Subgraph edit distance (Definition 5.1).
SGI	Subgraph isomorphism (Definition 3.8).

Chapter 1

Introduction

Embedding graphs into a vector space enables the application of machine learning methods to graph databases. Once graphs are represented by vectors, machine learning methods designed for vector-valued data (e.g. SVMs, neural networks, ...) can be applied to graph data. However, there is no known polynomial-time algorithm that assigns a unique vector to every graph while being invariant under graph isomorphism, so representing graphs by vectors is a challenging problem. In practice, a graph embedding only needs to embed graphs from one particular application domain into a vector space, so restricting graph embeddings to a single application domain does not reduce their applicability. Some examples of application domains for graphs are chemical molecules, social networks, or knowledge graphs.

A popular method for embedding domain-specific graphs into a vector space is to compute the frequent subgraphs of a graph database D, and to use the frequent subgraphs of D in order to construct feature vectors. Here, the frequent subgraphs of D are connected graphs whose structure is contained in the structure of at least t graphs in D, and D is a database that contains graphs from the desired application domain. The problem of generating all frequent subgraphs of D is called the frequent connected subgraph mining problem (FCSM), and the problem of deciding whether the structure of a graph H is contained in the structure of a graph G is called the subgraph isomorphism problem (SGI). Any graph G sampled from the same distribution as the graphs in D can be represented by a binary feature vector that indicates, for each frequent subgraph of D, whether the frequent subgraph is subgraph isomorphic to G. This method of embedding domain-specific graphs into a feature space has been successfully applied by Deshpande et al. (2005) to predict the bioactivity of molecules.

In this thesis, we propose an algorithm for FCSM that traverses the search space using a depth-first search while exploiting the Apriori Principle. The Apriori Principle states that any graph G containing an infrequent subgraph H can not be frequent, so there is no need to evaluate the frequency of such a graph G in the database. The Apriori Principle has been applied to breadth-first search approaches for FCSM, but no previous depth-first search approach for FCSM applies the Apriori Principle. We empirically evaluate our algorithm and show that it is often an order of magnitude faster

than a baseline DFS algorithm that does not apply the Apriori Principle.

Further, we propose a novel formulation for the problem of generating connected graphs that are approximately frequent in a database w.r.t. the graph edit distance. This problem has been studied before (Li and Wang, 2015), but the previous problem formulation generally results in a prohibitively large number of frequent approximate subgraphs. In contrast, we prove that our formulation guarantees that the number of frequent approximate subgraphs is upper-bounded in terms of the number of frequent subgraphs. We call our problem formulation the frequent approximate subgraph mining problem.

Finally, we show that relaxing the NP-complete subgraph isomorphism problem enables algorithms for FCSM to generate a subset of the set of frequent subgraphs with polynomial delay. That is, the time between printing two consecutive frequent subgraphs, as well as the time before printing the first, is polynomial in the size of the database. All previous methods that solve relaxations of the FCSM problem with polynomial delay are restricted to generating frequent subtrees, whereas we show that it is possible to generate a subset of the set of frequent subgraphs with polynomial delay. When graph embeddings are used to train machine learning models, which are inherently probabilistic, it is arguably sufficient to use a subset of the set of frequent subgraphs as features. Welke, Horváth, and Wrobel (2018) empirically show that, on databases containing molecular graphs, even a subset of the set of frequent subtrees results in a comparable predictive performance to the complete set of frequent subgraphs.

Our contributions are as follows:

- We propose an algorithm that applies the Apriori Principle to a DFS approach for frequent subgraph mining, thereby combining the advantages of Apriori-based approaches and DFS approaches (Chapter 4). We empirically show that our algorithm is often an order of magnitude faster than a baseline DFS algorithm (Chapter 7.1).
- We propose a measure between graphs that measures how far away the structure of a graph H is to being contained in the structure of a graph G (Chapter 5.1).
- We propose a novel formulation for the problem of generating the set of frequent approximate subgraphs of a database, which guarantees that the number of frequent approximate subgraphs is upper-bounded in terms of the number of frequent subgraphs (Chapter 5.2).
- We propose a novel method for embedding domain-specific graphs into a feature space. A previous comparable method embeds domain-specific graphs into a binary feature space, while our method embeds graphs into a real-valued feature space, which results in more expressive feature vectors (Chapter 5.3).
- We show that algorithms for FCSM can be adapted to generate a subset of the set of frequent subgraphs with polynomial delay (Chapter 6), and we empirically evaluate the recall of frequent subgraphs when using efficient, but incomplete, algorithms for FCSM (Chapter 7.2).

Chapter 2

Related Work

There are two general approaches for frequent subgraph mining: Level-wise approaches, which use a breadth-first search strategy, and pattern growth approaches, which use a depth-first search strategy.
Level-wise approaches for FCSM, like the FSG method (Kuramochi and Karypis, 2001), generate all frequent subgraphs of size l before they generate any frequent subgraph of size $l+1$, where the size of a graph is usually its number of edges. Crucially, level-wise approaches exploit the Apriori Principle, which states that any graph that contains an infrequent subgraph can not be frequent. Thus, any graph G that contains an infrequent subgraph can be pruned without computing the frequency of G. To compute the frequency of a graph, the NP-complete subgraph isomorphism problem needs to be solved repeatedly, so applying the Apriori Principle significantly reduces the number of SGI computations.
gSpan (Yan and Han, 2002) is a method for FCSM that follows a pattern growth approach. gSpan assigns a canonical DFS code to each connected graph, which determines the order in which the connected graphs are visited while guaranteeing that each isomorphism class of graphs is only visited once. In contrast to gSpan, FSG visits graphs from the same isomorphism class multiple times, but only evaluates the frequency of a subgraph when its isomorphism class is visited for the first time.

REAFUM (Li and Wang, 2015) generates a subset of the set of connected graphs that are β-subgraph isomorphic to at least t graphs in D, where t is a frequency threshold. A graph H is β-subgraph isomorphic to a graph G if G contains a subgraph G' s.t. the graph edit distance between H and G' is upper-bounded by β. The number of graphs that are β-subgraph isomorphic to at least t graphs in D is generally prohibitively large, even for very small values of β, which is why the REAFUM method focuses on reducing the number of graphs in its output. In the first step, REAFUM determines a subset of the database D that contains graphs that best represent the graphs in D. In the second step, REAFUM generates approximate frequent subgraphs of D, i.e. connected subgraphs of representative graphs that are β-subgraph isomorphic to at least t graphs in D. In the final step, REAFUM further reduces the size of its output by determining a representative subset of the set of generated graphs.

FSG, gSpan and REAFUM generate their output with exponential delay. That is, the time between printing two consecutive patterns, as well as the time before printing the first pattern, is exponential in the size of the database. Since there can be no algorithm that correctly solves the FCSM problem with polynomial delay (Horváth, Bringmann, and Raedt, 2006), any efficient algorithm for mining frequent subgraphs can only solve a relaxation of the FCSM problem. All previous methods that solve relaxations of the FCSM problem with polynomial delay restrict themselves to generating frequent subtrees.
The PS method (Welke, Horváth, and Wrobel, 2018) generates a random subset of the set of frequent subtrees with polynomial delay. PS replaces each graph G in the database by a forest T that contains spanning trees of G. Since the number of spanning trees of a graph G is generally exponential, T only contains a polynomial number of the spanning trees of G, so that the size of T is polynomial in the size of G. PS exploits the fact that the frequent subtrees of a database of forests can be generated with polynomial delay (Chi, Yang, and Muntz, 2003).

Chapter 3

Preliminaries

In this chapter, we present definitions and statements that will be used throughout this thesis.

3.1 Graph Morphisms

Graph morphisms are maps between graphs that preserve the partial or complete structures of the graphs. They are used to compare two graphs for structural equality (graph isomorphism) or to determine whether the structure of one graph is contained in the structure of another graph (subgraph isomorphism). Before discussing graph morphisms, we need to state some basic definitions from graph theory:

Definition 3.1 (Labeled Undirected Graph).
Let $V(G)$ be a finite set of vertices and let $E(G) \subseteq \{S \subseteq V(G) \mid |S| = 2\}$ be a set of edges. For a finite label alphabet Σ, let

$$\lambda^G \colon V(G) \cup E(G) \to \Sigma$$

be a labeling map. The triple $G \coloneqq (V(G), E(G), \lambda^G)$ is called a labeled undirected graph. We say that G has size $|E(G)|$ and order $|V(G)|$. We denote the empty graph $(\emptyset, \emptyset, \emptyset)$ by K_0, and the set of all labeled undirected graphs by $\mathcal{G}$. ◇

We often call the elements of $\mathcal{G}$ labeled graphs, or graphs, instead of labeled undirected graphs. In this thesis, we do not consider any graphs that are not in $\mathcal{G}$.

Definition 3.2 (Subgraph).
A labeled graph H is a subgraph of a labeled graph G if the following two conditions are satisfied:

(i) $\forall v \in V(H) \colon v \in V(G) \wedge \lambda^H(v) = \lambda^G(v)$.
(ii) $\forall e \in E(H) \colon e \in E(G) \wedge \lambda^H(e) = \lambda^G(e)$.

If H is a subgraph of G, we say that G contains H. ◇

Definition 3.3 (Connectivity).
A labeled graph G is connected if, for all vertices $u, v \in V(G)$, G contains a path with endpoints u and v. We denote the set of labeled connected graphs by $\mathcal{G}_{\text{con}}$. ◇

Note that by Definition 3.3, the empty graph K_0 is connected.

Definition 3.4 (Graph Homomorphism).
Let H and G be labeled graphs. A map $\varphi\colon V(H) \to V(G)$ is a graph homomorphism if φ is edge-preserving and label-preserving. That is,

(i) $\forall\{u, v\} \in E(H)\colon \{\varphi(u), \varphi(v)\} \in E(G)$,
(ii) $\forall u \in V(H)\colon \lambda^H(u) = \lambda^G(\varphi(u))$,
(iii) $\forall\{u, v\} \in E(H)\colon \lambda^H(\{u, v\}) = \lambda^G(\{\varphi(u), \varphi(v)\})$.

The problem of deciding whether a graph homomorphism between H and G exists is called the graph homomorphism problem. ◇

Proposition 3.5.
The graph homomorphism problem is NP-complete. ◇

Definition 3.6 (Graph Isomorphism).
Let H and G be labeled graphs. A map $\varphi\colon V(H) \to V(G)$ is a graph isomorphism if

(i) φ is a bijection,
(ii) φ and φ^{-1} are graph homomorphisms.

Graph isomorphisms define an equivalence relation $\cong$ on $\mathcal{G}$ as follows:

$$H \cong G \text{ if there is a graph isomorphism } \varphi\colon V(H) \to V(G).$$

If $H \cong G$, we call H and G isomorphic. We denote the isomorphism class of a graph H by $[H]$, and the set of all isomorphism classes by $\mathcal{G}/\!\cong$. The problem of deciding whether two given graphs are isomorphic is called the graph isomorphism problem (GI). ◇

The GI problem is in NP, but it is unknown whether GI is in P, NP-complete, neither in P nor NP-complete, or both in P and NP-complete. Resolving the complexity of GI is widely regarded to be one of the most important open problems in theoretical computer science.

Given a set of graphs. A naive algorithm that removes all redundant graphs from this set, i.e. all graphs that are isomorphic to another graph in the set, requires a quadratic number of GI computations. A more efficient solution for this problem is to assign a hash value to each graph in the set, and to compare the hashes in order to identify redundant graphs. However, this requires the hash function to be collision-free. Before continuing, let us formally define hash function for graphs.

Definition 3.7 (Graph Hash Function).
Let S be a set, and let $\mathcal{G}' \subseteq \mathcal{G}$. A map $f\colon \mathcal{G}' \to S$ is a graph invariant if

$$\forall G_1, G_2 \in \mathcal{G}'\colon G_1 \cong G_2 \implies f(G_1) = f(G_2).$$

A graph invariant $h\colon \mathcal{G}' \to \Sigma^*$ is a graph hash function if Σ is a finite label alphabet. $h\colon \mathcal{G}' \to \Sigma^*$ is a perfect graph hash function if

$$\forall G_1, G_2 \in \mathcal{G}'\colon G_1 \cong G_2 \iff h(G_1) = h(G_2). \qquad \diamond$$

A graph hash function can generally produce hash collisions, i.e. there can be two non-isomorphic graphs that get mapped to the same hash value. A well-known example for a polynomial-time graph hash function that produces hash collisions is the Weisfeiler-Leman method (Weisfeiler and Leman, 1968). A perfect graph hash function, i.e. a graph hash function that never produces hash collisions, can be used to solve the GI problem, so computing a perfect graph hash function is at least as hard as solving GI.

We often need to store a large number of graph hashes in a set data structure, which is a data structure that supports two operations called insert and search. The insert operation inserts an element into the set, and the search operation determines whether an element is contained in the set. A naive implementation of the search operation has an execution time that grows linearly in the size of the set. However, when using a prefix tree (De La Briandais, 1959) as the set data structure, the execution time of the insert and search operations is only linear in the size of the element that is searched for or inserted, and does not depend on the size of the set.

We now turn our attention to the problem of determining whether the structure of a graph is contained in the structure of another graph.

Definition 3.8 (Subgraph Isomorphism).
Let H and G be labeled graphs. A map $\varphi\colon V(H) \to V(G)$ is a subgraph isomorphism if it is an injective graph homomorphism. If a subgraph isomorphism $\varphi\colon V(H) \to V(G)$ exists, we call H subgraph isomorphic to G, and write $H \preceq G$. The problem of deciding whether a graph H is subgraph isomorphic to a graph G is called the SGI problem. $\diamond$

Proposition 3.9.
The SGI problem is NP-complete. $\diamond$

Proposition 3.10.
Let H and G be labeled graphs. Then H is subgraph isomorphic to G iff G contains a subgraph that is isomorphic to H. $\diamond$

Proposition 3.11.
$\preceq$ is a partial order on $\mathcal{G}/\cong$. $\diamond$

Since $\preceq$ is a partial order, we can use it to define monotone graph invariants.

Definition 3.12 (Monotonicity).
Let $(S_1, \leq_1)$ and $(S_2, \leq_2)$ be two partially ordered sets. A map $f\colon S_1 \to S_2$ is monotone if

$$\forall x, y \in S_1\colon x \leq_1 y \implies f(x) \leq_2 f(y). \qquad \diamond$$

For a graph property $p\colon \mathcal{G}' \to \{\text{true, false}\}$, i.e. a graph invariant that maps to $\{\text{true, false}\}$, we set true $\leq$ false. That is, a graph property p is monotone if for any graph G that has property p, all graphs $H \in \mathcal{G}'$ with $H \preceq G$ also have property p. In the pattern mining literature, many authors set true $\geq$ false, so monotone graph properties may be called anti-monotone by other authors.

3.2 Frequent Connected Subgraph Mining

In this section we formally define the frequent connected subgraph mining problem. But first, we define the more general theory extraction problem.

Definition 3.13 (Theory Extraction Problem, Mannila and Toivonen, 1997).
Given

(i) a pattern language $\mathcal{L}$, i.e. a set of patterns,
(ii) a transaction database D, i.e. a multiset of transactions that are indexed by a unique transaction ID,
(iii) and an interestingness predicate $q_D\colon \mathcal{L} \to \{\text{true, false}\}$ that determines whether a pattern $\varphi \in \mathcal{L}$ is "interesting":

$$q_D(\varphi) \coloneqq \text{true if } \varphi \in \mathcal{L} \text{ is interesting w.r.t. } D.$$

We call the set of all interesting patterns the theory of D, and denote it by $Th(\mathcal{L}, D, q_D)$. The problem of generating the theory of D is called the theory extraction problem. ◇

Definition 3.14 (Frequent Connected Subgraph Mining).
Given

(i) the pattern language $\mathcal{L} \coloneqq \mathcal{G}_{\text{con}}$,
(ii) a finite non-empty transaction database D of labeled graphs,
(iii) a frequency threshold $t \in [|D|]$, where we use the notation $[n] \coloneqq \{1, \ldots, n\}$ for $n \in \mathbb{N} \coloneqq \{1, 2, \ldots\}$,
(iv) and an interestingness predicate $q_D^t\colon \mathcal{L} \to \{\text{true, false}\}$ with

$$q_D^t(P) \coloneqq \text{true if } P \text{ is subgraph isomorphic to at least } t \text{ graphs in } D.$$

The problem of generating exactly one representative for each isomorphism class in $Th(\mathcal{L}, D, q_D^t)/\cong$ is called the frequent connected subgraph mining problem (FCSM). We sometimes write q_D instead of q_D^t. ◇

Many algorithms for pattern mining require the interestingness predicate to be monotone. Fortunately, q_D^t satisfies this requirement.

Proposition 3.15.
q_D^t is a monotone graph invariant. ◇

In order to analyze the time complexity of algorithms for FCSM, we introduce the notion of an enumeration problem, which is a class of computational problems that includes the FCSM problem.

Definition 3.16 (Enumeration Problem).
Let X be a set of instances. For each $x \in X$, let $O(x)$ be the set of valid solutions for x. Let each $o \in O(x)$ be a finite set. Then $\mathcal{P} \coloneqq \{(x, O(x)) \mid x \in X\}$ is called an enumeration problem. The enumeration problem $\mathcal{P}$ is polynomially balanced if there is a polynomial p s.t.

$$\forall (x, y) \in R\colon size(y) \leq p(size(x)),$$

where

$$R \coloneqq \{(x, y) \mid x \in X, o \in O(x), y \in o\},$$

and where $size(x)$ is the length of a binary representation of x. An algorithm $\mathcal{A}$ solves the enumeration problem $\mathcal{P}$ correctly if, given any instance $x \in X$, there is a valid solution $o \in O(x)$ s.t.

(i) $\mathcal{A}$ does not output anything that is not an element of o. (soundness)
(ii) $\mathcal{A}$ outputs every element of o. (completeness)
(iii) $\mathcal{A}$ does not output any element more than once. (irredundancy)

$\mathcal{A}$ may generate the elements of o in an arbitrary order. An algorithm for an enumeration problem that is correct and terminates in finite time is totally correct. In this thesis, when we talk about the correctness of an algorithm, we mean total correctness. ◇

Proposition 3.17.
The FCSM problem is a polynomially balanced enumeration problem. ◇

For an enumeration problem $\mathcal{P} = \{(x, O(x)) \mid x \in X\}$, the cardinality of a valid solution $o \in O(x)$ may be exponential in the size of x. If there is a valid solution for an instance x that is exponential in $size(x)$, then no polynomial-time algorithm for the enumeration problem can exist. In Definition 3.18, we define complexity measures for enumeration problems (Johnson, Yannakakis, and Papadimitriou, 1988) that can be used to distinguish efficient from inefficient algorithms, even when no polynomial-time algorithm exists.

Definition 3.18 (Output-Sensitive Complexity Measures).
An algorithm generates its output with polynomial delay if, for any valid input x, the algorithm generates a list L of elements s.t.

(i) The time before printing the first element of L is polynomial in $size(x)$.
(ii) The time between printing two consecutive elements of L is polynomial in $size(x)$.
(iii) The time after printing the last element of L is polynomial in $size(x)$.

An algorithm generates its output in output polynomial time if, for any valid input x, the algorithm generates a list L and terminates in time polynomial in $size(x) + size(L)$.◇

Proposition 3.19 (Horváth, Bringmann, and Raedt, 2006).
The FCSM problem cannot be solved in output polynomial time, unless P=NP. ◇

3.3 Algorithms for FCSM

Algorithms for pattern mining make use of refinement operators in order to systematically search for interesting patterns in the pattern space.

Definition 3.20 (Downward Refinement Operator).
Let $\mathcal{H}$ be a set (usually a hypothesis space), and let $\preceq$ be a partial order on $\mathcal{H}$ (usually a generality relation, i.e. $h' \preceq h$ if h' is more general than h). A map $\rho\colon \mathcal{H} \to 2^{\mathcal{H}}$ is a downward refinement operator if

$$\forall h, h' \in \mathcal{H}\colon h \in \rho(h') \implies h' \preceq h.$$

Suppose $\mathcal{H}$ contains a smallest element $\bot$. A refinement operator ρ is complete if, for all $h \in \mathcal{H}$, there is a finite chain

$$\bot = h_0 \preceq h_1 \preceq \ldots \preceq h_k = h$$

with $\rho(h_{i-1}) = h_i$ for $i \in [k]$. ρ is optimal if there is exactly one such finite chain for each $h \in \mathcal{H}$. ◇

We will now define a refinement operator that can be used to systematically traverse the pattern space $\mathcal{G}_{\text{con}}$.

Definition 3.21.
Let $\mathcal{L} \coloneqq \mathcal{G}_{\text{con}}$. We define a refinement operator $\rho\colon \mathcal{L} \to 2^{\mathcal{L}}$ as follows:

(i) $\rho(K_0)$ is the set of labeled graphs of order 1.
(ii) For $P \in \mathcal{L} \setminus \{K_0\}$, $\rho(P)$ is the set of labeled graphs that can be constructed from P by adding exactly one labeled edge to P that must be incident to at least one vertex in $V(P)$.

ρ is complete, but not optimal. For each pattern $H \in \mathcal{L} \setminus \{K_0\}$, we define

$$\rho^{-1}(H) \coloneqq \{P \in \mathcal{L} \mid H \in \rho(P)\}.$$ ◇

When searching for interesting patterns in a pattern space with monotone interestingness predicate, one usually starts from the smallest element $\bot$ (if it exists), repeatedly applies a downward refinement operator ρ, and evaluates the interestingness predicate on the encountered candidate patterns. Since q_D is monotone, $q_D(P) = \text{false}$ implies $q_D(H) = \text{false}$ for any pattern H with $P \preceq H$. So, once a pattern P with $q_D(P) = \text{false}$ is encountered, the search space can be pruned. A related technique for reducing the number of evaluations of q_D is the Apriori Principle.

Definition 3.22 (Apriori Principle, Agrawal and Srikant, 1994).
Let $\mathcal{L}$ be a pattern language and let $q_D\colon \mathcal{L} \to \{\text{true}, \text{false}\}$ be a monotone interestingness predicate. A pattern $P \in \mathcal{L}$ is a strong candidate if all patterns $P' \prec P$ are interesting. Any pattern that is not a strong candidate is a weak candidate. Since the interestingness predicate q_D is monotone, every interesting pattern is a strong candidate. The concept of evaluating $q_D(P)$ only if P is a strong candidate, and pruning weak candidates without evaluating q_D, is called the Apriori Principle. ◇

Algorithm 1 Generic Level-wise Search Algorithm for FCSM

Input: A finite non-empty database D of labeled graphs, a threshold $t \in [|D|]$.
Output: One representative of every isomorphism class in $Th(\mathcal{L}, D, q_D^t)/\cong$.

```
1:  F_{-1} = {K_0}
2:  for (l = 0; F_{l-1} ≠ ∅; l += 1) do
3:      F_l = ∅                                  ▷ Interesting patterns of size l
4:      C_l = ∅                                  ▷ Hashes of strong candidates of size l
5:      for all P ∈ F_{l-1} do
6:          print P
7:          for all H ∈ ρ(P) with h(H) ∉ C_l and ρ^{-1}(H) ⊆ F_{l-1} do
8:              C_l = C_l ∪ {h(H)}
9:              if q_D^t(H) = true then
10:                 F_l = F_l ∪ {H}
```

The Apriori Principle gets applied in Algorithm 1 (Mannila and Toivonen, 1997, Horváth and Ramon, 2010), which is a generic level-wise algorithm for FCSM, i.e. an algorithm for FCSM that uses a breadth-first search strategy.

Proposition 3.23.
Let h be a perfect graph hash function. Then Algorithm 1 solves the FCSM problem correctly. ◇

A simple, but effective, method for reducing the execution time of algorithms for FCSM is to only compute $\mathrm{SGI}(H, G)$ for graphs G in the support of a subgraph of H.

Definition 3.24 (Support).
Let $(\mathcal{L}, \preceq)$ be a partially ordered pattern language and let D be a transaction database. For any pattern $P \in \mathcal{L}$, the set

$$S_P := \{G \in D \mid P \preceq G\}$$

is called the support of P. ◇

When evaluating $q_D(H)$ for $H \in \rho(P)$, $\mathrm{SGI}(H, G)$ needs only be computed for transactions G in the support of P. So, in order to reduce the number of SGI computations in Algorithm 1, the support of every pattern $P \in \mathcal{F}_{l-1} \cup \mathcal{F}_l$ can be stored in memory as a list of transaction IDs.

A different approach for solving the FCSM problem are pattern growth approaches, which are algorithms that follow a depth-first search strategy. Algorithm 2 (Yan and Han, 2003) shows a generic pattern growth approach for FCSM.

Proposition 3.25.
Let h be a perfect graph hash function. Then Algorithm 2 solves the FCSM problem correctly. ◇

Algorithm 2 Generic Pattern Growth Algorithm for FCSM

Input: A finite non-empty database D of labeled graphs, a threshold $t \in [|D|]$.
Output: One representative of every isomorphism class in $Th(\mathcal{L}, D, q_D^t)/\cong$.

```
1: Declare a set F with global scope
2: procedure Main
3:     F = {h(K_0)}                          ▷ Hashes of interesting patterns
4:     print K_0
5:     PatternGrowth(K_0)
6: procedure PatternGrowth(P)
7:     for all H ∈ ρ(P) with h(H) ∉ F do
8:         if q_D^t(H) = true then
9:             F = F ∪ {h(H)}
10:            print H
11:            PatternGrowth(H)
```

Suppose all possible embeddings from a graph $P \in \rho^{-1}(H)$ into a graph G have already been computed and are stored in memory. If the number of embeddings is not too large, then $\mathrm{SGI}(H, G)$ can be computed efficiently using these stored embeddings, because any subgraph isomorphism between H and G is an extension of at least one of the stored embeddings. This method for computing SGI goes back to Ullmann (1976). By storing, for each pattern $P_0 = K_0, P_1, \ldots, P_l$ with $P_i \in \rho(P_{i-1})$, the lists of embeddings from P_i into all graphs in the support of P_i, the SGI computations in Algorithm 2 can be performed efficiently, assuming the number of stored embeddings does not become too large. Unfortunately, this assumption can only be made if the transaction database only contains "simple" graphs. For example, it has been shown that FCSM methods that use the method by Ullmann (1976) work well on databases containing molecular graphs (Nijssen and Kok, 2005). However, the number of embeddings from a graph P into a graph G is generally exponential in $size(P)$, so the method by Ullmann (1976) generally requires a prohibitive amount of storage space.

3.4 The Subgraph Feature Space

The frequent subgraphs can be used to embed graphs into a domain-specific feature space.

Definition 3.26 (Subgraph Feature Space).
Let $\mathcal{F}$ be a set of feature graphs, for example a set of frequent connected subgraphs of a database D. Let G be a labeled graph. For each $H \in \mathcal{F}$, let

$$\Phi_H(G) := \begin{cases} 1 & \text{if } H \preceq G \\ 0 & \text{o/w} \end{cases}$$

be the value of G w.r.t. feature H. Then

$$\Psi(G) := (\Phi_H(G))_{H \in \mathcal{F}}.$$

is the subgraph feature vector of G w.r.t. $\mathcal{F}$. ◇

Suppose $\mathcal{F}$ is the set of frequent connected subgraphs of a sufficiently large graph database D, and suppose the frequency threshold t has been chosen s.t. $\mathcal{F}$ contains a sufficient number of graphs. Then the distances between subgraph feature vectors (e.g. L^1-distance, Jaccard distance, ...), as well as the standard inner product, can arguably be taken as measures for the similarity between graphs that are sampled from the same distribution as the graphs in D. Because of this, any machine learning method that accepts vector-valued data as input, including methods that make use of distances or inner products in the feature space (e.g. clustering methods, kNN, SVM, ...), can be applied to the subgraph feature vectors.

3.5 The Graph Edit Distance

The graph edit distance (GED) is a measure for the dissimilarity between two labeled graphs. Two graphs H and G are interpreted to be dissimilar w.r.t. GED if, for any sequence of edit operations that transforms H into G, the cost incurred by the sequence of edit operations is large. The GED was first introduced by Sanfeliu and Fu (1983). We present the definition given by Bougleux et al. (2017) which, contrary to the original definition, is based on restricted edit paths.

Definition 3.27 (Graph Edit Distance).
Let H and G be labeled graphs, let Σ be a finite label alphabet, and let ε be a blank symbol, i.e. a symbol that is not an element of Σ. Denoting $\Sigma \cup \{\varepsilon\}$ by Σ_ε, we call a function

$$c\colon \Sigma_\varepsilon \times \Sigma_\varepsilon \to [0, \infty)$$

an edit cost function if

$$\forall \alpha \in \Sigma_\varepsilon\colon c(\alpha, \alpha) = 0.$$

An edit cost function assigns an edit cost to each edit operation. Table 3.1 contains a comprehensive list of all considered edit operations and their associated edit costs. An edit path π between H and G is a finite sequence of edit operations $(o_i)_{i=1}^k$ that transforms H into a graph $\pi(H)$ that is isomorphic to G. The cost incurred by π is defined as

$$c(\pi) \coloneqq \sum_{i=1}^{k} c(o_i),$$

where $c(o_i)$ denotes the edit cost of the edit operation o_i. A restricted edit path π between H and G is an edit path s.t. each vertex and each edge is edited (inserted, deleted or substituted) at most once. We denote the set of restricted edit paths between H and G by $\Pi(H, G)$, and define the graph edit distance between H and G as follows:

$$\mathrm{GED}(H, G) \coloneqq \min_{\pi \in \Pi(H,G)} c(\pi).$$

The problem of computing the graph edit distance is called the GED problem. ◇

Edit operation	Edit cost
Insert an isolated vertex with label $\alpha \in \Sigma$	$c(\varepsilon, \alpha)$
Delete an isolated vertex u	$c(\lambda(u), \varepsilon)$
Substitute the label of a vertex u by $\alpha \in \Sigma$	$c(\lambda(u), \alpha)$
Insert an edge with label $\alpha \in \Sigma$	$c(\varepsilon, \alpha)$
Delete an edge e	$c(\lambda(e), \varepsilon)$
Substitute the label of an edge e by $\alpha \in \Sigma$	$c(\lambda(e), \alpha)$

Table 3.1: Edit operations and their associated edit costs. Deleting an edge $\{u, v\}$ does not delete u or v, and inserting an edge $\{u, v\}$ is only possible if u and v have been previously inserted or are vertices of H.

For certain choices of edit cost functions, the graph edit distance is a distance function. The following Proposition tells us which condition the edit cost function must satisfy in order for the graph edit distance to satisfy the triangle inequality.

Proposition 3.28 (Triangle Inequality).
Let c be an edit cost function that satisfies the triangle inequality, i.e.

$$\forall \alpha, \beta, \beta' \in \Sigma_\varepsilon \colon c(\alpha, \beta) \leq c(\alpha, \beta') + c(\beta', \beta).$$

Then the graph edit distance fulfills the triangle inequality as well, i.e.

$$\forall H, G, G' \in \mathcal{G} \colon \mathrm{GED}(H, G) \leq \mathrm{GED}(H, G') + \mathrm{GED}(G', G). \qquad \diamond$$

We finish this section by stating a negative result concerning the complexity of GED.

Definition 3.29.
Given two labeled graphs H and G, and a threshold $\tau \in [0, \infty)$. The decision problem associated to the GED problem is to decide whether $\mathrm{GED}(H, G)$ is upper-bounded by τ. $\diamond$

Proposition 3.30 (Zeng et al., 2009).
The decision problem associated to GED is NP-complete, even if each non-trivial edit operation has an edit cost of 1, i.e. even if $\forall \alpha, \beta \in \Sigma_\varepsilon \colon \alpha \neq \beta \iff c(\alpha, \beta) = 1$. $\diamond$

3.6 Heuristic Algorithms

We now formally specify what we mean when we talk about heuristic algorithms.

Definition 3.31 (Heuristic Algorithm).
A heuristic algorithm is an algorithm that is not required to yield correct results. A heuristic algorithm $\mathcal{A}$ for a decision problem is a true-biased (false-biased) algorithm if $\mathcal{A}$ is always correct when it returns true (false). $\diamond$

A common use case for heuristic algorithms are computational problems for which no correct algorithm with acceptable execution time is known. In such cases, it may be

preferable to use a heuristic algorithm that only approximately solves the problem, but has an acceptable execution time.
Ideally, a heuristic algorithm should come with a provable performance guarantee. An example for a class of heuristic algorithms that come with provable performance guarantees are α-approximation algorithms.

Definition 3.32 (α-Approximation Algorithm).
Let OPT be an optimization problem with a non-negative objective, and let $\alpha \geq 1$. A polynomial-time heuristic algorithm $\mathcal{A}$ for OPT is an α-approximation algorithm if, for all instances x,

$$\alpha^{-1} \cdot \mathrm{OPT}(x) \leq \mathcal{A}(x) \leq \alpha \cdot \mathrm{OPT}(x).$$

Here, $\mathrm{OPT}(x)$ denotes the optimum value on instance x, and $\mathcal{A}(x)$ denotes the value returned by $\mathcal{A}$ on instance x. ◇

For the GED problem, however, we have the following negative result:

Proposition 3.33 (Blumenthal, Gamper, et al., 2021).
There is no α-approximation algorithm for GED for any α, unless GI is in P. ◇

Proposition 3.33 is true even when each non-trivial edit operation has a cost of 1.

Chapter 4

The Apriori Pattern Growth Algorithm

We propose the Apriori Pattern Growth Algorithm, which is a novel pattern growth algorithm for FCSM that exploits the Apriori Principle. Our algorithm significantly reduces the number of evaluations of the interestingness predicate q_D when compared to previous pattern growth approaches. In Chapter 7.1, we empirically show that our algorithm is often an order of magnitude faster than a pattern growth algorithm that does not apply the Apriori Principle.

The main advantage of level-wise approaches for FCSM is that they exploit the Apriori Principle, which means they only evaluate q_D on strong candidates. The main disadvantage of level-wise approaches is that they need to store $\mathcal{F}_{l-1} \cup \mathcal{F}_l$ in memory, i.e. the frequent subgraphs of both the current level l and the previous level $l-1$, because the frequent subgraphs of size $l-1$ are needed to generate candidate graphs of size l. Moreover, the support of each frequent subgraph in $\mathcal{F}_{l-1} \cup \mathcal{F}_l$ needs to be stored in memory. Even worse, if the method by Ullmann (1976) for computing SGI is used, the list of embeddings between P and every transaction graph in the support of P needs to be stored in memory, for every pattern $P \in \mathcal{F}_{l-1} \cup \mathcal{F}_l$.

Pattern growth approaches, on the other hand, use a depth-first search strategy, and only need to store a single chain $P_0 = K_0, P_1, \ldots P_l$ of patterns in memory, where $P_i \in \rho(P_{i-1})$ for $i \in [l]$. As a consequence, pattern growth approaches only need to store the support sets of l patterns. If the method by Ullmann (1976) is used, the embedding lists between P_i and every transaction graph in the support of P_i need to be stored in memory, for $i \in [l]$. Regardless of whether or not the method by Ullmann (1976) is used, pattern growth approaches require significantly less memory than level-wise approaches, which is why pattern growth approaches are more popular than level-wise approaches. The main disadvantage of previous pattern growth approaches is that they do not apply the Apriori Principle, and need to evaluate the interestingness predicate q_D on all patterns P for which at least one pattern $P' \in \rho^{-1}(P)$ exists with $q_D(P') = \text{true}$.

Algorithm 3 Apriori Pattern Growth Algorithm

Input: A finite non-empty database D of labeled graphs, a threshold $t \in [|D|]$.
Output: One representative of every isomorphism class in $Th(\mathcal{L}, D, q_D^t)/\cong$.

```
1: Declare two sets C and F with global scope
2: procedure Main
3:     C = {h(K_0)}                          ▷ Hashes of strong candidates
4:     F = {h(K_0)}                          ▷ Hashes of interesting patterns
5:     print K_0
6:     PatternGrowth(K_0)
7: procedure PatternGrowth(P)
8:     for all H ∈ ρ(P) with h(H) ∉ C and {h(H') | H' ∈ ρ^{-1}(H)} ⊆ F do
9:         C = C ∪ {h(H)}
10:        if q_D^t(H) = true then
11:            F = F ∪ {h(H)}
12:            print H
13:            PatternGrowth(H)
```

Pattern growth approaches, like gSpan (Yan and Han, 2002), focus on reducing the number of graph isomorphism computations, while the Apriori Principle reduces the number of subgraph isomorphism computations. However, not only are algorithms for GI generally significantly faster than algorithms for SGI, but the input to GI in algorithms for FCSM are pattern graphs, whereas the input to SGI are a pattern graph and a transaction graph, with the latter generally having a much larger size than the former. So we argue that reducing the number of SGI computations is more important than reducing the number of GI computations of an FCSM method. If the interestingness predicate q_D of a pattern mining problem has a larger execution time than the interestingness predicate of the FCSM problem, which is the case for the interestingness predicate of the FASM problem (Chapter 5.2), then the execution time of GI is even less significant in comparison to the execution time of q_D.

Our proposed Apriori Pattern Growth Algorithm (Algorithm 3) is a pattern growth algorithm for FCSM that exploits the Apriori Principle. That is, our algorithm only evaluates q_D on strong candidates, i.e. patterns whose proper subgraphs are interesting, thereby significantly reducing the number of SGI computations when compared to previous pattern growth approaches. When a pattern growth algorithm visits a pattern P for the first time, it is generally unknown whether the proper subgraphs of P are interesting, because not all subgraphs of P have been visited at this point. So, when P is visited for the first time, it is generally unknown whether P is a strong candidate or not. In contrast to previous pattern growth approaches, Algorithm 3 does generally not process a strong candidate when it is visited for the first time. Instead, a strong candidate is only processed after all of its proper subgraphs have already been processed, which enables the application of the Apriori Principle.

Proposition 4.1.
Let h be a perfect graph hash function. Then Algorithm 3 solves the FCSM problem correctly. ◇

Proof. Soundness: A pattern H is printed only if $q_D(H) = \text{true}$. This also holds for the empty graph K_0, as $q_D(K_0) = \text{true}$ for $t \in [|D|]$.

Completeness: Assume $H \in \mathcal{L} \setminus \{K_0\}$ is interesting and $h(H)$ is never inserted into C. W.l.o.g. H is a minimal (w.r.t. $\preceq$) pattern in the set

$$\{H \in \mathcal{L} \setminus \{K_0\} \mid q_D(H) = \text{ true } \wedge h(H) \notin C\}.$$

By monotonicity of q_D, all patterns $H' \in \rho^{-1}(H)$ are interesting, and so, by minimality of H, $h(H') \in C$ for all $H' \in \rho^{-1}(H)$.
Let $H' \in \rho^{-1}(H)$ be s.t. $h(H')$ gets inserted into C after all other hashes of patterns in $\rho^{-1}(H)$ are already elements of C. W.l.o.g. H' is the representative of $[H']$ that gets processed in the body of the for-loop.
Then $H \in \rho(H')$ is generated as a candidate, and H meets the conditions of the for-loop, i.e. $h(H) \notin C$ (by assumption), and $\{h(H') \mid H' \in \rho^{-1}(H)\} \subseteq F$. Thus, $h(H)$ gets added to C, contradicting the assumption. So the hash of every interesting pattern gets inserted into C, and thus a representative of every interesting pattern is printed.

Irredundancy: Assume graphs H_1 and H_2 with $H_1 \cong H_2$ get printed. W.l.o.g. H_1 is printed before H_2. Since h is a graph invariant, we have $h(H_1) = h(H_2)$. Thus, when H_2 is processed by the for-loop, $h(H_2)$ is already an element of C, so H_2 gets skipped by the for-loop and is never printed, contradicting the assumption.

Termination: Since D is finite, $Th(\mathcal{L}, D, q_D)$ is finite as well. So the algorithm soundly and irredundantly generates a finite set of patterns, which implies termination. ■

A possible disadvantage of Algorithm 3 is the need to store the set C of hashes of strong candidates, while Algorithm 2 only stores the smaller set F of hashes of interesting patterns. On the other hand, Algorithm 3 only evaluates q_D at most once for every isomorphism class, while Algorithm 2 evaluates q_D on graphs that are isomorphic to each other. In order to avoid having to evaluate q_D on isomorphic graphs, Algorithm 2 can store the set of hashes of candidates, which is even larger than C, and test, for every new candidate, whether its interestingness has already been evaluated. Alternatively, Algorithm 2 can use the candidate generation technique of gSpan (Yan and Han, 2002), which guarantees that no two isomorphic graphs are generated, without requiring any additional storage.

Chapter 5

Frequent Approximate Subgraph Mining

In this chapter, we propose a novel formulation for the problem of generating connected graphs that are approximately frequent in a graph database w.r.t. the graph edit distance. To motivate the study of mining frequent approximate subgraphs, consider a graph G that has the following two properties:

(i) G is not a frequent subgraph of D.
(ii) Given a threshold $\tau > 0$, there are graphs $G_1, G_2, \ldots, G_k$ with $\text{GED}(G, G_i) \leq \tau$ for $i \in [k]$, which are subgraph isomorphic to transaction graphs in D, but which are not frequent subgraphs of D.

G is a frequent approximate subgraph if the union of the support sets of the patterns G_i exceeds the frequency threshold t. In other words, the frequent approximate subgraph mining problem uncovers patterns that need not be frequent themselves, but that have several variations in the database whose total support exceeds the frequency threshold.

Li and Wang (2015) have previously studied the problem of generating frequent approximate subgraphs, but their problem formulation generally results in a prohibitively large number of frequent approximate subgraphs. For our problem formulation, which we present in Section 5.2, we prove that there is an upper bound on the number of frequent approximate subgraphs in terms of the number of frequent subgraphs. That is, when using our problem formulation, the number of frequent approximate subgraphs is not larger than the number of frequent subgraph for a smaller frequency threshold.

Our formulation of the frequent approximate subgraph mining problem (FASM) uses a novel measure between graphs that measures how close a graph H is to being subgraph isomorphic to a graph G. We call this measure the subgraph edit distance (SGED) and study its properties in Section 5.1. In Section 5.3, we propose a novel method for embedding domain-specific graphs into a feature space. We call this feature space the SGED feature space, because feature vectors in this feature space are computed using the subgraph edit distance. The SGED feature space is comparable to the subgraph feature space (Section 3.4), except that subgraph feature vectors are binary vectors, while

SGED feature vectors are real-valued vectors, which makes them more expressive.

5.1 The Subgraph Edit Distance

In this section, we propose a measure between graphs which we call the subgraph edit distance (SGED). The SGED can be used to answer the question whether the structure of a graph H is approximately (w.r.t. GED) contained in the structure of a graph G.

Definition 5.1 (Subgraph Edit Distance).
Let H and G be labeled graphs, let Σ be a finite label alphabet and let ε be a blank symbol, i.e. a symbol that is not an element of Σ. Denoting $\Sigma \cup \{\varepsilon\}$ by Σ_ε, let

$$c\colon \Sigma_\varepsilon \times \Sigma_\varepsilon \to [0, \infty)$$

be an edit cost function that satisfies the following four constraints:

$$\forall \beta \in \Sigma_\varepsilon\colon c(\varepsilon, \beta) = 0 \qquad \text{(free insertions)} \tag{5.1}$$

$$\forall \alpha \in \Sigma\colon c(\alpha, \varepsilon) > 0 \qquad \text{(paid deletions)} \tag{5.2}$$

$$\forall \alpha, \beta \in \Sigma\colon \alpha \neq \beta \iff c(\alpha, \beta) > 0 \qquad \text{(paid substitutions)} \tag{5.3}$$

$$\forall \alpha, \beta, \beta' \in \Sigma_\varepsilon\colon c(\alpha, \beta) \leq c(\alpha, \beta') + c(\beta', \beta) \qquad \text{(triangle inequality)} \tag{5.4}$$

As in Definition 3.27, we denote the set of restricted edit paths between H and G by $\Pi(H, G)$, and the cost incurred by an edit path π by $c(\pi)$. The subgraph edit distance between H and G is defined as follows:

$$\mathrm{SGED}(H, G) \coloneqq \min_{\pi \in \Pi(H,G)} c(\pi).$$

The problem of computing the subgraph edit distance is called the SGED problem. ◇

We remark that the subgraph edit distance is a special case of the graph edit distance, with additional restrictions placed on the edit cost function. As an example for Definition 5.1, consider the edit cost function where each deletion and each non-trivial substitution has a cost of 1, and each insertion has a cost of 0. This edit cost function satisfies all four constraints in Definition 5.1, which shows that SGED is not ill-defined.

We intentionally defined the subgraph edit distance s.t. the following proposition is true.

Proposition 5.2.
Let H and G be labeled graphs. Then the following equivalency holds:

$$H \preceq G \iff \mathrm{SGED}(H, G) = 0.$$

As a consequence, the decision problem associated to SGED generalizes the SGI problem. ◇

Proof. Suppose $H \preceq G$, and let H' be a subgraph of G s.t. $H \cong H'$. Consider the edit path π between H and G that, for each $v \in V(G) \setminus V(H')$, inserts a vertex v' with label $\lambda^G(v)$, and afterwards inserts each edge $e \in E(G) \setminus E(H')$ with label $\lambda^G(e)$. By Equation (5.1), π incurs a cost of 0, which implies $\mathrm{SGED}(H, G) = 0$.

Suppose $\mathrm{SGED}(H, G) = 0$. By Equation (5.1) and Inequalities (5.2) and (5.3), any optimum edit path π between H and G entirely consists of

(i) insertions
(ii) and substitutions that do not have any effect (trivial substitutions).

Thus, H is a subgraph of $\pi(H)$, which implies $H \preceq G$. ∎

Because of Proposition 5.2, we can interpret the subgraph edit distance between two graphs H and G to be a measure for how close H is to being subgraph isomorphic to G. If $\mathrm{SGED}(H, G)$ is a small number, then we consider H to be almost subgraph isomorphic to G.

The following (and previous) proposition is true for SGED, but generally not for GED.

Proposition 5.3 (Monotonicity of SGED).
SGED is monotone in its first argument and anti-monotone in its second argument. That is, for labeled graphs H, H', G and G', we have:

$$\begin{aligned} H' \preceq H &\implies \mathrm{SGED}(H', G) \leq \mathrm{SGED}(H, G), \\ G' \preceq G &\implies \mathrm{SGED}(H, G') \geq \mathrm{SGED}(H, G). \end{aligned}$$ ◇

Proof. By Inequality (5.4) and Proposition 3.28, SGED satisfies the triangle inequality. Suppose $H' \preceq H$. Then, by Proposition 5.2,

$$\mathrm{SGED}(H', G) \leq \mathrm{SGED}(H', H) + \mathrm{SGED}(H, G) = \mathrm{SGED}(H, G).$$

Suppose $G' \preceq G$. Then, again by Proposition 5.2,

$$\mathrm{SGED}(H, G) \leq \mathrm{SGED}(H, G') + \mathrm{SGED}(G', G) = \mathrm{SGED}(H, G').$$ ∎

We finish our discussion of the subgraph edit distance with two negative results concerning the time complexity of SGED.

Proposition 5.4.
The decision problem associated to SGED is NP-complete. ◇

Proof. NP-hardness: Consider an instance (H, G) of SGI. Set $\tau = 0$. By Proposition 5.2, the following equivalency holds:

$$H \preceq G \iff \mathrm{SGED}(H, G) \leq \tau.$$

As a consequence, there is a polynomial-time Turing reduction from the NP-complete SGI problem to the decision problem associated to SGED.

Membership in NP: Follows from Proposition 3.30. ■

Proposition 5.5.
There is no α-approximation algorithm for SGED for any α, unless P=NP. ◇

Proof. Suppose $\mathcal{A}$ is an α-approximation algorithm for SGED, i.e.

$$\forall H, G \in \mathcal{G}\colon \alpha^{-1} \cdot \text{SGED}(H, G) \leq \mathcal{A}(H, G) \leq \alpha \cdot \text{SGED}(H, G).$$

Then the following chain of equivalencies holds:

$$\mathcal{A}(H, G) = 0 \iff \text{SGED}(H, G) = 0 \iff H \preceq G.$$

Thus $\mathcal{A}$ can be used to solve SGI, which implies P=NP. ■

The proofs of Propositions 5.4 and 5.5 reveal that both propositions are true even if the cost of each deletion and non-trivial substitution is set to 1.

5.2 Frequent Approximate Subgraph Mining

In this section, we propose a novel formulation for the frequent approximate subgraph mining problem (FASM) and show that the number of frequent approximate subgraphs is upper-bounded in terms of the number of frequent subgraphs.

Definition 5.6 (Frequent Approximate Subgraph Mining).
Given

(i) the pattern language $\mathcal{L} \coloneqq \mathcal{G}_{\text{con}}$,
(ii) a non-empty transaction database $D = \{G_1, G_2, \ldots, G_N\}$ of labeled graphs,
(iii) an edit cost function for SGED that defines integer edit costs,
(iv) a non-decreasing sequence $t = (t_i)_{i=0}^{n}$ of non-negative integers s.t. $t_n \in [|D|]$, where t_n is called frequency threshold,
(v) and an interestingness predicate $q_D^t\colon \mathcal{L} \to \{\text{true, false}\}$ that is defined as follows: $q_D^t(H) \coloneqq$ true if, for all $i \in \{0, 1, \ldots, n\}$, there is a set $I_{H,i} \subseteq [|D|]$ containing at least t_i transaction IDs s.t.

$$\max_{j \in I_{H,i}} \text{SGED}(H, G_j) \leq i.$$

The problem of generating exactly one representative for each isomorphism class in $Th(\mathcal{L}, D, q_D^t)/\cong$ is called the frequent approximate subgraph mining problem (FASM). We sometimes write q_D instead of q_D^t. ◇

We can assume w.l.o.g. that all edit costs are integer, since the finiteness of D implies that there can only be a finite number of distinct vertex and edge labels, and thus a finite

number of possible edit costs. So the edit cost function can be scaled in order to make the edit costs integer.

Proposition 5.7.
The FASM problem is a generalization of the FCSM problem. ◇

Proof. The FASM problem is identical to the FCSM problem if t is a constant sequence. ■

Since the decision problem associated to SGED is NP-complete, we can not expect the FASM problem to be efficiently solvable. The following proposition confirms this.

Proposition 5.8.
The FASM problem cannot be solved in output polynomial time, unless P=NP. ◇

Proof. Follows from Proposition 3.19, and from the fact that FASM generalizes FCSM. ■

Many algorithms for pattern mining rely on the monotonicity of the interestingness predicate. We intentionally defined the interestingness predicate q_D^t s.t. it is monotone.

Proposition 5.9.
q_D^t is a monotone graph invariant. ◇

Proof. q_D is a graph invariant because SGED is invariant under graph isomorphism. The monotonicity of q_D Follows from the monotonicity of SGED in its first argument. ■

The following proposition shows that the monotonicity of q_D^t enables the FASM problem to be solved correctly by many algorithms for pattern mining.

Proposition 5.10.
Let h be a perfect graph hash function, and let q_D^t be the interestingness predicate as defined in Definition 5.6. Then Algorithm 1, Algorithm 2 and Algorithm 3 solve the FASM problem correctly. ◇

Proof. All three algorithms solve the theory extraction problem correctly for any monotone graph invariant $q\colon \mathcal{L} \to \{\text{true, false}\}$. ■

The main disadvantage of frequent approximate subgraph mining is the often prohibitively large number of interesting patterns. Proposition 5.12 shows that, depending on the choice of the sequence t, our formulation of FASM does not have this disadvantage. However, in general we still get the following negative result:

Proposition 5.11.
The number of frequent approximate subgraphs can be exponentially larger than the number of frequent subgraphs for the same frequency threshold. ◇

Proof. Let the edit costs of each deletion and non-trivial substitution be 1, and let D be the database that contains the two star graphs depicted in Figure 5.1.

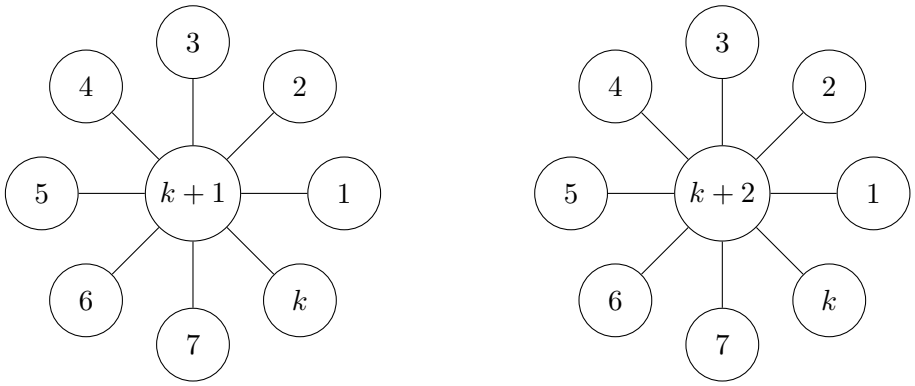

Figure 5.1: Two vertex-labeled star graphs S_k and S'_k on $k+1$ vertices, with label alphabet $\Sigma = \{1, 2, \ldots, k+2\}$. S_k and S'_k differ only in the label of their roots.

Then $Th(\mathcal{L}, D, q_D^{(2,2)})/\cong$ contains $k+1$ graphs: the k leaves of S_k and S'_k, and the empty graph K_0. $Th(\mathcal{L}, D, q_D^{(1,2)})/\cong$, on the other hand, contains all $2^{k+1} + k + 1$ connected subgraphs of S_k and S'_k. ■

We now state the main result of this chapter. We show that the cardinality of the output of FASM for $t = (t_i)_{i=0}^n$ is at most as large as the cardinality of the output of FCSM for frequency threshold t_0, regardless of the values chosen for $(t_i)_{i=1}^n$.

Proposition 5.12.
Let D be a non-empty finite transaction database of labeled graphs, let $t = (t_i)_{i=0}^n$ be a non-decreasing sequence of non-negative integers with $t_0 \geq 1$ and $t_n \in [|D|]$, and let $f_D(t) \coloneqq |Th(\mathcal{L}, D, q_D^t)/\cong|$. Then the following inequality holds:

$$f_D(t) \leq f_D(t_0, t_0, \ldots, t_0). \qquad \diamond$$

Proof. Suppose $q_D^t(H) =$ true, i.e. for any $i \in \{0, \ldots, n\}$ there is a set $I_{H,i}$ of cardinality at least t_i s.t.

$$\max_{j \in I_{H,i}} \text{SGED}(H, G_j) \leq i.$$

Since for all $i \in \{0, \ldots, n\}$ we have $t_0 \leq t_i$, this implies $q_D^{(t_0, \ldots, t_0)}(H) =$ true. So $Th(\mathcal{L}, D, q_D^t) \subseteq Th(\mathcal{L}, D, q_D^{(t_0, \ldots, t_0)})$. ■

We finish this section by discussing alternative definitions for the interestingness predicate q_D^t.

Definition 5.13.
Given $t' \in [|D|]$ and $\tau \in \mathbb{N}_0$. For each $H \in \mathcal{L}$, we define

(i) $q_D^{t',\tau}(H) \coloneqq$ true if there is a set $I_H \subseteq [|D|]$ containing at least t' elements s.t.

$$\max_{j \in I_H} \text{SGED}(H, G_j) \leq \tau.$$

(ii) $\widetilde{q}_D^{t',\tau}(H) \coloneqq$ true if there is a set $I_H \subseteq [|D|]$ containing at least t' elements s.t.

$$\sum_{j \in I_H} \text{SGED}(H, G_j) \leq \tau.$$

◇

$q_D^{t',\tau}$ is a special case of the interestingness predicate given in Definition 5.6, and it is equivalent to the definition given by Li and Wang (2015). However, for $\tau > 0$, there is no upper bound on the number of interesting patterns w.r.t. $q_D^{t',\tau}$ in terms of the number of frequent subgraphs. Moreover, even for very small values of τ, the number of interesting patterns w.r.t. $q_D^{t',\tau}$ is generally prohibitively large, which is why Li and Wang (2015) introduce multiple techniques for reducing the cardinality of the output of their proposed algorithm for approximate frequent subgraph mining.
As is the case for the interestingness predicate given in Definition 5.6, the number of interesting patterns w.r.t. $\widetilde{q}_D^{t',\tau}$ is upper-bounded in terms of the number of frequent subgraphs. However, when using $\widetilde{q}_D^{t',\tau}$, τ generally needs to be much larger than n in order to get a set of interesting patterns that is comparable to the set of interesting patterns w.r.t. q_D^t for $t = (t_i)_{i=0}^n$. As a consequence, support sets are generally much larger when using $\widetilde{q}_D^{t',\tau}$, which leads to significantly more SGED computations.
To summarize, the interestingness predicate given in Definition 5.6 results in a significantly smaller set of interesting patterns than $q_D^{t',\tau}$ when the sequence t is chosen appropriately, and generally leads to smaller support sets than $\widetilde{q}_D^{t',\tau}$.

5.3 The SGED Feature Space

In this section, we propose a novel method for embedding domain-specific graphs into a real-valued feature space, which we call the SGED feature space. The SGED feature space is comparable to the subgraph feature space, but in contrast to subgraph feature vectors, SGED feature vectors are real-valued and thus more expressive.

Definition 5.14 (SGED Feature Space).
Let $\mathcal{F}$ be a set of feature graphs, for example a set of frequent (approximate) subgraphs of a database D. Let G be a labeled graph. For each $H \in \mathcal{F}$, let $\Phi_H(G) \coloneqq \text{SGED}(H, G)$ be the value of G w.r.t. feature H. Then

$$\Psi(G) \coloneqq (\Phi_H(G))_{H \in \mathcal{F}}.$$

is the SGED feature vector of G w.r.t. $\mathcal{F}$. ◇

We want to emphasize that SGED feature vectors can be defined w.r.t. any appropriate set $\mathcal{F}$ of feature graphs, since Definition 5.14 does not specify the set of feature graphs $\mathcal{F}$.

Suppose $\mathcal{F}$ is the set of frequent (approximate) subgraphs of a sufficiently large graph database D, and suppose the frequency threshold t (sequence t) has been chosen s.t. $\mathcal{F}$ contains a sufficient number of graphs. Then, distances (e.g. L^1-distance,

L^2-distance, ...) between SGED feature vectors, as well as the standard inner product in the SGED feature space, can be seen as similarity measures between graphs that are sampled from the same distribution as the graphs in the database D. So any machine learning method that operates on vector-valued data, including methods that make use of distances or inner products in the feature space, can be applied to SGED feature vectors.

Next, we analyze the value ranges of the features Φ_H, since value ranges of features can have an effect on machine learning models.

Proposition 5.15.
Let H and G be labeled graphs, and let $c(H) \coloneqq \mathrm{SGED}(H, K_0)$ be the cost associated to H. Then $c(H)$ is a tight upper bound on $\Phi_H(G)$. That is, the range of Φ_H is $[0, c(H)]$. $\diamond$

Proof. We apply Proposition 3.28 and Proposition 5.2.

$$\Phi_H(G) = \mathrm{SGED}(H, G) \leq \mathrm{SGED}(H, K_0) + \mathrm{SGED}(K_0, G) = c(H).$$

$c(H)$ is tight because $\Phi_H(K_0) = c(H)$. ∎

Suppose all deletions have an edit cost of 1. Then $c(H) = |V(H)| + |E(H)|$, so the range of Φ_H is directly proportional to the size of H. That is to say, features Φ_H corresponding to small graphs H have a small value range, while features corresponding to large graphs have a large value range.
Many machine learning methods, for example methods that make use of distances or inner products in the feature space (e.g. clustering methods, kNN, SVM, ...), are more sensitive to features with a large value range, and less sensitive to features with a small value range. In order for all features to have the same importance, the feature values can be scaled s.t. the value ranges of all features are equal. However, giving more importance to large feature graphs and less importance to smaller feature graphs may arguably be preferable. We can achieve this by simply not scaling the feature values.

Chapter 6

Relaxed Frequent Subgraph Mining with Polynomial Delay

In this chapter, we show that algorithms for FCSM can be adapted to generate a subset of the set of frequent subgraphs with polynomial delay. All previous methods that solve relaxations of the FCSM problem with polynomial delay only generate frequent subtrees, whereas our method for relaxing FCSM results in algorithms that generate frequent subgraphs. In Chapter 7.2, we empirically evaluate the recall of frequent subgraphs when using efficient, but incomplete, algorithms for FCSM.

We start by defining a relaxation of the FCSM problem.

Definition 6.1 (Relaxed FCSM).
Given a finite non-empty database D of labeled graphs, and a frequency threshold $t \in [|D|]$. The problem of generating exactly one representative for each isomorphism class of some subset of $Th(\mathcal{L}, D, q_D^t)/\cong$ is called the relaxed frequent connected subgraph mining problem (Relaxed FCSM). ◇

We now state and prove the main result of this chapter.

Proposition 6.2.
Consider a modification of Algorithm 3, where each computation of SGI is replaced by a true-biased heuristic algorithm $\mathcal{A}$ for SGI, and where h is a graph hash function.
This modification of Algorithm 3 soundly and irredundantly solves the Relaxed FCSM problem with polynomial delay if $\mathcal{A}$ and h can be computed in polynomial time. Analogous statements are true for Algorithms 1 and 2. ◇

Proof. Efficiency: The execution time of the interestingness evaluation is bounded by a polynomial in the size of D, since the execution time of $\mathcal{A}$ is polynomial by assumption. The number of iterations of the for-loop is polynomial because the cardinality of $\rho(P)$ is bounded by a polynomial in $size(P) + size(\Sigma)$. Taking C and F to be prefix trees, the condition of the for-loop can be decided in polynomial time.

Suppose the pattern P is printed, and PatternGrowth(P) is called. If the set

$$\{H \in \rho(P) \mid h(H) \notin C \wedge \{h(H') \mid H' \in \rho^{-1}(H)\} \subseteq F\}$$

contains a pattern H s.t. there are at least t transactions $G \in D$ with $\mathcal{A}(H, G) = \text{true}$, then H is printed, and the time between printing P and H is polynomial.
Otherwise, the algorithm backtracks by returning from the recursive call to PatternGrowth. The number of consecutive backtrack steps, without printing any pattern, is bounded by a polynomial, because the recursion depth is bounded by the size of the largest transaction in D.
So the time between printing P and either printing the next interesting pattern, or terminating without printing another pattern, is polynomial.

Soundness: Suppose H gets printed. Then there are at least t transactions G in D s.t. $\mathcal{A}(H, G) = \text{true}$. Since $\mathcal{A}$ is true-biased, this implies $q_D(H) = \text{true}$.

Irredundancy: The irredundancy proof of Proposition 4.1 only makes use of the fact that h is a graph invariant, so we can reuse the proof.

Termination: Follows from the finiteness of $Th(\mathcal{L}, D, q_D)$, and from the soundness and irredundancy of the algorithm. ∎

Next, we generalize Definition 6.2.

Definition 6.3 (Relaxed FASM).
Given a finite non-empty database D of labeled graphs, and a non-decreasing sequence $t = (t_i)_{i=0}^n$ of non-negative integers s.t. $t_n \in [|D|]$. The problem of generating exactly one representative for each isomorphism class of some subset of $Th(\mathcal{L}, D, q_D^t)/\cong$ is called the relaxed frequent approximate subgraph mining problem (Relaxed FASM). ◇

We can now state the following proposition that is anticipated by the title of this thesis.

Proposition 6.4.
Consider a modification of Algorithm 3, where each SGED computation is replaced by a heuristic algorithm $\mathcal{A}$ that computes upper bounds for SGED, and where h is a graph hash function.
This modification of Algorithm 3 soundly and irredundantly solves the Relaxed FASM problem with polynomial delay if $\mathcal{A}$ and h can be computed in polynomial time. Analogous statements are true for Algorithms 1 and 2. ◇

Proof. Soundness: Suppose $H \in \mathcal{L} \setminus \{K_0\}$ gets printed. Then, for all $i \in \{0, 1, \ldots, n\}$, there is a set $I_{H,i}$ containing at least t_i elements s.t.

$$\max_{j \in I_{H,i}} \mathcal{A}(H, G_j) \leq i.$$

Since $\mathcal{A}$ computes upper bounds for SGED, this implies $q_D^t(H) = \text{true}$.

Efficiency, irredundancy and termination follow from Proposition 6.2. ∎

We remark that a heuristic algorithm for SGED is, in general, neither invariant under graph isomorphism nor monotone in its first argument.

When the set of feature graphs $\mathcal{F}$ is computed using an efficient algorithm for the Relaxed FCSM or Relaxed FASM problem, it is appropriate to also use an efficient algorithm for computing the SGED feature vectors. However, SGED cannot be computed efficiently, so instead we compute lower and upper bounds a and b for SGED(H, G) with heuristic polynomial-time algorithms, and set

$$\Phi_H(G) \coloneqq \frac{a+b}{2}.$$

This yields an efficiently computable method for embedding graphs into a domain-specific feature space.

Chapter 7

Empirical Evaluation

In this chapter, we answer the following two empirical questions:

(i) How does the execution time of the Apriori Pattern Growth algorithm compare to the execution time of pattern growth approaches that do not apply the Apriori Principle?
(ii) How high is the recall of frequent subgraphs when using heuristic algorithms for Relaxed FCSM?

The former question is investigated in Section 7.1 and the latter in Section 7.2.

7.1 Evaluation of the Apriori Pattern Growth Algorithm

We compare the execution time of the Apriori Pattern Growth algorithm to the execution time of a modification of Algorithm 2, which stores the set of hashes of all candidates, and tests, for each candidate, whether a candidate with the same hash value has previously been generated. This way, the interestingness predicate is only evaluated at most once for each isomorphism class.

We measure the execution times of correct algorithms for FCSM, because the number of patterns in the output, and hence the execution times, can vary wildly when evaluating heuristic algorithms for FCSM. That is, we use a perfect graph hash function h and an algorithm that correctly computes SGI. Applying Proposition 5.2, we can use the subgraph edit distance in order to compute SGI. To compute perfect graph hashes we use nauty (McKay and Piperno, 2013), and to compute GED we solve a binary linear programming formulation of GED (Lerouge et al., 2017) with the Gurobi optimizer (Gurobi Optimization, LLC, 2021). We run our single-threaded program on an Intel Core i5 processor with a clock speed of 1.6 GHz.
We conduct all of our experiments on the MUTAG (Debnath et al., 1991) and PTC-MR datasets (Helma et al., 2001), which contain graph representations of chemical compounds. Statistical properties of the datasets are summarized in Table 7.1. The MUTAG and PTC-MR datasets are provided in a uniform file format by Morris et al. (2020). Since nauty ignores edge labels, we remove the edge labels of all graphs and consider only the

Dataset	Graphs	Avg. Nodes	Avg. Edges	Node Labels	Edge Labels
MUTAG	188	17.9	19.8	7	4
PTC-MR	344	14.3	14.7	18	4

Table 7.1: Dataset statistics: The number of graphs, the average number of vertices and edges of each graph, and the number of distinct vertex and edge labels in the dataset.

Dataset	θ	$\lvert\mathcal{F}\rvert$	Baseline	Apriori	Speedup
MUTAG	80%	51	418	43	9.7
	70%	108	1204	177	6.8
	60%	303	9153	907	10.1
	50%	690	48336	2601	18.6
PTC-MR	30%	46	477	115	4.1
	20%	90	1760	338	5.2
	10%	342	12356	1471	8.4
	5%	1564	62476	6787	9.2

Table 7.2: Execution times of the modified generic pattern growth algorithm (baseline) and the Apriori Pattern Growth Algorithm (Apriori) in seconds. θ is the relative frequency threshold, and $|\mathcal{F}|$ is the number of frequent subgraphs.

vertex labels.

The execution times of both algorithms are summarized in Table 7.2. On the MUTAG dataset, the Apriori Pattern Growth algorithm is 18.6 times faster than the modified generic pattern growth algorithm for the smallest tested frequency threshold of $\theta = 50\%$. On the PTC-MR dataset, our algorithm is 9.2 times faster for the smallest tested frequency threshold of $\theta = 5\%$. The speedup tends to increase when the frequency threshold is decreased.
We conclude that exploiting the Apriori Principle significantly improves the execution time of pattern growth approaches for frequent pattern mining. Our algorithm is often an order of magnitude faster than the baseline algorithm, and it can be expected that the speedup increases if the frequency thresholds are further decreased.

7.2 Evaluation of Algorithms for Relaxed FCSM

In this section, we evaluate the recall of frequent subgraphs when solving the Relaxed FCSM problem using efficient heuristic algorithms. That is, we empirically determine the percentage of frequent subgraphs that are generated by efficient heuristic algorithms for Relaxed FCSM.

To solve the Relaxed FCSM problem, we use a modification of Algorithm 2 that stores the hashes of all generated candidates in order to reduce the number of heuristic SGI

Dataset	θ	$\lvert\mathcal{F}\rvert$	RING	IBP-BEAM	IPFP	REFINE
MUTAG	80%	51	11.5	**26.5**	9.8	9.6
	70%	108	11.1	**15.0**	6.8	5.5
	60%	303	3.9	**6.4**	2.8	2.0
	50%	690	2.2	**3.1**	1.3	1.0
PTC-MR	30%	46	21.7	**34.8**	15.4	17.4
	20%	90	20.0	**26.9**	12.1	11.1
	10%	342	10.5	**12.8**	6.1	5.5
	5%	1564	**6.3**	6.1	3.0	2.1

Table 7.3: Recall of frequent subgraphs (in percent) for different heuristic GED methods, averaged over 100 runs. θ is the relative frequency threshold, and $\lvert\mathcal{F}\rvert$ is the number of frequent subgraphs.

computations. Applying Proposition 5.2, we replace SGI computations in Algorithm 2 by heuristic computations of upper bounds for GED.
We compute the recall of frequent subgraphs for each of the following heuristic methods for GED: RING (Blumenthal, Gamper, et al., 2021), IBP-BEAM (Ferrer, Serratosa, and Riesen, 2015), IPFP (Leordeanu, Hebert, and Sukthankar, 2009) and REFINE (Zeng et al., 2009). RING heuristically constructs an instance of the linear sum assignment problem, which can be solved correctly in polynomial time. The other three methods are based on heuristics for local search. Blumenthal, Boria, et al. (2020) give an in-depth comparison of these methods. We make use of GEDLIB (Blumenthal, Bougleux, et al., 2019), which is a library that contains implementations of several heuristic methods for GED. We use the default parameters suggested by GEDLIB for each GED method, with the exception that we increase the number of iterations for IBP-BEAM to 20, as suggested by Blumenthal, Boria, et al. (2020).
We use a perfect graph hash function h derived from nauty, so that the only error source of the algorithms for Relaxed FCSM is the heuristic algorithm for GED. We verified in preliminary experiments that replacing h by the Weisfeiler-Leman method (Weisfeiler and Leman, 1968), which may produce hash collisions, does not decrease the recall of frequent subgraphs by much. This is likely because the Weisfeiler-Leman method asymptotically almost never produces a hash collision (Babai, Erdös, and Selkow, 1980), and for small graphs, the probability of a hash collision is already very low.

The recall of frequent subgraphs for different choices of heuristics for GED is summarized in Table 7.3. For frequency threshold $\theta = 50\%$, the best-performing method on the MUTAG dataset has a recall of 3.1%, while for frequency threshold $\theta = 5\%$, the best-performing method on the PTC-MR dataset has a recall of 6.3%. On both datasets, the recall decreases when the frequency threshold is decreased.
We conclude that the recall of frequent subgraphs is very low when using efficient heuristic methods to solve Relaxed FCSM.

Chapter 8

Conclusion and Future Work

In this thesis, we have shown empirically that our proposed Apriori Pattern Growth algorithm is often an order of magnitude faster than a baseline pattern growth algorithm for FCSM that does not apply the Apriori Principle. Further, we have proven that our proposed formulation for the frequent approximate subgraph mining problem guarantees that the number of frequent approximate subgraphs is upper-bounded in terms of the number of frequent subgraphs, in contrast to the previous formulation by Li and Wang (2015). Finally, we proposed a method that generates a subset of the set of frequent subgraphs with polynomial delay, whereas previous methods that solve relaxations of FCSM with polynomial delay only generate frequent subtrees. However, our empirical evaluation shows that the recall of frequent subgraphs is very low when solving Relaxed FCSM with efficient heuristic algorithms.
This suggests that solving the Relaxed FCSM problem efficiently and with high recall may be difficult if the pattern language $\mathcal{L}$ is not restricted in some way. We conjecture that there is an algorithm that solves the Relaxed FCSM problem efficiently and with high recall if the pattern language $\mathcal{L}$ is restricted to graphs of bounded tree-width.

It can be expected expected that solving the Relaxed FASM problem with efficient heuristic algorithms will result in a very low recall of frequent patterns as well. In order to solve the Relaxed FASM problem efficiently and with high recall, we suggest to restrict the pattern language $\mathcal{L}$ to tree patterns. The BPS method (Welke, Horváth, and Wrobel, 2019) generates a random subset of the set of frequent subtrees of a general graph database with polynomial delay, and has high recall of frequent subtrees. That is, BPS solves the Relaxed FCSM problem efficiently and with high recall, but restricts the pattern language $\mathcal{L}$ to tree patterns. Like PS (Welke, Horváth, and Wrobel, 2018), BPS exploits the fact that there is a polynomial-time algorithm for computing $\mathrm{SGI}(H, G)$ when H and G are trees. In future research, we will adapt the BPS method to generate a random subset of the set of frequent approximate subtrees with polynomial delay, by exploiting the fact that there is a polynomial-time algorithm for computing the constrained tree edit distance (Bille, 2005).

Further, we will empirically evaluate the predictive performance of frequent approximate subgraphs. That is, we will generate the set $\mathcal{F}$ of frequent approximate subgraphs

of a graph database, embed graphs into the SGED feature space w.r.t. $\mathcal{F}$, and apply machine learning methods designed for vector-valued data to the SGED feature vectors. We hypothesize that the set of frequent approximate subgraphs is more robust towards errors in the database than the set of frequent subgraphs. When graphs contain errors, feature vectors w.r.t. frequent approximate subgraphs may thus lead to better predictive performance than feature vectors w.r.t. frequent subgraphs.

In order to further improve the Apriori Pattern Growth algorithm, we will investigate whether it is possible to visit connected graphs with a DFS search s.t. the following two conditions are satisfied:

(i) No two isomorphic graphs are visited.
(ii) A graph G is visited only after representatives of the isomorphism classes of all subgraphs of G have been visited.

The candidate generation technique used in gSpan (Yan and Han, 2002) satisfies condition (i), but not condition (ii). An algorithm for FCSM whose candidate generation satisfies both conditions would not only benefit from a candidate generation that is as efficient as gSpan's, but could also exploit the Apriori Principle. Moreover, such an algorithm would only need to store the set F of hashes of frequent patterns in memory, while Algorithm 3 stores the larger set C of hashes of strong candidates.
Furthermore, we will investigate whether a technique that satisfies both conditions can generate candidates with polynomial delay if the pattern language $\mathcal{L}$ is restricted to tree patterns or to graphs of bounded tree-width.

Bibliography

Agrawal, Rakesh and Ramakrishnan Srikant (1994). "Fast Algorithms for Mining Association Rules in Large Databases." In: *International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., pp. 487–499.

Babai, László, Paul Erdös, and Stanley M. Selkow (1980). "Random Graph Isomorphism." In: *SIAM Journal on Computing* 9.3, pp. 628–635.

Bille, Philip (2005). "A survey on tree edit distance and related problems." In: *Theoretical Computer Science* 337.1, pp. 217–239.

Blumenthal, David B., Nicolas Boria, et al. (Jan. 2020). "Comparing heuristics for graph edit distance computation." In: *The VLDB Journal*. Vol. 29.

Blumenthal, David B., Sébastien Bougleux, et al. (2019). "GEDLIB: A C++ Library for Graph Edit Distance Computation." In: *Graph-Based Representations in Pattern Recognition*. Vol. 11510. LNCS. Springer, pp. 14–24.

Blumenthal, David B., Johann Gamper, et al. (2021). "Upper bounding the graph edit distance based on rings and machine learning." In: *International Journal of Pattern Recognition and Artificial Intelligence* 35.8, 2151008:1–2151008:32.

Bougleux, Sébastien et al. (2017). "Graph Edit Distance as a Quadratic Assignment Problem." In: *Pattern Recognition Letters* 87, pp. 38–46.

Chi, Y., Y. Yang, and R.R. Muntz (2003). "Indexing and mining free trees." In: *IEEE International Conference on Data Mining*, pp. 509–512.

De La Briandais, Rene (1959). "File Searching Using Variable Length Keys." In: *Western Joint Computer Conference*. Association for Computing Machinery, pp. 295–298.

Debnath, Asim K et al. (1991). "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity." In: *Journal of Medicinal Chemistry* 34.2, pp. 786–797.

Deshpande, M. et al. (2005). "Frequent substructure-based approaches for classifying chemical compounds." In: *IEEE Transactions on Knowledge and Data Engineering* 17.8, pp. 1036–1050.

Ferrer, Miquel, Francesc Serratosa, and Kaspar Riesen (2015). "A First Step Towards Exact Graph Edit Distance Using Bipartite Graph Matching." In: *Graph-Based Representations in Pattern Recognition*. Vol. 9069. LNCS. Springer, pp. 77–86.

Gurobi Optimization, LLC (2021). *Gurobi Optimizer Reference Manual*.

Helma, C. et al. (2001). "The Predictive Toxicology Challenge 2000–2001." In: *Bioinformatics* 17.1, pp. 107–108.

Horváth, Tamás, Björn Bringmann, and Luc De Raedt (2006). "Frequent Hypergraph Mining." In: *Inductive Logic Programming*. Vol. 4455. LNCS. Springer, pp. 244–259.

Horváth, Tamás and Jan Ramon (2010). "Efficient frequent connected subgraph mining in graphs of bounded tree-width." In: *Theoretical Computer Science* 411.31, pp. 2784–2797.

Johnson, David S., Mihalis Yannakakis, and Christos H. Papadimitriou (1988). "On generating all maximal independent sets." In: *Information Processing Letters* 27.3, pp. 119–123.

Kuramochi, Michihiro and George Karypis (2001). "Frequent Subgraph Discovery." In: *IEEE International Conference on Data Mining*. IEEE Computer Society, pp. 313–320.

Leordeanu, Marius, Martial Hebert, and Rahul Sukthankar (2009). "An Integer Projected Fixed Point Method for Graph Matching and MAP Inference." In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., pp. 1114–1122.

Lerouge, Julien et al. (2017). "New binary linear programming formulation to compute the graph edit distance." In: *Pattern Recognition* 72, pp. 254–265.

Li, Ruirui and Wei Wang (2015). "REAFUM: Representative Approximate Frequent Subgraph Mining." In: *SIAM International Conference on Data Mining*, pp. 757–765.

Mannila, Heikki and Hannu Toivonen (1997). "Levelwise Search and Borders of Theories in Knowledge Discovery." In: *Data Mining and Knowledge Discovery* 1, pp. 241–258.

McKay, Brendan and Adolfo Piperno (Sept. 2013). "Practical Graph Isomorphism II." In: *Journal of Symbolic Computation* 60, pp. 94–112.

Morris, Christopher et al. (2020). "TUDataset: A collection of benchmark datasets for learning with graphs." In: *ICML 2020 Workshop on Graph Representation Learning and Beyond*.

Nijssen, Siegfried and Joost Kok (2005). "The Gaston Tool for Frequent Subgraph Mining." In: *Electronic Notes in Theoretical Computer Science* 127.1, pp. 77–87.

Sanfeliu, Alberto and King-Sun Fu (1983). "A distance measure between attributed relational graphs for pattern recognition." In: *IEEE Transactions on Systems, Man, and Cybernetics* 13.3, pp. 353–362.

Ullmann, Julian R. (1976). "An Algorithm for Subgraph Isomorphism." In: *Journal of the ACM* 23, pp. 31–42.

Weisfeiler, B. and A. A. Leman (1968). "A reduction of a graph to a canonical form and an algebra arising during this reduction." In: *Nauchno-Technicheskaya Informatsia* 2.9, pp. 12–16.

Welke, Pascal, Tamás Horváth, and Stefan Wrobel (2018). "Probabilistic frequent subtrees for efficient graph classification and retrieval." In: *Machine Learning* 107, pp. 1847–1873.

— (2019). "Probabilistic and exact frequent subtree mining in graphs beyond forests." In: *Machine Learning* 108, pp. 1137–1164.

Yan, Xifeng and Jiawei Han (2002). "gSpan: Graph-Based Substructure Pattern Mining." In: *IEEE International Conference on Data Mining*. IEEE Computer Society, pp. 721–724.

— (2003). "CloseGraph: Mining Closed Frequent Graph Patterns." In: *International Conference on Knowledge discovery and Data Mining*, pp. 286–295.

Zeng, Zhiping et al. (2009). "Comparing Stars: On Approximating Graph Edit Distance." In: *Proceedings of the VLDB Endowment* 2.1, pp. 25–36.