Graph Set Data Mining

Clustering and Pattern Mining in the Context of Cheminformatics

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Till Schäfer

Dortmund
2023

Till Schäfer
Abteilung 1 - Computational Analytics
Institut für Informatik
Universität Bonn
Friedrich-Hirzebruch-Allee 8
53115 Bonn

Tag der mündlichen Prüfung: **05.07.2023**

Dekan **Prof. Dr. Gernot A. Fink**

Gutachter **Prof. Dr. Petra Mutzel**
(Universität Bonn, Institut für Informatik)
Prof. Dr. Kevin Buchin
(Technische Universität Dortmund, Fakultät für Informatik)

Abstract

Graphs are among the most versatile abstract data types in computer science. With the variety comes great adoption in various application fields, such as chemistry, biology, social analysis, logistics, and computer science itself. With the growing capacities of digital storage, the collection of large amounts of data has become the norm in many application fields. Data mining, i.e., the automated extraction of non-trivial patterns from data, is a key step to extract knowledge from these datasets and generate value. This thesis is dedicated to concurrent scalable data mining algorithms beyond traditional notions of efficiency for large-scale datasets of small labeled graphs; more precisely, structural clustering and representative subgraph pattern mining. It is motivated by, but not limited to, the need to analyze molecular libraries of ever-increasing size in the drug discovery process.

Structural clustering makes use of graph theoretical concepts, such as (common) subgraph isomorphisms and frequent subgraphs, to model cluster commonalities directly in the application domain. It is considered computationally demanding for non-restricted graph classes and with very few exceptions prior algorithms are only suitable for very small datasets. This thesis discusses the first truly scalable structural clustering algorithm StruClus with linear worst-case complexity. At the same time, StruClus embraces the inherent values of structural clustering algorithms, i.e., interpretable, consistent, and high-quality results. A novel two-fold sampling strategy with stochastic error bounds for frequent subgraph mining is presented. It enables fast extraction of cluster commonalities in the form of common subgraph representative sets. StruClus is the first structural clustering algorithm with a directed selection of structural cluster-representative patterns regarding homogeneity and separation aspects in the high-dimensional subgraph pattern space. Furthermore, a novel concept of cluster homogeneity balancing using dynamically-sized representatives is discussed.

The second part of this thesis discusses the representative subgraph pattern mining problem in more general terms. A novel objective function maximizes the number of represented graphs for a cardinality-constrained representative set. It is shown that the problem is a special case of the maximum coverage problem and is NP-hard. Based on the greedy approximation of Nemhauser, Wolsey, and Fisher for submodular set function maximization a novel sampling approach is presented. It mines candidate sets that contain an optimal greedy solution with a probabilistic maximum error. This leads to a constant-time algorithm to generate the candidate sets given a fixed-size sample of the dataset. In combination with a cheap single-pass streaming evaluation of the candidate sets, this enables scalability to datasets with billions of molecules on a single machine. Ultimately, the sampling approach leads to the first distributed subgraph pattern mining algorithm that distributes the pattern space and the dataset graphs at the same time.

Acknowledgments

This thesis would not have been possible without the support of many people in various forms.

Foremost, I would like to thank my advisor Prof. Petra Mutzel for her comprehensive support to realize my individual goals and ideas. This support includes, but is not limited to, her scientific guidance and supervision, her doings in providing an encouraging work environment, and her continuous effort to bring together people with different scientific backgrounds and problem fields as a rich source for novel ideas. I would like to thank my co-advisor Prof. Kevin Buchin for the helpful feedback regarding my thesis. Furthermore, I appreciate the agreement of Prof. Johannes Fischer and Prof. Thomas Liebig to be part of my defense committee.

I am thankful to my co-authors, co-workers, and numerous additional people for profound discussions that sharpened and deepened my understanding of scientific methods and topics. In this regard, I would like to name Prof. Nils Kriege for a plethora of good advises, fast implementations of various (sub)graph isomorphism problems, and the shared effort in the development of Scaffold Hunter together with Dr. Karsten Klein. I would like to point out the fruitful collaboration with Dr. Lina Humbeck and the working group of Prof. Oliver Koch. They provided a deep insight into the drug development process and provided rich datasets that served as the basis of my thesis. Furthermore, I would like to thank Prof. Sonja Kuhnt for providing support regarding statistical questions and Christine Dahn, Dr. Andre Droschinsky, Philipp Lewe, Dr. Lutz Oettershagen, Nicolai Parlog, and Dr. Henning Timm for proofreading my thesis.

Finally, I would like to thank my wife Lisa, my two children Lio and Ole, and my other family for their patience and emotional support at all times.

Supplementary Material

The source code and evaluation data of this thesis is published under the following digital object identifiers for reproducibility and transparency w.r.t. the provided results.

Source Code `https://doi.org/10.5281/zenodo.8298948`

Evaluation Data `https://doi.org/10.5281/zenodo.8298921`

Contents

Chapter 1

Introduction

Graphs are among the most versatile abstract data types in computer science. Relational information—such as social interactions, closeness or similarity of objects, connectivity, or semantic subject-object attributions—can be naturally modeled by the connectivity structure of graphs. Attributed and labeled graphs allow annotating entities and their relation, i.e., vertices and edges, with other types of data. Graphs are used to model molecules, social networks, road and train networks, complex knowledge, modular software architectures, hierarchical structures, metabolism models, electric circuits, and many more. With the variety comes a great adoption in various application fields, such as chemistry, biology, social analysis, logistics, and of course computer science itself.

With the growing capacities of digital storage, the collection of large amounts of data has become the norm and this changes the fundamental principles of data collection. The approach of collecting specific data relevant for a specific task is often replaced by a massive collection of data with low or no knowledge of the latter applicability. This era of big data comes with the promise that such data may contain hidden knowledge and that it can be revealed in a downstream knowledge discovery process. Data mining, i.e., the automated extraction of non-trivial patterns from data, is a key step in this knowledge discovery process.

At the same time, computing power does not grow at the same pace. Since clock rates and single-core performance are hitting limits in recent years, parallelization is more and more important to fully utilize modern computer resources. Also, Moore's law seems to hit an end or at least slow down. From a computer science point of view, this implies a stronger focus on algorithmic aspects and a shift to scalability beyond traditional notions of efficiency. In other words, polynomial complexities considered efficient in the past no longer scale with the growing amount of data in many application fields.

> Moore's Law used to grow at 10x every five years [and] 100x every 10 years. Right now Moore's Law is growing a few percent every year. Every 10 years maybe only 2x. [...] So Moore's Law has finished.
>
> (Nvidia CEO Jensen Huang, CES 2019)

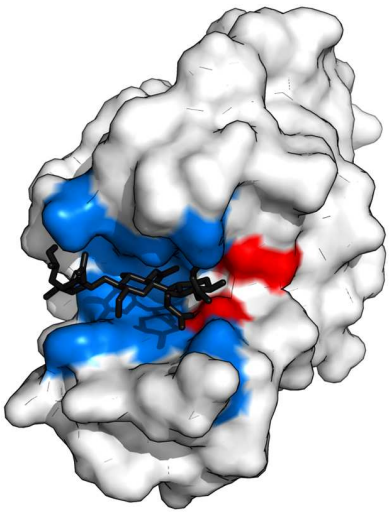


Figure 1.1: Enzyme with a substance (black) docked to a binding cite (blue). The catalytic site (i.e., the place where the chemical reaction is catalyzed) is marked in red and altered by the docked substance. ©Thomas Shafee, CC-BY 4.0

This thesis is dedicated to scalable data mining algorithms for multisets of labeled graphs with linear and sublinear complexities, in particular, clustering and representative subgraph pattern mining algorithms. The major motivation stems from the need to analyze large-scale molecular libraries for drug discovery. However, the presented algorithms are not limited to this use case.

1.1 Molecular Libraries in the Context of Drug Discovery

Drug discovery is the process of finding drugs, i.e., chemical substances or molecules, to cause a specific biological effect in a living organism. Its first step can be the identification of a target inside the organism. This can be a receptor of a cell or protein that is involved in the biological pathway of a disease.

After identifying a target, the next step is to find a suitable substance that alters the function of the cell or protein. For example, the reaction rate of an enzyme (i.e., a protein acting as a catalyst for a chemical reaction in the metabolic process) can be inhibited (decreased) or activated (increased) by a drug that docks to it. Figure 1.1 shows a docked substance to a protein. Since the protein and the drug substance are flexible objects that apply reciprocal forces to each other, the prediction of the docking behavior is a very challenging task [Men+11]. Especially, reverse engineering a specific substance from the intended binding site is usually not possible.

[**Men+11**] Meng et al., "Molecular Docking: A Powerful Approach for Structure-Based Drug Discovery". 2011

Pharmaceutical companies have resorted to brute force chemical substances in a highly automated high-throughput screening process. Thus, they apply a substance to a target protein and observe possible activity, e.g., by using fluorescent indicators. However, with an estimated number of 10^{12} to 10^{180} drug-like molecules [Bro09], this process clearly has limits when it comes to covering the chemical (search) space. As such, a virtual pre-selection process is typically applied to find novel drugs and to advance into unexplored regions of the chemical space.

Enumeration of the complete drug-like chemical space is unfeasible even for modern computers. Furthermore, many theoretical molecules can be ruled out for other reasons, e.g., because there exists no known way to synthesize them in the real world. For these reasons, a virtual pre-selection process usually starts with datasets of candidate molecules. These candidates are often selected in such a way that desired properties can be ensured (such as the synthesizability of the molecules [GC20]). Virtually generated large-scale de-novo datasets (e.g., CH*I*PMUNK [Hum+18] or GDB-17 [Rud+12]) with billions of molecules are often combined with datasets of existing knowledge about known molecules (e.g., PubChem [Kim+20]). To narrow down the set of candidates, the properties (e.g., activity, toxicity) of already known molecules are used to reason about the properties of novel candidates. One of the most fundamental assumption for such an analysis is that the structural similarity of molecules implies similar bioactivity behavior [Mag+14]. Structural similarity analysis is thereby a central problem during this phase of drug discovery. In particular, data mining techniques are often used to extract structural commonalities and align them with annotated properties.

[Bro09] Brown, "Chemoinformatics - an introduction for computer scientists". 2009

[GC20] Gao and Coley, "The Synthesizability of Molecules Proposed by Generative Models". 2020

[Hum+18] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

[Rud+12] Ruddigkeit et al., "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". 2012

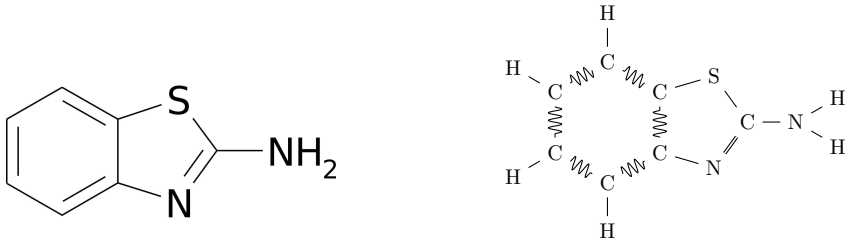
[Kim+20] Kim et al., "PubChem in 2021: new data content and improved web interfaces". 2020

[Mag+14] Maggiora et al., "Molecular Similarity in Medicinal Chemistry". 2014

1.1.1 Graph Representation of Molecules

Molecules are collections of atoms that are attracted by chemical bonds. While molecules are flexible in shape, with three-dimensional arrangements (conformations), they are often modeled or described by their structural formula (see fig. 1.2a for an example), i.e., their atom-bond-connectivity. In a structural formula, atoms are classified by their chemical element, e.g., carbon or hydrogen. The attraction forces, i.e., the bonds between the atoms, can be classified by the number of valence electrons and the way they are shared between atoms (e.g., single, double, or aromatic bonds). Charges, isotopes, and some three-dimensional properties are sometimes incorporated into these classifications. When it comes to the comparison of molecules, it might be tempting to add as much and precise information as possible, i.e., to use a three-dimensional representation. Interestingly, the structural formula as a basis for comparisons has been proven to be more robust in many situations [Mag+14]. One reason is the variance in three-dimensional information (due to flexibility and surrounding influences on the shape) and the uncertainty of three-dimensional information due to restrictions of the measurement. Also, three-dimensional properties or conformations are often not part of existing knowledge, which limits these approaches by the amount of available information.

A molecule's structural formula can be represented naturally as a graph. More precisely, vertices represent atoms and edges represent bonds. Figure 1.2 shows an



(a) Structural formula

(b) Graph representation. Double bonds are depicted as double edges, aromatic bonds are depicted as snake shaped edges.

Figure 1.2: Example structural formula of a molecule alongside a graph representation.

example structural formula and a corresponding graph representation. Labels of vertices and edges are derived from their classification. As seen in the example, some hydrogens (label H) of the graph representation are only implied by the structural formula. Implicit hydrogens can be unambiguously derived from the chemical properties of the adjacent atoms.

1.1.2 Molecular Comparisons

Given a graph representation of a molecule it is possible to use graph comparison methods to determine structural similarity. In drug discovery, the use of numerical or boolean molecular fingerprints as an intermediate representation is a predominantly observed practice [MS11]. Thus, a vectorized intermediate representation is used to compare graphs by means of classical vector distances, e.g., the Jaccard-Coefficient or L^p-norms. Commonly used are structural or circular fingerprints. The former encode the presence of subgraph patterns in the molecules. One problem with these approaches is that a set of patterns must be preselected since the encoding of all possible subgraph patterns would result in unfeasible large feature vectors (cf., section 2.6.1.2). Circular fingerprints—such as ECFP fingerprints [RH10]—encode properties of the neighborhoods of vertices in an iterative procedure, similar to the Weisfeiler-Lehman [Wei68] isomorphism test. While molecular fingerprints are broadly applied due to their computational speed compared to graph-theoretical comparison concepts, they suffer from their lossy representation and their limited interpretability in the application domain.

Graph-theoretical concepts are another method to compare graphs. They include distance measures directly applicable to graph data, e.g., the graph edit distance [SF83]. Furthermore (common) subgraph isomorphisms can express structural equivalence of parts of graphs or molecules. This has the advantage, that commonalities can be explained directly in the application domain (cf., fig. 1.3). As a result, these concepts are considered highly consistent and interpretable [Kri15]. Common subgraph patterns can be used to derive classical distance measures if the size or the weight of common

[MS11] Maggiora and Shanmugasundaram, "Molecular Similarity Measures". 2011

[RH10] Rogers and Hahn, "Extended-Connectivity Fingerprints". 2010

[Wei68] Weisfeiler, "A reduction of a graph to a canonical form and an algebra arising during this reduction". 1968

[SF83] Sanfeliu and Fu, "A Distance measure between attributed relational graphs for pattern recognition." 1983

[Kri15] Kriege, "Comparing Graphs: Algorithms & Applications". 2015

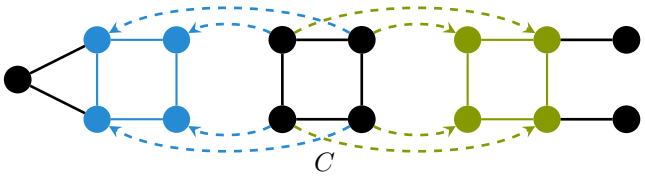


Figure 1.3: Commonality of two graphs expressed as common subgraph pattern C.

subgraph patterns is compared to the size or weight of the compared graphs [e.g., BS98; FV01; Wal+01]. Recent developments have shown, that the maximum common subgraph problem can be extended to incorporate domain-specific knowledge for drug discovery [DKM18] and that the problem is solvable in polynomial time for most molecular graphs if blocks and bridges are preserved [DKM17]. Nevertheless, the high computational complexity of general graph-theoretic approaches is often a limiting factor for their applicability.

1.1.3 Analysis of Molecular Libraries with Scaffold Hunter

Molecular similarity analysis is usually not a simple molecule-to-molecule comparison. Instead, data mining tools use similarities to extract patterns from data and handle datasets whose size make them intractable for human inspection. This section will exemplarily present the software Scaffold Hunter to give an insight into computer-aided drug discovery. It will serve as the main motivation for the content of this thesis.

Scaffold Hunter [Sch+17] is an award-winning[1] visual analytics tool for molecular datasets and drug discovery. The central concept is a multi-view-based (cf., fig. 1.4) analytical reasoning process, that combines data mining techniques with highly customizable interactive visualizations. By aligning structural and property-based classifications with visual expressions of annotated properties, it is possible to greatly reduce the complexity of dataset analysis while making patterns discoverable through human perception. Thus, Scaffold Hunter interleaves human reasoning and learning with a computer-aided structured view on data. This serves as an iterative filtering and refinement process of candidate sets for drug discovery. Incorporating human knowledge in an iterative process is especially useful when generating new hypotheses in an explorative manner.

A typical workflow in Scaffold Hunter starts with the creation of a molecular dataset by importing molecular data from (possibly multiple) external sources. These datasets contain structural information of the molecules as well as annotated properties. The properties can be imported from external sources or computed by various plugins, e.g., to create molecular fingerprints for latter comparisons, derive numerical properties such as molecular weight from structural information, or compute predictions for drug-likeliness. After data integration, it is possible to display the dataset in various views.

[1] `http://old.opentox.org/meet/opentoxeu2013/opentoxeuro13awards`

[BS98] Bunke and Shearer, "A graph distance metric based on the maximal common subgraph". 1998

[FV01] Fernández and Valiente, "A graph distance metric combining maximum common subgraph and minimum common supergraph". 2001

[Wal+01] Wallis et al., "Graph distances using graph union". 2001

[DKM18] Droschinsky, Kriege, and Mutzel, "Largest Weight Common Subtree Embeddings with Distance Penalties". 2018

[DKM17] Droschinsky, Kriege, and Mutzel, "Finding Largest Common Substructures of Molecules in Quadratic Time". 2017

[Sch+17] Schäfer et al., "Scaffold Hunter: a comprehensive visual analytics framework for drug discovery". 2017

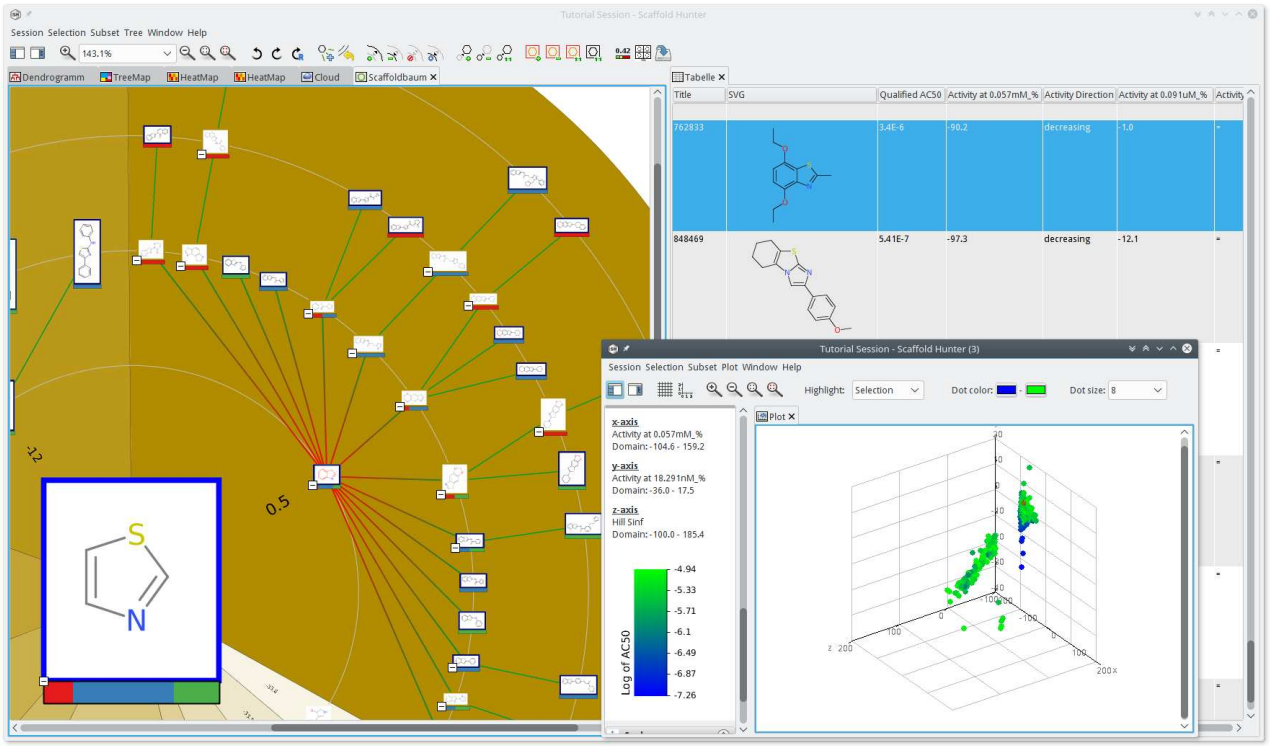


Figure 1.4: Scaffold Hunter showing a molecular dataset in multiple views and windows. The table view displays the dataset structure with annotated properties for each molecular structure. Also visible are the scaffold tree view (left side) and the plot view (window in front).

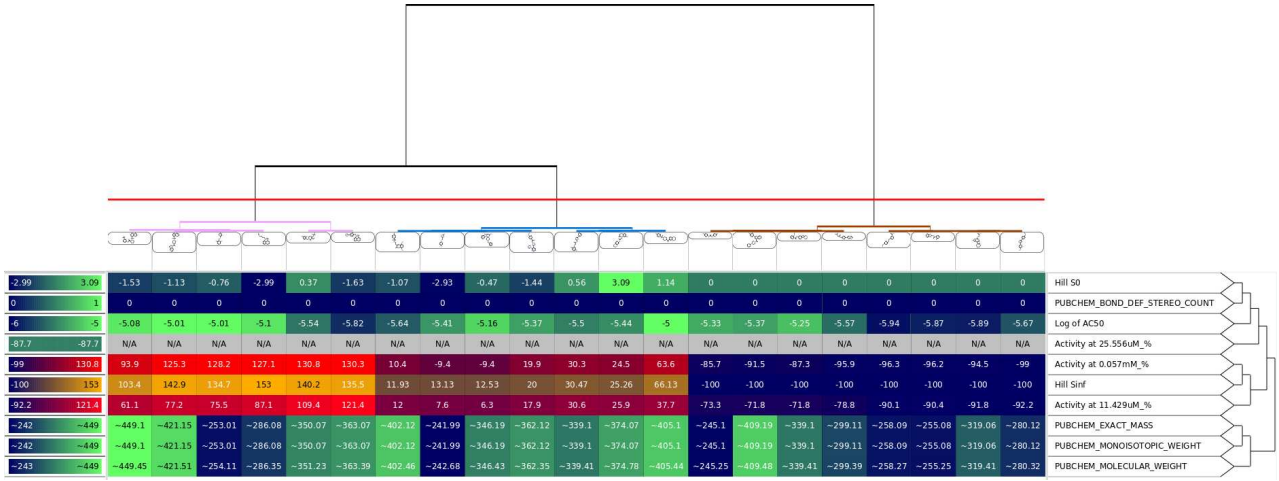


Figure 1.5: Hierarchical molecular clustering in Scaffold Hunter with attached heat map. The heat map displays the values of annotated properties (rows) of molecules (columns) with the help of color gradients. Property names are displayed on the right side; color legends on the left side. The red bar in the top dendrogram indicates a cutoff to determine a flat clustering with 3 clusters.

The refinement of a dataset is possible by creating subsets of the complete dataset that are of special interest. These subsets can be created in various ways: Most prominent is the manual selection of molecules or classes of molecules in the various views of Scaffold Hunter. In addition, it is possible to filter datasets by property-, similarity-, and substructure-based search criteria, by combining or intersecting existing subsets, and by sampling. Thus, it is possible to compare and integrate the results of multiple independent reasoning processes.

The name Scaffold Hunter originates from the concept of scaffold trees, a hierarchical chemical classification scheme developed by Schuffenhauer et al. [Sch+07]. Scaffold trees reduce molecules to scaffolds, which are roughly speaking ring systems without loose side chains. Molecules can then be classified by common scaffolds, which are selected using a rule-based reduction scheme. This initial classification scheme was later complemented by hierarchical clustering algorithms, including an accelerated SAHN (Sequential Agglomerative Hierarchical Non-Overlapping) clustering algorithm for metric distance measures [KMS14a]. The latest additions of data mining tools are various dimension-reduction techniques to embed structural or property-based similarities in low-dimensional representations.

Figure 1.5 show a hierarchical clustering of molecules with an attached heat map developed during a Google Summer of Code project [Stu+15]. The dendrogram on the top is cut by a distance threshold (horizontal red line), such that the dataset is

[Sch+07] Schuffenhauer et al. "The Scaffold Tree - Visualization of the Scaffold Universe by Hierarchical Scaffold Classification". 2007

[KMS14a] Kriege, Mutzel, and Schäfer, "Practical SAHN Clustering for Very Large Data Sets and Expensive Distance Metrics". 2014

[Stu+15] Sturm et al., "Extending the Scaffold Hunter Visualization Toolkit with Interactive Heatmaps". 2015

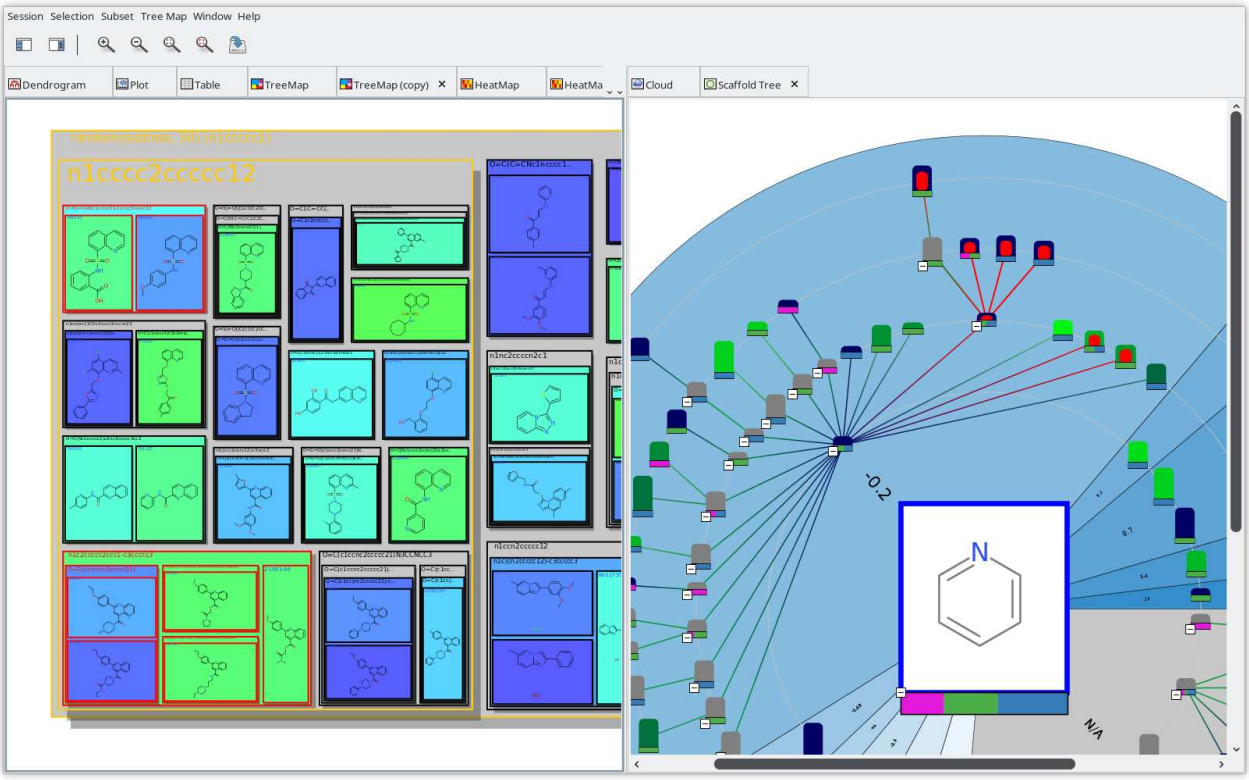


Figure 1.6: Treemap view (left side) and scaffold tree view (right hand side) of Scaffold Hunter. The selections (red highlighting in each view) are linked across both views. colors in both views show visual representations of annotated properties.

split into three different clusters (indicated by the colors of the dendrogram). The heat map shows annotated properties for each molecule, mapping values to colors for a quick visual perception. The screenshot reveals an alignment of the selected clusters with activity properties. The properties themselves are also clustered on the right side, making it easy to discover a high correlation of the several annotated weight properties. Such insights are useful in various ways during drug discovery. For example, one could add novel molecules with unknown activity to the clustering and infer activities by inductive reasoning.

The treemap view in fig. 1.6 is another way to display hierarchical classifications of molecules and scaffolds. A specific strength of this view is that classes aligning with the selected property (mapped to the background color) are easy to catch, even if they are present on different levels of the hierarchy. Furthermore, the screenshot

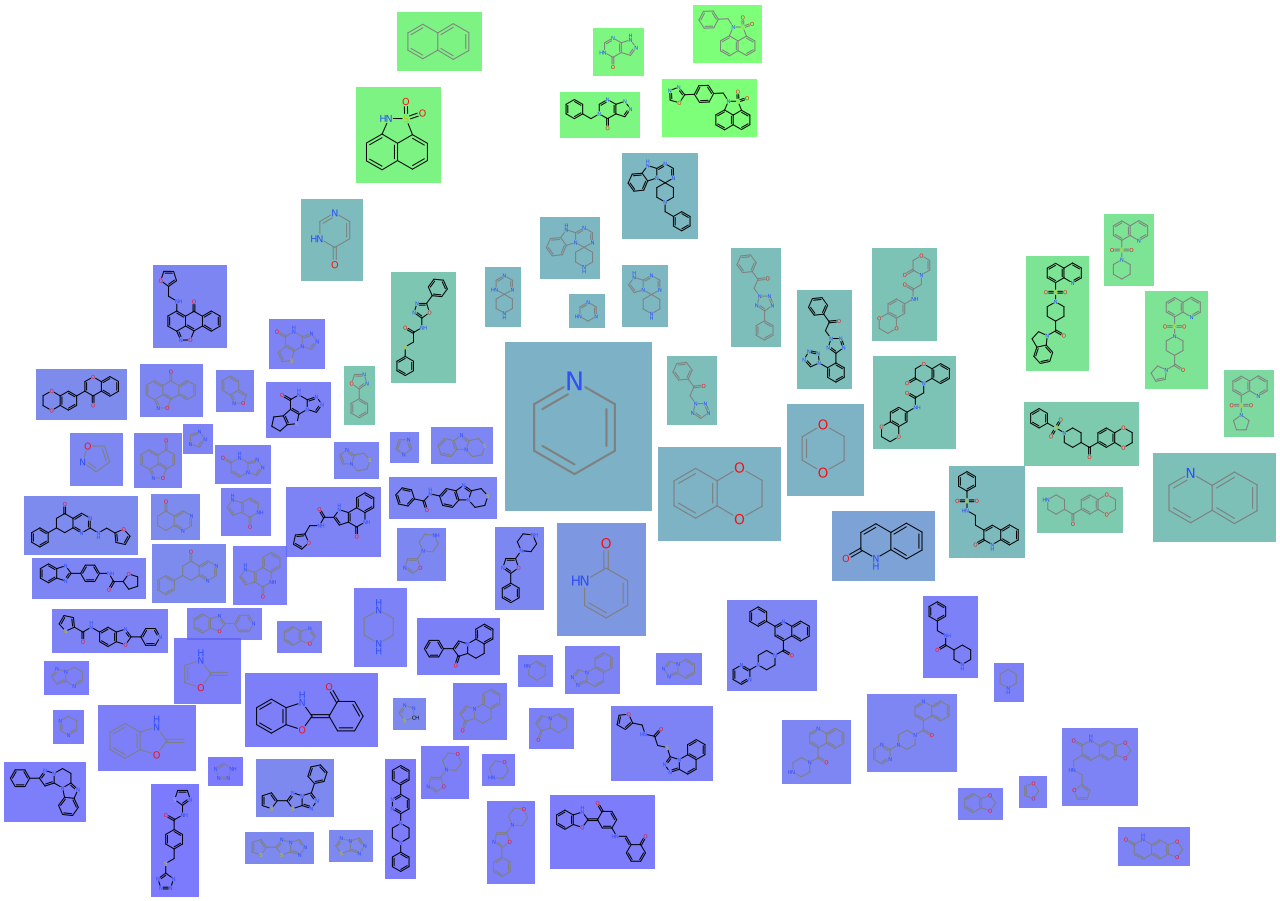


Figure 1.7: Molecular cloud view of Scaffold Hunter

displays another strength of Scaffold Hunter, which is the global selection of molecules spanning all views on the dataset. Here, the red selections in the left corners of the treemap are also visible at the top of the scaffold tree view. The screenshot also shows properties mapped to various visual representations of the scaffold tree view, such as the background color of nodes, colored tree edges, and info bars showing distributions of properties.

Figure 1.7 shows a view inspired by word clouds. The initial application to the domain of cheminformatics by Ertl and Rohde [ER12] was extended in Scaffold Hunter with the integration of semantic layout algorithms. Thus, similar molecules (using a broad range of similarity measures), are grouped close to each other. In this sense, a semantic layout can serve similar goals as clustering with the difference that no hard cluster borders are given. The relation between molecular similarity and an annotated

[**ER12**] Ertl and Rohde "The Molecule Cloud - compact visualization of large collections of molecules". 2012

property (mapped to the background color) is visible in the provided example. This allows the user to get a quick glance at the similarity relation of molecules concerning other properties such as the docking behavior.

It can be concluded, that various types of (unsupervised) classification are crucial for visual analytic tasks, especially for complexity reduction and inductive reasoning. Revealing (structural) patterns in a dataset can be the starting point to develop hypotheses in the drug discovery process.

1.2 Contributions

This thesis presents novel data mining algorithms for the structural analysis of datasets of small labeled graphs (e.g., molecules) with a special focus on scalability. It is shown that consistent and interpretable graph-theoretical concepts are applicable to graph data mining alongside scalability to large-scale datasets.

A novel structural projected clustering algorithm, named StruClus, for datasets of small labeled graphs is present in chapter 3. To my knowledge, this is the first approach using a directed selection of structural cluster-representative patterns with a ranking function that respects homogeneity and separation aspects. Additionally, it is the first approach that balances cluster homogeneity with the help of dynamically-sized representatives and an adaptive minimum-support threshold. A new error-bounded frequency test for maximal frequent subgraph sampling is able to accelerate the clustering algorithm to scale to dataset cardinalities that other structural clustering algorithms are unable to handle in a reasonable amount of time. The experimental evaluation shows that StruClus outperforms competitors not only in terms of running time but also in terms of quality by a large margin. In joint work with others, StruClus was used for the novelty analysis of the CH*I*PMUNK molecular library.

In chapter 4 a distributed algorithm for representative subgraph pattern mining is presented. To mine a set of representatives, a novel objective function is introduced that maximizes the number of represented graphs for a given number of representatives. It is shown that the problem is a special case of the maximum coverage problem and is NP-hard. An adoption of the greedy approximation algorithm for submodular set function maximization of Nemhauser, Wolsey, and Fisher [NWF78] for the problem is given. A novel sampling approach is presented to generate candidate sets, that contain an optimal greedy solution with a probabilistic maximum error. Since the sample size is fixed, this leads to a constant-time algorithm to generate the candidate sets. In combination with a cheap single-pass streaming evaluation of the candidate sets, this enables scalability to datasets with billions of molecules on a single machine. During the experimental evaluation, it is shown that the sampling strategy can increase performance by up to two orders of magnitudes even on medium-sized datasets. Furthermore, the sampling approach enables a distributed computation of representative sets. In addition to the speedups of the sampling approach, parallel speedups in the range of two magnitudes can be achieved using 16 workers with 10 cores each. The usefulness of the objective is demonstrated by applying it to the pre-clustering subroutine of StruClus, which results in a significantly increased

[NWF78] Nemhauser, Wolsey, and Fisher "An analysis of approximations for maximizing submodular set functions—I". 1978

clustering quality. To the best of my knowledge, the approach is the first subgraph pattern mining algorithm that distributes the pattern space and the dataset graphs at the same time.

1.3 Corresponding Publications

The results of this thesis are partially published in the work listed below.

(a) T. Schäfer and P. Mutzel. "StruClus: Scalable Structural Graph Set Clustering with Representative Sampling". In: *Advanced Data Mining and Applications*. Ed. by G. Cong, W.-C. Peng, W. E. Zhang, C. Li, and A. Sun. Springer International Publishing, 2017, pp. 343–359

(b) N. M. Kriege, P. Mutzel, and T. Schäfer. "SAHN Clustering in Arbitrary Metric Spaces Using Heuristic Nearest Neighbor Search". In: *Algorithms and Computation - 8th International Workshop, WALCOM 2014, Chennai, India, February 13-15, 2014, Proceedings*. Ed. by S. P. Pal and K. Sadakane. Vol. 8344. Lecture Notes in Computer Science. Springer, 2014, pp. 90–101. DOI: 10.1007/978-3-319-04657-0_11

(c) K. Klein, O. Koch, N. Kriege, P. Mutzel, and T. Schäfer. "Visual Analysis of Biological Activity Data with Scaffold Hunter". In: *Molecular Informatics* 32.11-12 [2013], pp. 964–975. DOI: 10.1002/minf.201300087

(d) N. Kriege, P. Mutzel, and T. Schäfer. "Practical SAHN Clustering for Very Large Data Sets and Expensive Distance Metrics". In: *Journal of Graph Algorithms and Applications* 18.4 [2014], pp. 577–602. DOI: 10.7155/jgaa.00338

(e) T. Schäfer, N. M. Kriege, L. Humbeck, K. Klein, O. Koch, and P. Mutzel. "Scaffold Hunter: a comprehensive visual analytics framework for drug discovery". In: *J. Cheminformatics* 9.1 [2017], 28:1–28:18. DOI: 10.1186/s13321-017-0213-3

(f) B. K. Stöcker, T. Schäfer, P. Mutzel, J. Köster, N. M. Kriege, and S. Rahmann. "Protein Complex Similarity Based on Weisfeiler-Lehman Labeling". In: *Similarity Search and Applications - 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2-4, 2019, Proceedings*. Ed. by G. Amato, C. Gennaro, V. Oria, and M. Radovanovic. Vol. 11807. Lecture Notes in Computer Science. Springer, 2019, pp. 308–322. DOI: 10.1007/978-3-030-32047-8_27

(g) L. Humbeck, S. Weigang, T. Schäfer, P. Mutzel, and O. Koch. "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". In: *ChemMedChem* 13.6 [2018], pp. 532–539. DOI: 10.1002/cmdc.201700689

(h) W. Sturm, T. Schäfer, T. Schreck, A. Holzinger, and T. Ullrich. "Extending the Scaffold Hunter Visualization Toolkit with Interactive Heatmaps". In: *Computer Graphics and Visual Computing (CGVC)*. ed. by R. Borgo and C. Turkay. The Eurographics Association, 2015. DOI: 10.2312/cgvc.20151247

I contributed as main author to the publications a and b. The publications c to f are joint work with other authors. I have mentored the work published in h during a Google Summer of Code project.

My contributions to Scaffold Hunter (cf., section 1.1.3) were especially focused on clustering, data integration, iterative workflow concepts, mentoring of students, and project management. These contributions are partialy published in the publications b to e and h. The publications d and f are related to the similarity analysis of graphs. The StruClus algorithm (cf., chapter 3) is presented and applied in the publications a and g.

The publication g was awarded as very important paper by the ChemMedChem Journal and presented as cover feature[2]. It was one of the 20 most read ChemMedChem articles in 2017-2018. Furthermore, it received the interdisciplinary award[3] of the DFG priority program 1736 (Algorithms for BIG DATA).

The distributed representative mining algorithm presented in chapter 4 is unpublished at this time.

[2] https://doi.org/10.1002/cmdc.201800126

[3] https://www.big-data-spp.de/publications/awards

Chapter 2
Preliminaries

2.1 Basic Notation

The natural numbers including zero are denoted by $\mathbb{N}$, the rational numbers by $\mathbb{Q}$ and the real numbers by $\mathbb{R}$. A restriction to a set of numbers is expressed as superscript inequality, e.g., $\mathbb{N}^{>0}$ for strictly positive natural numbers. The boolean set $\{\text{true}, \text{false}\}$ is donated by $\mathbb{B}$. A union of pairwise disjoint sets is indicated by the symbol $\uplus$. A multiset M is a tuple (A, m), where $A = \{a_1, \ldots, a_n\}$ is the underlying set of M and $m : A \to \mathbb{N}$ is a multiplicity function. M is also denoted by $\{\!\{a_1^{m(a_1)}, \ldots, a_n^{m(a_n)}\}\!\}$. The cardinality $|M|$ of a multiset $M = (A, m)$ is the sum of all multiplicities, i.e., $\sum_{a \in A} m(a)$. The union (intersection) of two multisets $M = (A, m_M)$ and $N = (B, m_N)$ is defined as a multiset $Q = (C = (A \cup B), m_Q)$ with $\forall c \in C : m_Q(c) := \max\{m_M(c), m_N(c)\}$ ($\forall c \in C : m_Q(c) := \min\{m_M(c), m_N(c)\}$). The support of a multiset $M = (A, m)$ is the set of distinct elements of the underlying set A with strictly positive multiplicity, i.e., $\{a \in A \mid m(a) > 0\}$. $\mathbf{1}_A : X \to A$ is the indicator function, that maps some $x \in X$ to 1 if $x \in A$ and maps to 0 otherwise. A k-ary relation $R \subseteq A_1 \times \cdots \times A_k$ is a subset of the Cartesian product of the sets $A_1, \ldots, A_k$. For convenience, binary relations are written in the infix notation, i.e., $a\ R\ b$ is written iff $(a, b) \in R$ and $a\ \not{R}\ b$ otherwise. For a binary relation $R \subseteq A_1 \times A_2$, A_1 is called domain of R and A_2 codomain of R. An equivalence relation is a binary relation $\sim\ \subseteq A \times A$ on a set A that is (a) reflexive, i.e., $\forall a \in A : a \sim a$, (b) symmetric, i.e., $\forall a, b \in A : a \sim b \Leftrightarrow b \sim a$, and (c) transitive, i.e., $\forall a, b, c \in A : a \sim b \wedge b \sim c \Rightarrow a \sim c$. The equivalence class of an element $a \in A$ is written as $[a] := \{x \in A \mid x \sim a\}$. The quotient set $\{[a] \mid a \in A\}$, i.e., the set of all equivalence classes of a set A, is written as $A/\!\sim$. Let A be a set, $\sim$ an equivalence relation, and $\rho : A/\!\sim \to A$ be an injective function that maps each equivalence class $E \in A/\!\sim$ to any $a \in E = [a]$. Then, ρ is called representative function of A under $\sim$, a is called representative of E under

ρ, and $[A]_\sim := \{\rho(x) \mid x \in A/\sim\}$ is called representative set of A for the equivalence relation $\sim$ and the representative function ρ. The notations lex max and lex min are used for the lexicographic maximum and minimum of tuples and strings.

2.2 Computation Models, Frameworks and Complexity

A model of computation defines in which way a mathematical problem can be solved on an abstract type of computer. Such a model is the basis for computational complexity analysis. Besides pure theoretical models, most of the models somehow abstract real hardware in order to perform theoretical analysis that can predict the behavior of algorithms on real-world machines. This section will cover models of computation as well as computation frameworks, that are needed for the analysis of algorithms presented in this thesis. Additionally, a model of computation defines the algorithmic operations, that can be used in pseudocode. Thereby, this chapter will define some operations that are not commonly used and reflect parallel or distributed execution. This section is partially based on Savage [Sav98] and Cormen et al. [Cor+09].

[**Sav98**] Savage *Models of computation - exploring the power of computing*. 1998

[**Cor+09**] Cormen et al. *Introduction to Algorithms, 3rd Edition*. 2009

2.2.1 Random-Access Machine

A random-access machine (RAM) is an abstract machine, that processes sequential algorithms. The RAM can be divided into two parts, the central processing unit (CPU, or processor) and the random-access memory. The random-access memory is a sequence of m b-bit words, where each word can be accessed by a unique address. It holds the executed program as well as data, such as the problem input, states, and (intermediate) results. Indirect addressing is supported by the RAM model, i.e., memory can be accessed by an address, that is stored in another word of the memory. The CPU operates in cycles, where each cycle fetches a command from the memory—which is addressed by a program counter—and executes it. Such a command can contain arithmetic operations (e.g., `add`, `mult`), logic operations (e.g., `or`, `and`), comparisons (e.g., `equals`), and jump instructions. Per default a program is read sequentially in the order it is stored in memory, i.e., the program counter is incremented after each cycle. Jump instructions are used to modify the program counter (based on other operations such as comparisons) and override the sequential order. They are the basis for loops (e.g., `while` or `for` statements), conditional jumps (e.g., `if` or `switch` statements), and re-used subroutines.

2.2.1.1 Computational Complexity in the RAM Model

It is assumed, that each command or operation is atomic and performed in a single cycle. Consequentially, the work that needs to be performed by an algorithm to terminate can be measured in terms of operations. Furthermore, it is assumed that each cycle requires an equal amount of time. Time and work are thereby equivalent in the RAM model. Additionally, the amount of memory, i.e., the number of words m, which are required to execute a program can be subject to analysis in the RAM model.

2.2.2 Parallel Random-Access Machine

A parallel random-access machine (PRAM) is an abstract machine, which processes parallel algorithms. It consists of a set of RAMs and a shared random-access memory. Thus, each RAM has access to its local and the shared memory. Each processor $P_{\mathrm{id}_1}, \ldots, P_{\mathrm{id}_p}$ has a unique identifier id that is accessible to the program, such that the program can make processor-dependent decisions. Each RAM executes its own program. The PRAM assumes that each RAM is operating synchronously and executes three steps in each cycle: (1) read a word from the shared memory to the local memory, (2) execute a command, (3) write a word from the local memory to the shared memory.

The access to the shared memory of a PRAM can be either exclusive (E) or concurrent (C) for read (R) and write (W) access. Thus, there exist four variants exclusive read / exclusive write (EREW), concurrent read / exclusive write (CREW), exclusive read / concurrent write (ERCW), and concurrent read / concurrent write (CRCW). In the exclusive case, a valid program needs to ensure, that no two processors access the same shared memory address in the same cycle. The concurrent write variants can be furthermore divided into submodels which define the way possible write conflicts are resolved: (a) the weak model restricts concurrent writes to the value zero, (b) the common model restricts concurrent writes to writes of a common value, (c) the arbitrary model selects a random write, (d) the priority model selects the write of the RAM with the lowest / highest processor id, (e) the strong model selects the write with the lowest / highest value. In this thesis, the CRCW PRAM with the arbitrary write conflict model for the analysis of parallel algorithms is used, if not stated otherwise.

The PRAM model gained the parameter p, i.e., the number of processors, in comparison to the RAM model. In general, it is assumed that an unlimited number of processors is available. However, a parallel algorithm for a fixed problem instance does only require a certain amount of processors. Furthermore, an algorithm might have a parameter $p_{\max}$ that reflects the maximum number of processors the algorithm is allowed to use.

2.2.2.1 Computational Complexity in the PRAM Model

In alignment with the RAM model, work specifies the number of operations a parallel algorithm performs on a PRAM, i.e., the sum of operations performed on each processor. Consequentially, the amount of time, i.e., the number of concurrent cycles until the algorithm terminates, is no longer equivalent to the performed work. When the algorithm is parameterized by a maximum number of processors $p_{\max}$, both measures are often given in dependency of $p_{\max}$. When t_i is the running time with i processors, $\frac{t_1}{t_p}$ is called speedup for p processors. Sometimes the time required by a sequential algorithm t_{seq} diverges from the parallel algorithm running time with one processor. Thereby, speedup may also be given by $\frac{t_{\mathrm{seq}}}{t_p}$ or in relation to another processor count (e.g., $\frac{t_{\frac{p}{2}}}{t_p}$) A parallel algorithm is called work optimal in the PRAM

model if the work $w(n) \in \mathcal{O}(t_1(n))$ w.r.t. the input size n. It is called work efficient if $w(n) \in \mathcal{O}(t_1(n) \text{ polylog}(n))$

2.2.3 Distributed-Memory Computers

> **Definition 2.1 (Distributed-Memory computer).** *A distributed-memory computer is a parallel computer, which consists of multiple computing nodes, which communicate with each other by sending messages.*

Thus, in contrast to shared-memory computers—which communicate with the help of a common memory—, processors in distributed computers usually are much looser tied to each other. Traditionally, a node was defined as a processor with local memory, which was connected over a network with other nodes. However, with increasing core counts, NUMA (non-uniform memory access) architectures, and increasingly powerful hardware, it becomes increasingly popular to have multiple processes running on a single hardware, that exchange messages with each other. It is also common practice to define a node as a parallel shared-memory computer, i.e., a distributed model might be a hybrid of shared- and distributed-memory layouts. This thesis will use the following nomenclature:

Node: A node is a real piece of hardware, that is connected to other hardware via some kind of network.

Worker: A worker is a process that communicates with other workers via message exchange.

Task: A task is a piece of work, i.e., an algorithm or algorithm subroutine that can be executed on a worker.

A worker can have shared resources with other workers if multiple workers run on the same node, e.g., memory, caches, CPU cores, or thermal/power budget.

Distributed computing models usually assume that processor cycles are running asynchronously for each node. The communication between nodes is considered to be much more costly than between processors in a shared-memory model. A message often has a latency α and a bandwidth limitation β associated with it. Thus, the time a message of length l needs to be sent may be modeled as $\alpha + l\beta$. The analysis of distributed algorithms strongly depends on the network architecture (e.g., some network topologies share resources, such that the message bandwidth or latency between two nodes is influenced by other communication) and the parallel processing framework, which often imposes limitations on the data or message flow.

2.2.4 Distributed Computation Models and Frameworks

Maybe the most prominent classical distributed computation framework is the message passing interfaces (MPI) standard. It assumes computation nodes to be connected to a network and provides a low-level API to send messages to other nodes. Besides some

basic grouping of nodes in order to perform efficient communication, the framework does not provide any further mechanisms to distribute work and data among the nodes, perform synchronization, provide a coherent view on data, manipulated data, or recover from failures. Thereby MPI is a very flexible, but low-level framework for distributed computing.

Another class of distributed frameworks are distributed shared-memory (DSM) abstractions (cf., [NL91]), which emulate a shared memory on a distributed computer. In comparison with MPI, DSM frameworks provide a higher abstraction and provide a unified view on the distributed memory. They provide mechanisms for cache coherence and synchronization. However, data and work must be distributed manually in order to ensure properties such as data locality.

[NL91] Nitzberg and Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms". 1991

In Big-Data analytics many high-level frameworks have been proposed recently, that resolve around a data-centric viewpoint and provide high-level operations on data rather than providing direct access to messaging or memory operations. This abstraction usually provides some kind of load balancing, fault tolerance, and a rich feature set for data manipulation. Examples are Hadoop[1], Flink[2], Storm[3], Samza[4], Heron[5] and Presto[6]. In the following the Spark framework—which implements the concept of resilient distributed datasets (RDDs)—is presented in more detail, since the implementations of distributed algorithms in this thesis are based on this framework. The MapReduce framework is often considered a forerunner model and shares some basic ideas.

2.2.4.1 MapReduce

Dean and Ghemawat introduced the MapReduce programming model in 2004 [DG04]. Their Google-internal implementation was quickly complemented by various open-source projects, such as Hadoop[1] and Phoenix[7]. The distribution of the data and work is handled by the MaReduce implementation with the ability to break down the work into smaller tasks. This enables the framework to recover from failures of workers and to provide an automated task scheduling.

[DG04] Dean and Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters". 2004

The MapReduce implementation of Dean and Ghemawat [DG08] runs in five phases: input, map, shuffle, reduce, and output. The programmer does only specify a `map` and a `reduce` function to implement the logic of the algorithm. The other phases are handled by the MapReduce implementation. The type signatures of `map` and `reduce` are defined as:

[DG08] Dean and Ghemawat "MapReduce: Simplified Data Processing on Large Clusters". 2008

$$\begin{aligned} \text{map}: &\quad (\text{K1}, \text{V1}) \to \text{seq}[(\text{K2}, \text{V2})] \\ \text{reduce}: &\quad (\text{K2}, \text{seq}[\text{V2}]) \to \text{seq}[\text{V2}] \end{aligned}$$

[1] `https://hadoop.apache.org`
[2] `https://flink.apache.org`
[3] `https://storm.apache.org/`
[4] `https://samza.apache.org/`
[5] `https://heron.apache.org/`
[6] `https://prestodb.io/`
[7] `https://github.com/kozyraki/phoenix`

K1 and K2 are key types and V1 and V2 are value types. The map phase is often associated with the extraction of features, while the reduce phase aggregates the features to a common value. Several implementations relax the above definition to allow other return types in the reduce phase. For example, Hadoop defines the `reduce` functions type signature as $(\text{K2}, \text{seq}[\text{V2}]) \rightarrow \text{seq}[(\text{K3}, \text{V3})]$.

In the input phase, the dataset is split into smaller parts, that align with record boundaries. A record is an atomic unit of data, that must be processed as a whole. The splits are then distributed to a set of workers (mappers), which apply the custom map function on each record (map phase). The results of the map function are then grouped by their key (shuffle phase) and fed to a set of workers (reducers), which apply the reduce function (reduce phase). In the last step (output phase), the reducer's results are written to persistent storage.

A famous algorithmic example for MapReduce is the word counting algorithm. The task is to count the number of occurrences of the words which are contained in a text. A straightforward MapReduce implementation splits the text input into records of words (strings). The mapper then outputs the word as the key and the number one as the value. The reducers then sum up all the values for each key.

[DG08] Dean and Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters". 2008

A major weakness of MapReduce is data locality. For example, the framework presented in [DG08] does not provide any local aggregation of values. This is addressed by the combine phase of Hadoop, which lets the programmer specify a `combine` function. The combine phase basically serves as a local reducer and is executed after the map and before the shuffle phase. In the above word count example, this would avoid the transfer of multiple key-value pairs with the same key from a single mapper to the reducer of the key. Nevertheless, the shuffle phase usually transfers a huge amount of data among the network to group the keys. A second weakness of MapReduce is the forced persistence after each reduce phase, especially for iterative algorithms which require multiple passes of the MapReduce phases. In this case, each intermediate result is persisted in the output phase and needs to be re-read in the input phase iteration.

2.2.4.2 Resilient Distributed Datasets

[Zah+12] Zaharia et al. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". 2012

Zaharia et al. [Zah+12] presented the concept of Resilient distributed Datasets (RDDs) in combination with an implementation called Spark[8]. RDDs are immutable distributed memory abstractions. While they can provide MapReduce functionality, they provide many more operations. In comparison with MapReduce, RDDs aim to provide a higher control and flexibility w.r.t. data locality and partitioning, data persistence and in-memory computations, re-use of intermediate results, and data transformations. RDDs are managed by a special worker, called driver, which can initiate distributed operations on them. RDD operations are divided into **transformations** (cf., table 2.1) and **actions** (cf., table 2.2). See [Zah+12] for an in-depth explanation of the individual operations, which is out of the scope of this thesis. While transformations are performed in a lazy fashion and define a new child RDD, actions actually launch

[8] `https://spark.apache.org/`

Table 2.1: Basic resilient distributed dataset transformations as given in [Zah+12]. Additional transformations of interest—which are not part of the publication but implemented in Spark—are shown in the second half of the table. O, T, U, V, W are value types and K is a key type. RDD[X] is a resilient distributed dataset of records with type X.

Name	Parameter	Input & Return Type
map	$f : \mathrm{T} \to \mathrm{U}$	$\mathrm{RDD[T]} \to \mathrm{RDD[U]}$
filter	$f : \mathrm{T} \to \mathbb{B}$	$\mathrm{RDD[T]} \to \mathrm{RDD[T]}$
flatMap	$f : \mathrm{T} \to \mathrm{seq[U]}$	$\mathrm{RDD[T]} \to \mathrm{RDD[U]}$
sample	fraction $\in \mathbb{Q}^{\geq 0, \leq 1}$	$\mathrm{RDD[T]} \to \mathrm{RDD[T]}$
groupByKey	$\emptyset$	$\mathrm{RDD[(K, V)]} \to \mathrm{RDD[(K, seq[V])]}$
union	$\emptyset$	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \to \mathrm{RDD[T]}$
join	$\emptyset$	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \to \mathrm{RDD[(K, (V, W))]}$
cogroup	$\emptyset$	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \to \mathrm{RDD[(K, (seq[V], seq[W]))]}$
crossProduct	$\emptyset$	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \to \mathrm{RDD[(T, U)]}$
mapValues	$f : \mathrm{V} \to \mathrm{W}$	$\mathrm{RDD[(K, V)]} \to \mathrm{RDD[(K, W)]}$
sort	comparator $\in f : (K, K) \to \{-1, 0, 1\}$	$\mathrm{RDD[(K, V)]} \to \mathrm{RDD[(K, V)]}$
partitionBy	partitioner $\in f : K \to \mathbb{N}^{>0}$	$\mathrm{RDD[(K, V)]} \to \mathrm{RDD[(K, V)]}$
repartition	numPartitions $\in \mathbb{N}^{>0}$	$\mathrm{RDD[(K, V)]} \to \mathrm{RDD[(K, V)]}$
mapPartitions	$f : \mathrm{seq[T]} \to \mathrm{U}$	$\mathrm{RDD[T]} \to \mathrm{RDD[U]}$
zipPartitions	RDD[O], $f : (\mathrm{seq[T]}, \mathrm{seq[O]}) \to \mathrm{seq[U]}$	$\mathrm{RDD[T]} \to \mathrm{RDD[U]}$
sampleExact	fraction $\in \mathbb{Q}^{\geq 0, \leq 1}$, withReplacement $\in \mathbb{B}$	$\mathrm{RDD[T]} \to \mathrm{RDD[T]}$

Table 2.2: Resilient distributed dataset actions. T and V are value types and K is a key type. RDD[X] is a resilient distributed dataset of records with type X. Non-RDD types are returned to the driver.

Name	Parameter	Input & Return Type
count	$\emptyset$	RDD[T] $\rightarrow \mathbb{N}$
collect	$\emptyset$	RDD[T] $\rightarrow$ seq[T]
reduce	$f : (T, T) \rightarrow T$	RDD[T] $\rightarrow T$
lookup	$k \in$ K	RDD[(K, V)] $\rightarrow$ seq[V]
save	path	not applicable (saves RDD to shared storage)

a computation. RDDs that are the result of a transformation remember their lineage, i.e., they link to the parent RDDs involved in the transformation. Consequentially, the application of several transformations leads to a directed acyclic graph (DAG) of operations that depend on each other. When an action is applied, the scheduler follows the lineage to derive tasks that can be executed by the workers.[9]

Similar to MapReduce, an initial RDD must be created by providing some data—e.g., reading from persistent storage—and individual records must be distributed among the workers to perform distributed operations. Spark groups the records into so-called partitions. Each partition contains several records and is worker local, i.e., cannot span multiple workers. Consequentially, the number of partitions of an RDD limits the worker-level concurrency. The partitioning of an RDD can be influenced directly with the `partitionBy` transformation. Furthermore, several other transformations influence the partitioning, e.g., `groupByKey`, `union`, and `join`. Other transformations, such as `map` and `filter` retain the RDDs partitioning. Dependencies in the lineage graph are classified into narrow and wide dependencies. Narrow dependencies are defined as dependencies, where each parent RDDs partition is used in at most one child partition. Wide dependencies are all other dependencies. Narrow dependencies are special from a distributed point of view because they can be pipelined on a single worker instance without communication over the network. A fitting RDD partitioning is thereby a key factor to reduce network usage and obtain data locality.

In many cases, intermediate results—emerging from the application of transformations—need to be accessed multiple times. In this case, it might be beneficial to remember and re-use the intermediate results instead of re-computing them. Similar to MapReduce, intermediate results can be stored on a shared persistent storage. However, in some cases—especially in the case of iterative algorithms with many iterations—persisting the data on a shared storage may cause a significant overhead w.r.t. time, disk usage, and network bandwidth. For this reason, RDDs provide a configurable local caching mechanism, accessible via the methods `persist` and `unpersist`. Similar to

[9] In fact the scheduler of Spark is more sophisticated. For example, it groups several operations to stages of narrow dependencies to increase worker local pipelining of transformations (cf., [Zah+12]).

the lazy evaluation of transformations, calling `persist` on an RDD does not trigger the computation. However, whenever the computation is caused by some action, the intermediate result is cached locally by the worker who performed the computation on a per partition basis. Caching modes are either in-memory, on a local storage, or a memory-storage hybrid, which only uses the local storage whenever the cache exceeds the local memory. The cache is evicted in a least recently used (LRU) fashion. In the case of eviction, RDDs know their lineage and the missing partition elements can be re-computed.

Without going into to much detail, it should be mentioned, that the Spark scheduler uses the above concepts of lineage, partitioning, and caching for fault tolerance. Whenever a worker fails (e.g., because of a hardware failure or an out-of-memory situation), Spark re-schedules the computation. Given the lineage of the RDD, the re-scheduled computations are performed using intact caches and partition elements whenever possible.

2.2.5 Pseudocode

Algorithms given in this thesis have pseudocode keywords and header information that are not part of common knowledge and will be explained in the following.

2.2.5.1 Header Information

The header of an algorithm always contains the input and output fields, which specify the parametrization and the return values of an algorithm. In addition, the header of an algorithm might specify a set of fixed parameters, which are assumed to be globally visible in the pseudocode. These parameters are assumed to express variants of an algorithm. Thus, passing them as parameters to the procedure would otherwise clutter the pseudocode. Furthermore, the header of the pseudocode may specify shared-memory variables for shared-memory parallelized algorithms. These variables are also assumed to have a global scope and are accessible from all concurrent processes. Finally, the header may contain other information, that is special to the context and is explained alongside the algorithms.

2.2.5.2 Parallel Pseudocode

Multiple keywords are used to express parallelism in pseudocode. Loops annotated with the keyword *in parallel* will be processed in parallel. It is assumed, that the loop condition can be evaluated for all parallel executions at the time the loop is entered. Thus, it is not allowed to have a condition, that depends on the result of the body of the loop. Additionally, the keyword *async* is used to execute a procedure asynchronously. Thus, the execution of the procedure is forked from the calling thread and the execution of the calling process is not blocked. Asynchronous procedures never have a return value and must write their results to shared-memory variables. If a parallel execution needs to wait until some condition is met, the keyword *wait until* is used. Furthermore, a synchronized block (keyword *synchronized*) is used to force

mutually exclusive executions of code regions. If a process tries to enter such a region while another process already executes it, the process waits until the other process leaves the region. A synchronized block can be parameterized with a key to group multiple code regions that need to be mutually exclusively executed. For example, a data structure with multiple operations may group all modifying operations, such that only a single processor can perform modifications at the same time.

2.3 Statistical Inference and Hypothesis Testing

Statistical inference describes the deduction of properties of a population from a sample. The field of statistical inference includes methods for hypothesis testing, point estimation, confidence intervals or sets, and others.

2.3.1 Notation

The sample space of a probabilistic experiment, i.e., the set of all possible outcomes, is denoted by Ω. For some σ-Algebra $\mathfrak{A} \subseteq \mathcal{P}(\Omega)$, a function $\mathbf{Pr} : \mathfrak{A} \to [0,1]$ is called probability measure if $\mathbf{Pr}(\Omega) = 1$ and $\mathbf{Pr}$ is σ-additive. Each element $A \in \mathfrak{A}$ is called an event. The triple $(\Omega, \mathfrak{A}, \mathbf{Pr})$ is then called a probability space. A measurable function $X : \Omega \to M$ which maps to some measurable space with set M and σ-Algebra $\mathfrak{B} \subseteq \mathcal{P}(M)$ is called a random variable if $X^{-1}(B) = \{\omega \in \Omega \mid X(\omega) \in B\}$ for all $B \in \mathfrak{B}$. In this thesis, random variables are always denoted in uppercase letters. A random variable is characterized by its probability distribution over M, which implies a probability measure $\mathbf{Pr}[X = B] := \mathbf{Pr}(X^{-1}(B))$ for every $B \in \mathfrak{B}$. Continuous random variables, i.e., $M = \mathbb{R}$, are uniquely quantified by their probability density function $f_X : \mathbb{R} \to \mathbb{R}^{\geq 0}$ with $\mathbf{Pr}[X \leq x] = \int_{-\infty}^{x} f_X(y)dy$. Discrete random variables are uniquely quantified by their probability mass function $f_X(x) = \mathbf{Pr}[X = x]$, for each $x \in M$. If the distribution of a random variable X follows a probability mass or density function f_X, it is also written as $X \backsim f_X$. The cumulative distribution function $F_X : \mathbb{R} \to [0,1]$ is defined as $F_X(x) := \mathbf{Pr}[X \leq x]$ for a continuous random variable X or a discrete random variable X with $M \subseteq \mathbb{R}$. The y-quantile of a probability density function can be calculated and is written as inverse function $F_X^{-1}(y)$ of the cumulative distribution function. For a random variable X, the expected value is denoted by μ_X, the variance by σ_X^2, and the standard deviation by σ_X. A random sample $\mathfrak{S} = (x_1, x_2, \ldots, x_N)$ is defined as observations of identically independent random (i.i.d.) variables $X_1, X_2, \ldots, X_N$ over Ω. During this thesis, such a random sample is sometimes treated as multiset $\{\!\{x_1, x_2, \ldots, x_N\}\!\}$, when the ordering of the sample is not relevant. In this manner, the notion of operations on multisets is also applicable to random samples, e.g., $|\mathfrak{S}| = N$.

2.3.2 Basic Distributions

A selection of distributions used throughout this thesis is introduced in the following.

2.3.2.1 Bernoulli Distribution

The Bernoulli distribution characterizes a discrete random variable X, that follows the probability mass function

$$f_X(x) = B_{1,p}(x) = \begin{cases} 1-p & \text{if } x = 0, \\ p & \text{if } x = 1 \end{cases}$$

for $x \in \{0, 1\}$ and a parameter $p \in [0, 1]$. A random variable that follows $B_{1,p}$ has a variance $\sigma^2 = p(1-p)$ and an expected value $\mu = p$. The event $x = 1$ is also called success. The event $x = 0$ is called failure.

2.3.2.2 Binomial Distribution

A binomial distribution describes the number of successes x of n repetitions of a Bernoulli experiment with parameter p. The probability mass function of a discrete random variable X, that follows the binomial distribution is

$$f_X(x) = B_{n,p}(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for $n \in \mathbb{N}^{>0}$, $p \in [0, 1]$, and $x \in \{0, \ldots, n\}$. A random variable that follows $B_{n,p}$ has a variance $\sigma^2 = np(1-p)$ and an expected value $\mu = np$.

2.3.2.3 Normal Distribution

The normal or Gaussian distribution characterizes a continuous random variable X, that follows the probability mass function

$$f_X(x) = \mathcal{N}_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

for the parameters $\mu \in \mathbb{R}$ (expected value) and $\sigma^2 \in \mathbb{R}^{>0}$ (variance). The probability mass function $\mathcal{N}_{0,1}(x)$ is called standard normal distribution or z-distribution. The cumulative distribution function of $\mathcal{N}_{0,1}$ is written $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy$.

2.3.2.4 Poisson Distribution

The Poisson distribution describes the number of events that fall into an interval of a fixed length t under the assumption that events occur with a constant rate λ that is normalized by t. The probability mass function of a discrete random variable X, that follows the Poisson distribution is

$$f_X(x) = \mathrm{Po}_\lambda(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x \in \mathbb{N}$, and $\lambda \in \mathbb{R}^{>0}$. A random variable that follows Po_λ has a variance $\sigma^2 = \lambda$ and an expected value $\mu = \lambda$.

2.3.3 Hypothesis Testing

Hypothesis testing is a method of statistical inference. The question asked in hypothesis testing is whether a statistical hypothesis is plausible under the observation of a sample from a population.

In the following a statistical test problem will be formalized by the assumptions of the test about the underlying sample $\mathfrak{S}$, the specification of a null hypothesis H_0 in combination with an alternative hypothesis H_1, a test statistic $T = f(\mathfrak{S})$, and a test decision rule based on the test statistic T. The goal of a statistical test is to decide whether H_0 can be rejected with enough confidence (i.e., a high probability) considering the test's assumptions to be true. In other words, whether H_1 holds with a high probability. A statistical test can have two types of error. A type I error is made if H_0 is rejected by the test while H_0 is true. A type II error is made if H_0 is not rejected, but H_0 is false. These two types of errors are dependent. For a fixed set of test parameters, only one error can be bounded to a fixed probability. Conventionally, it is the type I error that is bounded by a significance level α, which has to be specified a priori to the test application. More precisely, α is defined as the maximum probability of a type I error. A fixed α implies a set of values of T for which H_0 can be rejected. This set is called critical or rejection region. It is common practice that the result of a statistical test is not given in form of a binary decision, but as a p-value. The p-value is the most extreme (i.e., lowest) value of α that would result in a rejection of H_0. Hence, the p-value is more precise regarding the confidence of a possible rejection of H_0.

2.3.4 Multiple Hypothesis Testing

Statistical experiments executing multiple hypothesis tests suffer from the so-called multiple testing problem. The problem states, that with more experiments the likelihood of a statistical error increases. This includes type I and type II errors. The family-wise error rate (FWER) focuses on the type 1 error and is defined as the probability of conducting at least one type I error in a family of tests. Consequentially, if the FWER should be controlled by some significance level α, the individual tests in the family must be executed with an adjusted (stricter) significance level. This adjustment is called multiply hypothesis testing correction.

2.3.4.1 Simple Bonferroni Correction

The first-order Bonferroni Inequality is used for several multiple hypothesis testing correction methods [Sha95]. The simple Bonferroni method is a conservative testing correction, that does not require any assumptions about the p-value distribution or dependencies between individual tests. To control the FWER of a family with cardinality n and a significance level α, a corrected significance level of $\frac{\alpha}{n}$ is applied to each individual test. Consequentially, the method requires an a priori knowledge about n.

[Sha95] Shaffer, "Multiple hypothesis testing". 1995

2.3.5 Basic Statistical Tests

A selection of statistical tests used throughout this thesis is introduced in the following. This content of this section is based on [TK14].

[**TK14**] Taeger and Kuhnt, *Statistical Hypothesis Testing with SAS and R*. 2014

2.3.5.1 Binomial Test

The binomial test decides whether the proportion p of a population differs from a value p_0.

Assumptions: The random sample $\mathfrak{S}$ is an observation of N independent random variables $X_i \backsim B_{1,p}$, $1 \leq i \leq N$ for some unknown $p \in [0,1]$. Thus, $X = \sum_{i=1}^{N} X_i \backsim B_{N,p}$.

Hypotheses:

(i) $H_0 : p = p_0, H_1 : p \neq p_0$

(ii) $H_0 : p \leq p_0, H_1 : p > p_0$

(iii) $H_0 : p \geq p_0, H_1 : p < p_0$

Test statistic:

$$t = \sum_{x \in \mathfrak{S}} x$$

Test decision: Let $Y \backsim B_{N,p_0}$. Reject H_0 if

(i) $F_Y(t) \leq \frac{\alpha}{2} \vee F_Y(t) \geq 1 - \frac{\alpha}{2}$

(ii) $F_Y(t) \geq 1 - \alpha$

(iii) $F_Y(t) \leq \alpha$

2.3.5.2 Two-Sample Approximate Binomial Test on Equality

The two-sample approximate binomial test on equality is a large sample test on the proportions p_1 and p_2 of two populations. It tests whether p_1 differs from p_2. The binomial distribution can be approximated by the normal distribution w.r.t. the central limit theorem. For this reason, this test is also known as the two-sample z-test with pooled variance.

Assumptions: Each random sample $\mathfrak{S}_j$ for $j \in \{1,2\}$ is an observations of N_j independent random variables $X_{j,i} \backsim B_{1,p_j}$, $1 \leq i \leq N_j$ for some unknown $p_j \in [0,1]$. Thus, $X_j = \sum_{i=1}^{N} X_{j,i} \backsim B_{N,p_j}$. This is a large sample test. The absolute frequencies of observations of success and failure must be larger than 5 in each sample, i.e., $N_1 p_1 > 5 \wedge N_1(1-p_1) > 5 \wedge N_2 p_2 > 5 \wedge N_2(1-p_2) > 5$ [TK14] for a sufficient approximation ratio.

[**TK14**] Taeger and Kuhnt, *Statistical Hypothesis Testing with SAS and R*. 2014

Hypotheses:

(i) $H_0 : p_1 = p_2, H_1 : p_1 \neq p_2$
(ii) $H_0 : p_1 \leq p_2, H_1 : p1 > p_2$
(iii) $H_0 : p_1 \geq p_2, H_1 : p1 < p_2$

Test statistic:

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{N_1} + \frac{1}{N_2})}}, \text{ with} \qquad \hat{p}_j = \frac{\sum_{x \in \mathfrak{S}_j} x}{N_j}, \qquad \hat{p} = \frac{\hat{p}_1 N_1 + \hat{p}_2 N_2}{N_1 + N_2}.$$

Test decision: Reject H_0 if

(i) $z < \Phi^{-1}(\frac{\alpha}{2})$
(ii) $z > \Phi^{-1}(1-\alpha)$
(iii) $z < \Phi^{-1}(\alpha)$

p-value:

(i) $p = 2\Phi(-|z|)$
(ii) $p = 1 - \Phi(z)$
(iii) $p = \Phi(z)$

2.4 Submodular Set Function Maximization

Combinatorial optimization (search) problems have to find a subset with optimal value of objects from a superset. An objective function that assigns a value to such a subset is called set function.

Definition 2.2 (Marginal Gain of a Set Function). *Let A be a finite set and $f : \mathcal{P}(A) \to \mathbb{R}$ a set function over A. Then, the discrete derivative*

$$\Delta_f(x|X) := f(X \cup \{x\}) - f(X)$$

for some $X \subseteq A$ and $x \in A$ is called marginal gain of x w.r.t. X over A [cf., Bad+14].

[Bad+14] Badanidiyuru et al., "Streaming Submodular Maximization: Massive Data Summarization on the Fly". 2014

In other words, the marginal gain describes the additional value an object $x \in A$ has when it is added to a subset of A. A set function is said to be monotone increasing, if $\forall X \subseteq A, x \in A \setminus X : \Delta_f(x|X) \geq 0$.

Definition 2.3 (Submodular Set Function). *Let A be a finite set. A set function $f : \mathcal{P}(A) \to \mathbb{R}$ over A is called submodular iff one of the following*

equivalent conditions holds:

$$\forall X, Y \subseteq A : f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \tag{2.1}$$
$$\forall X \subseteq Y \subseteq A, x \in A \setminus Y : \Delta_f(x|X) \geq \Delta_f(x|Y) \tag{2.2}$$
$$\forall X \subseteq A, x, y \in A \setminus X : f(X \cup \{x\}) + f(X \cup \{y\}) \geq f(X \cup \{x, y\}) + f(X) \tag{2.3}$$
$$\forall X, Y \subseteq A : f(Y) \leq f(X) + \sum_{z \in Y \setminus X} \Delta_f(z|X) - \sum_{z \in X \setminus Y} \Delta_f(z|X \cup Y \setminus \{z\}) \tag{2.4}$$

[cf., NWF78]

[**NWF78**] Nemhauser, Wolsey, and Fisher, "An analysis of approximations for maximizing submodular set functions—I". 1978

Some classical optimization problems are known to have submodular objective functions, e.g., the maximum coverage problem. Additionally, a large number of objectives occurring in data mining and machine learning problems are known to be submodular [Bad+14].

[**Bad+14**] Badanidiyuru et al., "Streaming Submodular Maximization: Massive Data Summarization on the Fly". 2014

Problem 2.1 (MAX-CSSF). Constrained Submodular Set Function Maximization

Input: *A submodular set (utility) function f over a finite set A and a positive cardinality constraint k.*

Task:

$$\arg\max_{\substack{X \subseteq A, \\ |X| \leq k}} f(X)$$

The constrained maximization problem is of special interest in this thesis (cf., chapter 4). Without further restrictions, problem 2.1 (MAX-CSSF) is known to be NP-hard [Fei98]. However, Nemhauser, Wolsey, and Fisher [NWF78] presented a greedy approximation (cf., algorithm 1) for the problem.

[**Fei98**] Feige, "A Threshold of ln n for Approximating Set Cover". 1998

[**NWF78**] Nemhauser, Wolsey, and Fisher "An analysis of approximations for maximizing submodular set functions—I". 1978

Algorithm 1: Greedy approximation for problem 2.1 (MAX-CSSF)

Input: finite set A, positive cardinality constraint $k \in \mathbb{R}^{>0}$, submodular set function f
Output: solution set X

```
1 X_0 ← ∅;
2 i ← 1;
3 while i ≤ k and max_{x∈A\X_{i-1}} Δ_f(x|X_{i-1}) > 0 do
4     X_i ← X_{i-1} ∪ {arg max_{x∈A\X_{i-1}} Δ_f(x|X_{i-1})};
                                          ▶ ties settled arbitrary
5     i ← i + 1;
6 return X_{i-1};
```

Line 4 of algorithm 1 successively adds elements with maximum marginal gain to the solution set until the cardinality constraint is reached or no more element with strictly positive marginal gain exists. The second exit criterion is valid since marginal gains are monotonically decreasing with i (cf., eq. (2.2)).

Lemma 2.1. *For a monotone increasing submodular function f with $f(\emptyset) = 0$, algorithm 1 yields a solution with approximation ratio $1-(1-\frac{1}{k})^k \geq (1-\frac{1}{e}) \approx 0.63$ for problem 2.1 (MAX-CSSF).*

[NWF78] Nemhauser, Wolsey, and Fisher "An analysis of approximations for maximizing submodular set functions—I". 1978

Proof. The original proof of Nemhauser, Wolsey, and Fisher [NWF78] covers the case of negative marginal gains and $f(\emptyset) \neq 0$, while lemma 2.1 only states the special case of positive marginal gains (i.e., monotone increasing submodular functions) and $f(\emptyset) = 0$. Therefore, the following proof will start with the general case and restrict itself to the special case its second part.

Let $\text{OPT} = f(T)$ be the optimal solution value for a given instance of problem 2.1 (MAX-CSSF) with submodular function f over set A and cardinality constraint k. Let $\textsc{Greedy} = f(\emptyset) + \delta_0 + \cdots + \delta_{k^*-1}$ be the value of a particular result of algorithm 1 for the same instance where $k^* \leq k$ is the solution size after termination and δ_i is the marginal gain of the $(i+1)$th element added by the greedy approximation (i.e., $\delta_i = \max_{x \in A \setminus X_i} \Delta_f(x|X_i)$). Furthermore, let the marginal gain be bounded from below by $-\theta \leq 0$, i.e., $\forall Y \subseteq A, y \in A \setminus Y : \Delta_f(y|Y) \geq -\theta$.

The proof is based on the submodular set function definition (2.4). Using the bounded marginal gain the right sum of the inequality can bounded from below by $-|X \setminus Y|\ \theta$, resulting in the eq. (2.5).

$$\forall X, Y \subseteq A : f(Y) \leq f(X) + \sum_{z \in Y \setminus X} \Delta_f(z|X) + |X \setminus Y|\ \theta \tag{2.5}$$

Using this inequality, it is possible to relate OPT to the value of partial solutions of the greedy approximation. Let t be an iteration of the greedy approximation, then the following inequalities hold.

$$\begin{aligned} \text{OPT} &\leq f(\emptyset) + \sum_{i=0}^{t-1} \delta_i + k\delta_t + t\theta, && 0 \leq t \leq k^* - 1 \\ \text{OPT} &\leq f(\emptyset) + \sum_{i=0}^{k^*-1} \delta_i + k^*\theta, && \text{if } k^* < k \end{aligned} \tag{2.6}$$

The first inequality is easy to comprehend considering (a) $\forall x \in A \setminus X_t : \Delta_f(x|X_t) \leq \delta_t$ (monotonically decreasing marginal gains), (b) $\delta_t \geq 0$ (otherwise the the algorithm would have terminated before), (c) $|X_t \setminus T| \leq t$ (since $|X_t| = t$), and (d) $|T \setminus X_t| \leq k$. The second inequality follows by setting $t = k^* - 1$ and $\delta_{k^*} \leq 0$, since the algorithm only terminates prior to k iterations if $\max_{x \in A \setminus X_{i-1}} \Delta_f(x|X_{i-1}) \leq 0$.

For the special case of $\theta = 0$, the second inequality implies optimal solution values for the greedy algorithm in case of $k^* < k$, since $\text{OPT} - f(\emptyset) \leq \sum_{i=0}^{k^*-1} \delta_i$. Thus, the

sum of the marginal gains for the optimal solution is (smaller or) equal to the sum of the marginal gains of the greedy approximation.

Since lemma 2.1 only states the special case $\theta = 0$, the following proof will be a simplified version of the proof presented in the original paper utilizing the above stated fact that the greedy algorithm produces optimal solutions for $k^* < k$. Thus is is sufficient to prove the approximation ratio for $k^* = k$ in the following.

The inequalities (2.6) eliminate sequences of marginal gains that are not possible to obtain as a result of the greedy algorithm s.t., k, k^*, θ, and $f(\emptyset)$ in relation to OPT. With this insight, Nemhauser, Wolsey, and Fisher [NWF78] derive a family of bounds depending on θ for Greedy by minimizing the sum of marginal gains s.t., inequalities (2.6). For this task consider the following linear program (simplified for $\theta = 0$). Let $j < k$ be a a positive integer and b a positive real number.

[NWF78] Nemhauser, Wolsey, and Fisher "An analysis of approximations for maximizing submodular set functions—I". 1978

$$\begin{aligned} &\min \sum_{i=0}^{j} x_i \\ &\sum_{i=0}^{t-1} x_i + k x_t \geq 1 - (k+t)b \qquad 0 \leq t \leq j \end{aligned} \tag{2.7}$$

The constraints of the linear program (2.7) exactly match the first k^* inequalities in (2.6) if $b = \frac{\theta}{\text{OPT} - f(\emptyset) + k\theta}$ and $x_i = \frac{\delta_i}{\text{OPT} - f(\emptyset) + k\theta}$. For $P(b) := kb + \min \sum_{i=0}^{j} x_i$ Nemhauser, Wolsey, and Fisher [NWF78] show that the following holds.

$$P(b) = 1 + (k - j - 1)b - \left(\frac{k-1}{k}\right)^{j+1} \tag{2.8}$$

For $j + 1 = k^* = k$, equation (2.9) is obtained.

$$P(b) \geq 1 + (k-k)b - \left(\frac{k-1}{k}\right)^{k} = 1 - \left(\frac{k-1}{k}\right)^{k} \tag{2.9}$$

Substituting the above described values for b and x_i in the definition of $P(b)$, and using the fact $\forall j < k : \sum_{i=0}^{j} \delta_i \leq \text{Greedy} - f(\emptyset)$ the following inequality is obtained.

$$P(b)(\text{OPT} - f(\emptyset) + k\theta) \leq k\theta + \text{Greedy} - f(\emptyset) \tag{2.10}$$

Substituting equation (2.9) into (2.10) with $\theta = 0$ and $f(\emptyset) = 0$ (cf., lemma 2.1) results in equation (2.11).

$$\begin{aligned} &\frac{\text{OPT} - \text{Greedy}}{\text{OPT}} \leq \left(\frac{k-1}{k}\right)^{k} \\ \Leftrightarrow \qquad &\frac{\text{Greedy}}{\text{OPT}} \geq 1 - \left(\frac{k-1}{k}\right)^{k} \end{aligned} \tag{2.11}$$

□

The approximation ratio cannot be improved in the general case. Feige [Fei98] has proven, that k-cover—a special case of problem 2.1 (MAX-CSSF)—cannot be approximated in polynomial time within a ratio of $(1 - \frac{1}{e} + \epsilon)$, unless $P = NP$.

[Fei98] Feige "A Threshold of ln n for Approximating Set Cover". 1998

Problem 2.2 (MAX-Cov). Maximum Coverage

Input: *A finite family of sets F over U, a positive cardinality constraint k*

Task:

$$\underset{\substack{X \subseteq F,\\ |X| \leq k}}{\arg\max} \left| \bigcup_{S \in X} S \right|$$

The NP-hard maximum coverage problem in is a special case of problem 2.1 (MAX-CSSF) and $f(X) := \left|\bigcup_{S \in X} S\right|$ is a submodular set function. It is of special interest in chapter 4.

Several distributed and streaming approximation algorithms have been proposed to overcome the strict serial nature of the above-mentioned approximation algorithm.

[Mir+13] Mirzasoleiman et al. "Distributed Submodular Maximization: Identifying Representative Elements in Massive Data". 2013

Mirzasoleiman et al. [Mir+13] presented a two-round MapReduce algorithm called GreeDi, which basically runs the approximation algorithm from Nemhauser, Wolsey, and Fisher independently on m nodes and combines the results to a global solution. The algorithm is a $\frac{(1-e^{-\frac{\kappa}{k}})(1-e^{-\frac{l}{k}})}{\min(m,k)}$-approximation, where l is the final global solution size and κ is the solution size on each node. Thus, l is a relaxation of the original cardinality constraint in order to increase the solution's utility. For the special case of λ-Lipschitz continuous objective functions and further density assumptions, they showed a better approximation ratio of $(1 - e^{-\frac{\kappa}{k}})(f(X_{\text{central}}) - \lambda ak)$, where a is a parameter of their density definition and X_{central} is the solution of the centralized greedy algorithm of Nemhauser, Wolsey, and Fisher. Mirrokni and Zadimoghaddam [MZ15] published another constant round, MapReduce algorithm with an expected approximation ratio of 0.545. They also follow the general idea to combine local solutions to a global one. However, they have more sophisticated local algorithms, which need to be β-nice. β-niceness basically assumes, that the local algorithms produce results for which the marginal gain of a non-selected object is not more than β times the average utility value of the selected objects. Badanidiyuru et al. [Bad+14] presented the first efficient streaming algorithm which is a constant-factor approximation. The algorithm is called Sieve-Streaming and provides a $\frac{1}{2} - \epsilon$ approximation by greedily selecting objects that exceed the threshold $\beta\frac{\text{OPT}}{k}$. Since the optimum is unknown, they maintain multiple selections for different thresholds, which replace the single optimum. The thresholds are chosen such that they differ by a constant factor, which limits the distance of the optimum to the closest threshold. The highest value selection of any threshold is then selected as output.

[MZ15] Mirrokni and Zadimoghaddam "Randomized Composable Core-sets for Distributed Submodular Maximization". 2015

[Bad+14] Badanidiyuru et al. "Streaming Submodular Maximization: Massive Data Summarization on the Fly". 2014

A common problem of the distributed algorithms is the dependency of some submodular functions $f : \mathcal{P}(A) \to \mathbb{R}$ on A itself. In other words, the computation of $f(X)$ for some set X cannot be performed in an isolated black box. For example, for some clustering tasks it is necessary to compute the distance of all objects in A to the selected objects $X \subseteq A$ in order to compute the utility of X. A counterexample is the maximum coverage problem, where the cardinality of the set union can be

computed independently of A. If A does not fit in the memory of a distributed worker, a dependency on A imposes an additional computational challenge.

Definition 2.4 (Additively Decomposable Submodular Set Function). *A submodular set function $f : \mathcal{P}(A) \to \mathbb{R}$ is said to be additively decomposable over a ground set B iff there exists a submodular function f_o for each object $o \in A$ and*

$$\forall X \subseteq A : f(X) = \sum_{o \in B} f_o(X)$$

holds.

Definition 2.4 distinguishes between the sets A and B, i.e., the domain of f and the set over which the function is decomposable over. Nevertheless, in many cases (e.g., the clustering example above) A and B are actually the same set. Given such an additively decomposable submodular set function, one can distribute the work among several distributed workers to compute the value of a set. Additionally, Badanidiyuru et al. [Bad+14] proposed a sampling method for submodular set function maximization that utilized the decomposition w.r.t. to definition 2.4 with $A = B$ on a random sample $S \subseteq A$ to overcome the dependency problem in the streaming setting. Given on a sample of size $\frac{2\log(\frac{2}{\delta})k^2 + 2k^3\log(|A|)}{\epsilon^2}$, they presented a variant of Sieve-Streaming that calculates a solution with approximation ratio $(\frac{1}{2} - \epsilon)(\mathrm{OPT} - \epsilon)$ with probability $1 - \delta$.

[Bad+14] Badanidiyuru et al. "Streaming Submodular Maximization: Massive Data Summarization on the Fly". 2014

2.5 Graphs

Definition 2.5 (Simple Graph). *A simple graph is a tuple $G = (V, E)$, comprising a finite set of vertices $V(G) = V$ and a finite set of edges $E(G) = E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$*

If not explicitly stated otherwise, the term graph will refer to simple graphs. Two vertices $u, v \in V$ are said to be *adjacent* to each other if $e = \{u, v\} \in E$. Furthermore, the node u (or v respectively) and the edge e are said to be *incident* to each other. $|G|$ is used as a short term for $|V(G)| + |E(G)|$. A multiset of graphs $\mathcal{G} := \{\!\{G_1, \ldots, G_n\}\!\}$ is also called *graph dataset*. The infinite set of all simple graphs is written $\mathcal{S}$. A sequence of consecutive adjacent vertices $(v_1, \ldots, v_n)$ is called walk of length $n - 1$. Walks with unique vertices are called paths. A graph is said to be connected if there exists a path between any two vertices and disconnected otherwise. A contraction of two vertices u, v to a vertex w in a Graph $G = (V, E)$ results in a graph $G' = (V', E')$ with $V' := V \setminus \{u, v\} \uplus \{w\}$ and $E' := (E \setminus \{\{x, y\} \in E \mid \{x, z\} \cap \{u, v\} = \emptyset\}) \cup \{\{w, x\} \mid x \in V' \wedge x \text{ is adjacent to } u \text{ or } v \text{ in } G\}$. Each edge $e = \{w, x\}$, may be associated to multiple edges ($e' = \{u, x\}$ and $e'' = \{v, x\}$) in E. It is said, that e' and e''

are merged to e by the contraction. The empty set symbol is used as a shortcut for the empty graph $\emptyset := (\emptyset, \emptyset)$.

Definition 2.6 (Subgraph). *Let G and H be graphs. Then, G is a subgraph of H, if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$.*

If G is a subgraph of H, it is written $G \subseteq H$. $G \subset H$ is a short form for $|G| < |H| \wedge G \subseteq H$. If G is a subgraph of H, H is also termed *supergraph* of G. All subgraphs of a graph G are written as $G_{\subseteq} := \{(V', E') \mid V' \subseteq V(G), E' \subseteq E(G)\}$. The subgraphs of a graph dataset $\mathcal{G}$ are denoted as $\mathcal{G}_{\subseteq} := \cup_{G \in \mathcal{G}} G_{\subseteq}$. A subgraph G of H is said to be (vertex) induced if $E(G) = \{\{u, v\} \in E(H) \mid u, v \in V(G)\}$

Definition 2.7 (Graph Isomorphism). *Let G and H be two graphs. An bijection $\psi : V(G) \to V(H)$ is called isomorphism between G to H, if*

$$\forall u, v \in V(G) : \{u, v\} \in E(G) \Leftrightarrow \{\psi(u), \psi(v)\} \in E(H).$$

The existence of a graph isomorphism between a graph G to a graph H is written $G \simeq H$. The graph isomorphism relationship is an equivalence relation. An isomorphism $\psi : V(G) \to V(G)$ between the vertices of a single graph G is called automorphism. The decision problem "Are two graphs isomorphic?" is one of the few problems for which it is unknown whether it is NP-complete or polynomial-time solvable [GJ79].

[GJ79] Garey and Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979

Definition 2.8 (Subgraph Isomorphism). *Let G and H be two graphs. An injection $\psi : V(G) \to V(H)$ is called subgraph isomorphism from G to H, if*

$$\forall u, v \in V(G) : \{u, v\} \in E(G) \Rightarrow \{\psi(u), \psi(v)\} \in E(H).$$

The existence of a subgraph isomorphism from a graph G to a graph H, is written $G \sqsubseteq H$. $G \sqsubset H$ is a short form for $|G| < |H| \wedge G \sqsubseteq H$. Both relations are transitive and $\sqsubseteq$ is reflexive. A subgraph isomorphism can be seen as an isomorphism from a G to a subgraph H' of H, where $V(H')$ is the image of ψ, i.e., $G \sqsubseteq H \Leftrightarrow \exists H' : G \simeq H' \subseteq H$. A subgraph isomorphism from a graph G to a graph H is also called embedding of G into H. The decision problem "Does a subgraph isomorphism between two graphs exist?" is known to be NP-complete even when limited to quite simple graph classes such as outerplanar graphs [Sys82].

[Sys82] Sysło, "The subgraph isomorphism problem for outerplanar graphs". 1982

Definition 2.9 (Induced Subgraph Isomorphism). *Let G and H be two graphs. An injection $\psi : V(G) \to V(H)$ is called induced subgraph isomorphism from G to H, if*

$$\forall u, v \in V(G) : \{u, v\} \in E(G) \Leftrightarrow \{\psi(u), \psi(v)\} \in E(H).$$

An induced subgraph isomorphism can be also seen as an isomorphism from a graph G to a vertex-induced subgraph H' of H, where $V(H')$ is the image of ψ, i.e., $G \sqsubseteq H \Leftrightarrow \exists H' : G \simeq H' \subseteq_{\text{induced}} H$.

Definition 2.10 (Common Subgraph Isomorphism). *Let G and H be two graphs. Then, an isomorphism ψ between $G' \subseteq G$ and $H' \subseteq H$ is called common subgraph isomorphism between G and H.*

Any graph $C \simeq G' \simeq H'$ is called common subgraph of G and H. Any isomorphism between any common subgraph C and G' or H' induces a subgraph isomorphism from C to G or H. In other words, a common subgraph C of two graphs G and H is a graph for which $H \sqsupseteq C \sqsubseteq G$ holds.

Definition 2.11 (Labeled Graph). *Let $G = (V, E)$ be a graph. G is called a labeled graph, if it is equipped with an injective labeling function $\Gamma : V \uplus E \to \mathcal{L}$ that maps each vertex and each edge to a finite set of labels $\mathcal{L} = \mathcal{L}_V \cup \mathcal{L}_E$, such that $\forall v \in V : \Gamma(v) \in \mathcal{L}_V$ and $\forall e \in E : \Gamma(e) \in \mathcal{L}_E$.*

A labeled graph G is also written as triple $G = (V, E, \Gamma)$. Let G and H be two labeled graphs equipped with the label functions Γ_G and Γ_H. Then, a graph isomorphism ψ between G and H or a subgraph isomorphism ψ from G to H is said to be *label preserving*, if (a) $\forall v \in V(G) : \Gamma_G(v) = \Gamma_H(\psi(v))$ and (b) $\forall \{u, v\} \in E(G) : \Gamma_G(\{u, v\}) = \Gamma_H(\{\psi(u), \psi(v)\})$. In this thesis, it is assumed that all (sub)graph isomorphisms of labeled graphs are label preserving, if not explicitly mentioned otherwise.

2.6 Graph Data Mining

Graph data mining is a special field in data mining, which focuses on graph data. In this thesis, we will focus on graph data in the form of multisets of graphs, i.e., graph datasets as defined in section 2.5. In more general terms, graph mining also includes the analysis of other graph datasets, such as networks or single graphs.

Definition 2.12 (Data Mining). *Data mining is a subtask of the knowledge discovery process in databases. The task is to enumerate (non-trivial) patterns or models over the data. [FPS96]*

[**FPS96**] Fayyad, Piatetsky-Shapiro, and Smyth, "From data mining to knowledge discovery in databases". 1996

Data mining is therefore an intermediate step in order to extract knowledge or insights from data, which can be used for further reasoning about or interpretation of the data. Throughout the literature, there are numerous methods, that fall under the data mining task. Aggarwal and Han [AH14] identify four main data mining super problems: clustering, classification, outlier analysis, and frequent pattern mining. Other sources, e.g., Aggarwal and Wang [AW10] and Fayyad, Piatetsky-Shapiro, and Smyth [FPS96], mention multiple other methods, such as summarization, regression, discriminative pattern or feature mining, and dense subgraph mining.

[**AH14**] Aggarwal and Han Frequent Pattern Mining. 2014

[**AW10**] Aggarwal and Wang Managing and Mining Graph Data. 2010

2.6.1 Graph Pattern Mining

In this thesis, a graph pattern is an identifier, that serves as a template for structural features or commonalities of graphs in the sense of isomorphism or subgraph isomorphism relationship. Thus, the matter of interest is the enumeration of structural features or commonalities that occur in the underlying dataset. A formal definition is given below.

2.6.1.1 Graph Pattern Space

> **Definition 2.13 (Graph Pattern Space).** *Let $\mathcal{S}$ be the infinite set of all simple graphs. Then, $\mathcal{S}/\simeq$ is called the graph pattern space and each equivalence class in $\mathcal{S}/\simeq$ is called graph pattern.*

The above definition is closely related to the concept of a *complete graph invariant*. A complete graph invariant is a function I on $\mathcal{S}$ that fulfills the condition $H \simeq G \Leftrightarrow I(G) = I(H)$. Since $\simeq$ is an equivalence relation, each graph pattern $P \in \mathcal{S}/\simeq$ can be uniquely identified by a single element of the equivalence class. With a fixed representative function ρ of $\mathcal{S}$ under $\simeq$ all graphs $G \in P$ can be mapped to a unique graph $H \in P$ via $C(G) := \rho([G])$. Then, C is a complete graph invariant and the graph $H = \rho(G)$ is called canonical form of G and representative graph of P. The representative set $[\mathcal{S}]_{\simeq}$ for some fixed representative function ρ is denoted as the *representative or canonical pattern space*. For the sake of simplicity graph patterns will be depicted by representative graphs in figures and examples.

The subgraph isomorphism relationships $\sqsubseteq$ and $\sqsubset$ can be naturally extended to graph patterns since the subgraph isomorphism relationship is preserved under the graph isomorphism. Thus, for any $G, G', H \in \mathcal{S}$ it holds, that $G \sqsubseteq H \wedge G \simeq G' \Leftrightarrow G' \sqsubseteq H$ (the same is true for $\sqsubset$). For this reason, the relations $\sqsubseteq$ and $\sqsubset$ are also used for graph patterns and mixed forms of graph patterns and graphs in the following. Since a common subgraph can be defined over the subgraph isomorphism relation (cf., definition 2.10), the term common subpattern is used in this thesis as the extension of this concept to graph patterns. The concept of connectivity is also preserved under the graph isomorphism. The size of a pattern P can be measured in terms of vertices, edges, or the sum of both, i.e., ($|V(\rho(P))|$, $|E(\rho(P))|$, or $|\rho(P)|$) given an arbitrary representative function ρ. Since the pattern size is independent of the representative function, the simplified notions $|V(P)|$, $|E(P)|$, and $|P|$ will be used for patterns in analogy to the notation for graphs.

The pattern space can be partially ordered with the help of the subgraph isomorphism relation $\sqsubseteq$, i.e., $(\mathcal{S}/\simeq, \sqsubseteq)$ forms a poset. The poset $(\mathcal{S}/\simeq, \sqsubseteq)$ can be ranked by the pattern size, such that rank level k contains all patterns of size k. The rank definition thereby depends on the used measure of size, i.e., vertices, edges or the sum of both. An example poset for a connected labeled graph pattern space is shown in fig. 2.1. Furthermore, many graph mining problems (cf., section 2.6.1.3) do not consider the complete infinite graph pattern space, but only the finite patterns that are contained in the given problem instance in the form of subgraphs. Thus, for any graph

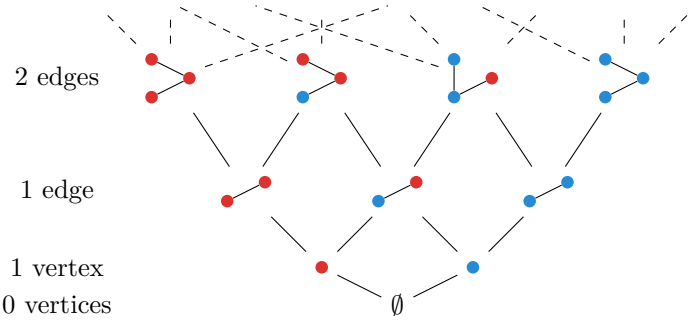


Figure 2.1: Hasse diagram of the pattern space of connected labeled graph patterns with $|\mathcal{L}_V| = 2$ partially ordered by the subgraph isomorphism relation $\sqsubseteq$. Vertex colors indicate vertex labels.

dataset $\mathcal{G}$ the *subgraph pattern space of* $\mathcal{G}$ is defined as $\mathcal{G}_{\sqsubseteq} := \{P \in \mathcal{S}/\simeq \mid P \cap \mathcal{G}_{\subseteq} \neq \emptyset\}$. The notation $[\mathcal{G}]_{\sqsubseteq} := [\mathcal{G}_{\subseteq}]_{\simeq}$ is used for the *representative or canonical subgraph pattern space* of a graph dataset $\mathcal{G}$.

The above definitions naturally extend to labeled graphs using the label preserving (sub)graph isomorphism. The extension to labeled graphs is also possible for all following problem definitions over the (sub)graph pattern space(s), if not explicitly mentioned otherwise. In this thesis, problem definitions for labeled graphs are omitted in all cases which do not require special handling of labels besides using a label preserving (sub)graph isomorphism.

2.6.1.2 Size of the Graph Pattern Space

The number of connected patterns grows exponentially with the size of the pattern, regardless if vertices, edges, or the sum of both are considered as reference. The same is true for a subgraph pattern space in the worst case. Even the subset of frequent subgraph patterns (cf., section 2.6.2) may grow exponentially [JCZ13].

Harary [Har55] presented recursive formulas to compute the number of unlabeled non-isomorphic graphs (i.e., graph patterns) with n vertices and distinguished undirected/directed, rooted/unrooted, and connected/disconnected graphs. However, no asymptotic bounds are given. Table 2.3 lists the computed numbers for connected simple non-isomorphic graphs up to $n = 19$.

Lupanov [Lup59; Lup62, (Russian)] (see [TK70] for an English summary) gave an asymptotic bound $G(n)$ for the number of unlabeled connected simple graph patterns with n edges:

$$G(n) = \left(\frac{2}{e}\frac{n}{\log^2 n}\gamma(n)\right)^n$$

[JCZ13] Jiang, Coenen, and Zito, "A survey of frequent subgraph mining algorithms". 2013

[Har55] Harary "The number of linear, directed, rooted, and connected graphs". 1955

[Lup59] Lupanov "Asymptotic estimates of the number of graphs with n edges". 1959

[Lup62] Lupanov "An asymptotic estimate of the number of graphs and networks with n edges," 1962

[TK70] Turner and Kautz, "A Survey of Progress in Graph Theory in the Soviet Union". 1970

Table 2.3: Number of connected simple graph patterns with n vertices [SI21, A001349]

n	Number of Patterns with n Vertices
0	1
1	1
2	1
3	2
4	6
5	21
6	112
7	853
8	11117
9	261080
10	11716571
11	1006700565
12	164059830476
13	50335907869219
14	29003487462848061
15	31397381142761241960
16	6396956011322517617627 7
17	24587183168208402651952856 8
18	17873317252488990888902005765 80
19	24636021429399867655322650759681644

where

$$\frac{2 \log \log n}{\log n} \lesssim \gamma(n) - 1 \lesssim \frac{\log \log n}{\log n}$$

Clearly, the formulas of Harary; Lupanov also applies to the number of subgraphs of a complete graph. Thus, given a maximum number of vertices, the unlabeled subgraph pattern space is not smaller than the general pattern space in the worst case. Since graphs with a single label can be interpreted as unlabeled graphs, the number of labeled connected graph patterns exceeds the number of unlabeled graphs w.r.t. to their size.

2.6.1.3 Generalized Graph Pattern Mining Problems

Graph pattern mining problems are defined to enumerate graph patterns of the subgraph pattern space in this thesis. The limitation to subgraph patterns is quite common in literature since many of the graph pattern mining problems have a frequency bounded search space (cf., section 2.6.2). As a consequence, these problems are intrinsically limited to subgraph patterns. Furthermore, this limitation has a clear relation to the size of the search space, which is usually beneficial in terms of computational complexity. Nevertheless, it should be mentioned, that there are several valid use cases outside this boundary, such as finding common supergraphs or generalized median graphs (i.e., the median graph is selected from a superset of a dataset). For example, Ferrer et al. [Fer+09] used generalized median graphs for clustering purposes. Throughout this thesis, graph pattern mining problems are furthermore limited to connected graph patterns if not explicitly stated otherwise.

[Fer+09] Ferrer et al. "Graph-Based k-Means Clustering: A Comparison of the Set Median versus the Generalized Median Graph". 2009

In the following, we will distinguish between the subgraph pattern enumeration and optimization problem as given by the definitions below.

Problem 2.3 (SPE). Subgraph Pattern Enumeration

Input: *A graph dataset $\mathcal{G}$ and a boolean utility function $f : \mathcal{G}_{\sqsubseteq} \to \mathbb{B}$.*

Task: *Enumerate all connected $P \in \mathcal{G}_{\sqsubseteq}$ with $f(P) = \text{true}$.*

Problem 2.4 (SPO). Subgraph Pattern Optimization

Input: *A graph dataset $\mathcal{G}$ and a utility function $f : \mathcal{P}(\mathcal{G}_{\sqsubseteq}) \to \mathbb{R}$.*

Task: *Find $S \in \mathcal{P}(\mathcal{G}_{\sqsubseteq})$ of connected graph patterns with maximum/minimum utility value $f(S)$.*

The maximization and minimization variants of SPO are denoted by MAX-SPO and MIN-SPO.

2.6.2 Frequent Subgraph Pattern Mining

The frequent subgraph pattern mining problem is a broadly studied graph pattern mining problem in the literature. This comes as no surprise since the related frequent

itemset mining problem is a fundamental data mining problem and applied in a variety of application domains, particularly to find association rules. The Apriori algorithm [AS94] for the frequent itemset mining problem is one of the most cited articles in computer science [21]. Note that this thesis will only cover the transactional frequent subgraph pattern mining problem, i.e., graph datasets of multiple graphs as input. The mining of frequent subgraphs from a single graph is not discussed.

[AS94] Agrawal and Srikant, "Fast Algorithms for Mining Association Rules in Large Databases". 1994

[21] *Most Cited Computer Science Articles*. 2021

To define the frequent subgraph pattern mining problem the terms graph pattern support and frequent graph pattern are defined first:

Definition 2.14 (Graph Pattern Support). *Given a graph dataset $\mathcal{G}$ and a graph pattern P, let $\mathcal{G}_{\sqsupseteq P} := \{\!\{G \in \mathcal{G} \mid P \sqsubseteq G\}\!\}$ be the multiset off all graphs in $\mathcal{G}$ that P is subgraph isomorphic to. Then, the support of P in $\mathcal{G}$ is defined as* $\mathrm{supp}_{\mathcal{G}}(P) := |\mathcal{G}_{\sqsupseteq P}|$.

While the support is given as an absolute value in this definition, it is sometimes beneficial to specify it as a fraction of the dataset cardinality, i.e., $\frac{|\mathcal{G}_{\sqsupseteq P}|}{|\mathcal{G}|}$. Thus, whenever a support value of a pattern P is given as a fraction or percentage, the appropriate definition will be used interchangeably. $\mathcal{G}_{\sqsupseteq P}$ is also called supporting graph multiset of $\mathcal{G}$. For a graph $H \in \mathcal{G}_{\sqsupseteq P}$ it is said, that H supports P and P is supported by H. Definition 2.14 is aligned to the graph mining literature [e.g., IWM00; YH02].

[IWM00] Inokuchi, Washio, and Motoda, "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data". 2000

[YH02] Yan and Han, "gSpan: Graph-Based Substructure Pattern Mining". 2002

Definition 2.15 (Frequent Graph Pattern). *Given a graph dataset $\mathcal{G}$ and a minimum support threshold* $\mathrm{supp}_{\min} > 0$*, a graph pattern P is said to be frequent in $\mathcal{G}$ if and only if* $\mathrm{supp}_{\mathcal{G}}(P) \geq \mathrm{supp}_{\min}$.

The set of frequent graph patterns for a fixed minimum support $\mathrm{supp}_{\min}$ and dataset $\mathcal{G}$ will be denoted by $\mathcal{G}_{\mathrm{freq}}^{\geq \mathrm{supp}_{\min}}$ (superscript may be omitted if unambiguously given from the context). Analogously to the graph pattern support, $\mathrm{supp}_{\min}$ can be alternatively specified as a fraction or percentage. Since $\mathrm{supp}_{\min}$ is defined to be larger than zero, it follows directly, that a frequent graph pattern P must be in $\mathcal{G}_{\sqsubseteq}$. Thus, the following definition of the frequent subgraph pattern mining problem is intrinsically limited to the subgraph pattern space.

Problem 2.5 (FSM). Frequent Subgraph Pattern Mining

Input: *A graph dataset $\mathcal{G}$ and a minimum support threshold* $\mathrm{supp}_{\min}$.

Task: *Solve SPE with*

$$f_{\mathcal{G}}(P) := \begin{cases} \text{true} & P \text{ is frequent,} \\ \text{false} & \text{otherwise.} \end{cases}$$

For the efficient enumeration of frequent graph patterns one of the most central insight is the following property of graph patterns:

Lemma 2.2 (Support Anti-Monotonicity Property). *Given a graph dataset $\mathcal{G}$ the following property holds:*

$$\forall P, Q \in \mathcal{S}/\simeq : P \sqsubseteq Q \Rightarrow \mathcal{G}_{\sqsupseteq P} \supseteq \mathcal{G}_{\sqsupseteq Q} \Rightarrow \mathrm{supp}_{\mathcal{G}}(P) \geq \mathrm{supp}_{\mathcal{G}}(Q)$$

Proof. The property follows directly from the transitivity of the subgraph isomorphism relationship and the definition of $\mathcal{G}_{\sqsupseteq Q}$ (cf., definition 2.14):

$$\begin{aligned}
&\forall P, Q \in \mathcal{S}/\simeq, \forall G \in \mathcal{G}_{\sqsupseteq Q} : P \sqsubseteq Q \Rightarrow P \sqsubseteq Q \wedge Q \sqsubseteq G \\
\Rightarrow\ &\forall P, Q \in \mathcal{S}/\simeq, \forall G \in \mathcal{G}_{\sqsupseteq Q} : P \sqsubseteq Q \Rightarrow P \sqsubseteq Q \sqsubseteq G \\
\Rightarrow\ &\forall P, Q \in \mathcal{S}/\simeq, \forall G \in \mathcal{G}_{\sqsupseteq Q} : P \sqsubseteq Q \Rightarrow G \in \mathcal{G}_{\sqsupseteq P} \\
\Rightarrow\ &\forall P, Q \in \mathcal{S}/\simeq : P \sqsubseteq Q \Rightarrow \mathcal{G}_{\sqsupseteq Q} \subseteq \mathcal{G}_{\sqsupseteq P} \\
\Rightarrow\ &\forall P, Q \in \mathcal{S}/\simeq : P \sqsubseteq Q \Rightarrow \mathrm{supp}_{\mathcal{G}}(Q) \leq \mathrm{supp}_{\mathcal{G}}(P)
\end{aligned}$$

□

In other words, the partial ordering of patterns induced by their subgraph isomorphism relationship is in reverse relation to the subset relationship of their supporting graphs. Lemma 2.2 further implies: (a) Given an infrequent graph pattern, all superpatterns are infrequent, too (*upward closure property*); (b) Given a frequent graph pattern, all subpatterns are frequent, too (*downward closure property*). Considering the partially ordered pattern space as depicted in fig. 2.1, the upward closure property offers a support-based pruning of the search space for problem 2.5. Namely, given any infrequent pattern P_k on a level-wise path $(P_0, \ldots, P_k, \ldots, P_n)$—where P_i is a pattern with poset rank i—each P_j with $k \leq j$ must be infrequent, too. Thus, any rank-increasing exploration of the pattern space can stop at any infrequent pattern (cf., section 2.6.2.1). Figure 2.2 gives a visual example for support-pruned search space. Furthermore, the downward closure property can be used to eliminate candidate patterns (that might be frequent) if the candidate pattern has infrequent subpatterns without ever computing the support of the candidate (cf., paragraph 2.6.2.1.4).

The implications of lemma 2.2 directly lead to the concept of *maximal frequent graph patterns*.

Definition 2.16 (Maximal Frequent Graph Pattern). *Given a graph dataset $\mathcal{G}$ and a minimum support threshold $\mathrm{supp}_{\min} > 0$, a frequent graph pattern P is said to be maximal if and only if it has no frequent superpattern.*

The set of maximal frequent graph patterns for a fixed minimum support $\mathrm{supp}_{\min}$ and dataset $\mathcal{G}$ will be denoted by $\mathcal{G}^{\geq \mathrm{supp}_{\min}}_{\mathrm{max\,freq}}$ (superscript may be omitted if unambiguously given from the context). The concept of maximal patterns is derived from the frequent itemset mining problem, where maximal frequent itemsets are defined as inclusion maximal sets of items [AS94]. The restriction to maximal frequent graph patterns serves multiple purposes. Some superproblems are interested in special properties of them, e.g., the StruClus algorithm (cf., section 3.2) prefers large patterns over small

[**AS94**] Agrawal and Srikant, "Fast Algorithms for Mining Association Rules in Large Databases". 1994

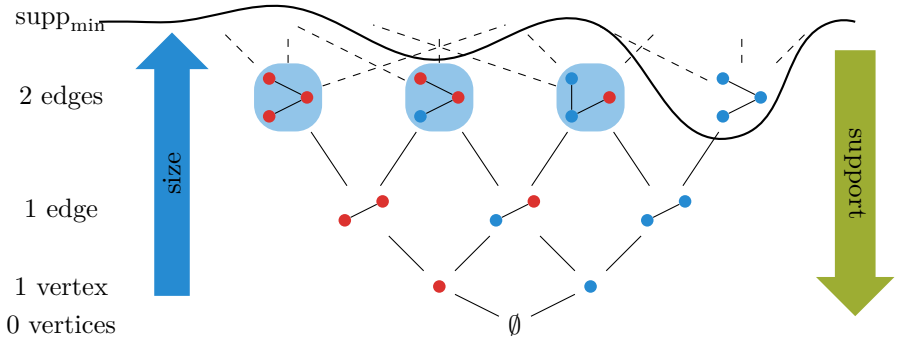


Figure 2.2: Example of a support-pruned labeled connected pattern space with $|\mathcal{L}_V| = 2$ partially ordered by the subgraph isomorphism relation $\sqsubseteq$ depicted as Hasse diagram. All graph patterns below the $\text{supp}_{\min}$ border are frequent; all above are infrequent. Frequent graph patterns with blue background are maximal. Vertex colors indicate vertex labels.

ones. Furthermore, maximal frequent graph patterns can be seen as a compressed representation of all frequent graph patterns since the frequent graph patterns of a dataset are the subpatterns of the maximal frequent graph patterns. Figure 2.2 highlights the maximal frequent graph patterns in an example of a support-pruned pattern space.

Similar to maximal frequent graph patterns are closed frequent graph patterns. Instead of being maximal w.r.t. a fixed support threshold, they are maximal regarding their own support value.

Definition 2.17 (Closed Frequent Graph Pattern). *Given a graph dataset $\mathcal{G}$ and a minimum support threshold $\text{supp}_{\min} > 0$, a frequent graph pattern P is said to be closed if and only if it has no frequent superpattern P' such that $P \sqsubset P'$ and $\text{supp}_{\mathcal{G}}(P) = \text{supp}_{\mathcal{G}}(P')$.*

2.6.2.1 Frequent Subgraph Pattern Space Exploration

The above discussion gives rise to an immediate concept for frequent pattern enumeration:

1. Start with an empty graph pattern.
2. Successively construct candidate patterns by extending or combining patterns of smaller size.
3. Stop at infrequent graph patterns.

To the best of my knowledge, all recent frequent subgraph pattern mining algorithms follow this common bottom-up exploration strategy in one or another way. Refer to section 2.6.2.2 for a discussion of implementations.

2.6.2.1.1 Apriori and Pattern Growth Candidate Generation There are two major strategies w.r.t. the candidate generation (i.e., item 2 in the above concept).

The first strategy is based on the idea of the apriori itemset mining algorithm. As such, frequent graph pattern mining algorithms based on this strategy are often called *apriori-based* or *pattern joining* algorithms. In this strategy, unordered pairs $\{\!\{P, P'\}\!\}$ of graph patterns of size k, sharing a common subpattern C of size $k-1$, are joined to candidate graph patterns S of size $k+1$, such that S is a superpattern of P and P'. This includes self-joins, i.e., $P = P'$. The size of a pattern in this context is either measured in terms of vertices (vertex-based candidate generation) or in terms of edges (edge-based candidate generation). It is important to distinguish these two variants of candidate generation as they may lead to different $k+1$-sized candidate patterns based on the same pair of k-sized patterns. Some edge-based algorithms ignore frequent patterns with a single vertex. In the following, the edge-based variant will be described exemplarily. In contrast to the above join definition over graph patterns, implementations operate on fixed pattern representations $R \in P$ and $R' \in P'$. Thus, C corresponds to (possibly multiple) common subgraph isomorphisms between R and R'. Multiple subgraph isomorphisms for a single pair of pattern representatives may exist for the following reasons ([see KK01]). First, there might exist multiple automorphisms of a common subgraph, which leads to multiple common subgraph isomorphisms covering the same set of vertices in R and R'. Second, multiple common subgraphs of size $k-1$ may exist and even a fixed common subgraph may be associated with different isomorphisms to subgraphs of R and R' that cover different vertex sets. A join between R and R' is then performed for each existing common subgraph isomorphism of size $k-1$. In a first step, the vertex and edge sets of the two graphs are joined. Afterwards, the vertices covered by the common subgraph isomorphism are contracted. Note that in the case of labeled datasets, the contracted vertices and the associated merged edges always have the same label, i.e., the contraction does not lead to label conflicts. If the two uncovered vertices have no different labels, two different candidates are created, one with contracted uncovered vertices and one with separate uncovered vertices. Thus, in contrast to the apriori itemset joining, the existence of multiple subgraph isomorphism of size $k-1$ and the choice of contracting the uncovered vertices may lead to multiple candidates when joining two patterns. Figure 2.3 shows an example of such an apriori-based pattern joining with edge-based candidate generation. A major challenge for the apriori-based graph pattern mining algorithms is the identification of the common subgraph isomorphisms between all frequent k-sized patterns in order to generate the candidate patterns of size $k+1$.

[KK01] Kuramochi and Karypis, "Frequent subgraph discovery". 2001

The second candidate generation strategy is called *pattern growth*. This strategy avoids the computationally demanding graph pattern joining from the apriori-based candidate generation. Instead, candidates are constructed by extending frequent patterns by frequent edge patterns. A sequence of such (single edge pattern) extensions

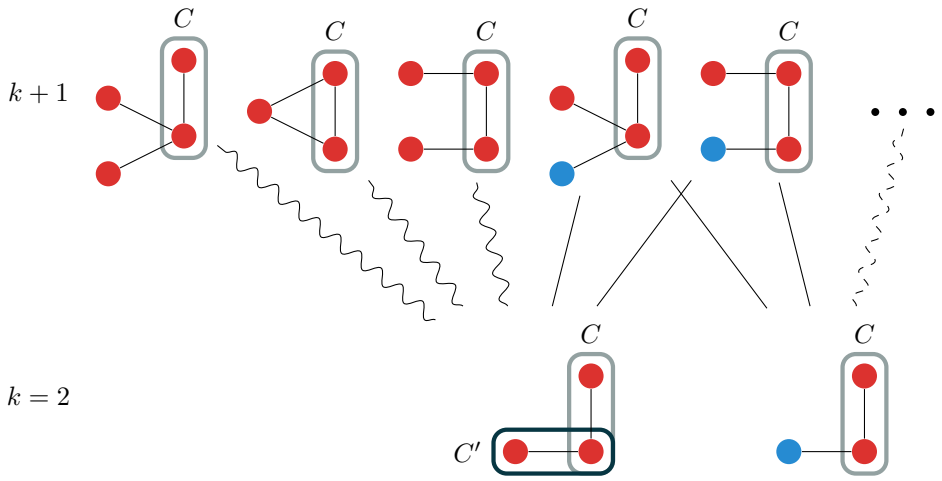


Figure 2.3: Example of apriori-based graph pattern joining with edge-based candidate generation. Patterns of size $k+1$ are created by joining two patterns of size k. C and C' are the vertices in the pattern representatives covered by the subgraph isomorphisms of the join. Straight lines indicate joins of two distinct patterns, wave lines indicate self joins, and dashed lines indicate further joining possibilities which are not shown in this example. Vertex colors indicate vertex labels.

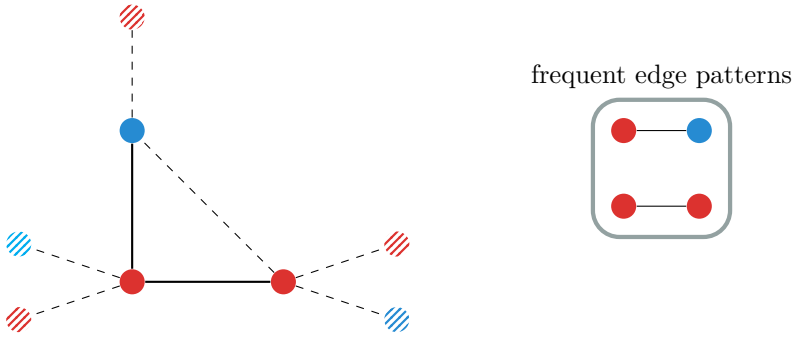


Figure 2.4: Possible connected frequent edge pattern extensions for a given graph pattern in the pattern growth approach. Dashed edges and vertices illustrate all possible extensions, consisting of one backward extension and six forward extensions. The set of frequent edge pattern is depicted on the right. Vertex colors indicate vertex labels.

can be seen as a depth-first search (DFS) ordered construction (cf., depth-first search in [Cor+09]) of a graph pattern representative. As such, in the pattern mining literature, a single extension is classified as forward extension (corresponding to a DFS tree edge; not to confuse with DFS forward edges!) or backward extension (corresponding to a DFS backward edge). Thus, a backward extension connects two existing vertices and a forward extension expands a graph by connecting an existing vertex of the pattern representation with a new vertex. For labeled graphs, the connecting vertices must match the labels of the frequent edge pattern.

[Cor+09] Cormen et al., *Introduction to Algorithms, 3rd Edition*. 2009

Definition 2.18 (Labeled Forward Extension). *Let $G = (V, E, \Gamma)$ be a labeled graph, $v \in V$ an explicit vertex of G, and $H = (\{u, w\}, \{e = \{u, w\}\}, \Gamma')$ an arbitrary representative of an edge pattern P. Furthermore, let $\Gamma(v) \cap \{\Gamma'(u), \Gamma'(w)\} \neq \emptyset$. Without loss of generality, let $\Gamma(v) = \Gamma'(u)$. Then, the labeled graph $\mathrm{fw}(G, v, P) = G' = (V \uplus \{x\}, E \cup \{\{v, x\}\}, \Gamma \cup (\{v, x\}, \Gamma'(e)) \cup (x, \Gamma'(w)))$ is called forward extension of G with P at extension vertex v.*

Definition 2.19 (Labeled Backward Extension). *Let $G = (V, E, \Gamma)$ be a labeled graph, $t, v \in V$ two explicit vertices of G with $\{t, v\} \notin E$, and $H = (\{u, w\}, \{e = \{u, w\}\}, \Gamma')$ an arbitrary representative of an edge pattern P. Furthermore, let $|\{\Gamma(t), \Gamma(v)\} \cap \{\Gamma'(u), \Gamma'(w)\}| = 2$. Without loss of generality, let $\Gamma(v) = \Gamma'(u)$. Then, the labeled graph $\mathrm{bw}(G, t, v, P) = G' = (V, E \cup \{\{t, v\}\}, \Gamma \cup (\{t, v\}, \Gamma'(e)))$ is called backward extension of G with P at extension vertices t and v.*

See fig. 2.4 for an example of forward and backward extensions.

2.6.2.1.2 Depth- and Breadth-First Search The ranked poset of frequent patterns as depicted in fig. 2.2 can be explored in a breadth-first (BFS) or depth-first (DFS) fashion (see Cormen et al. [Cor+09] for DFS and BFS definitions). Apriori-based pattern mining algorithms are intrinsically bound to the breadth-first exploration since the k-sized candidate pattern generation needs to consider all $k-1$ sized pattern pairs (cf., paragraph 2.6.2.1.1). Pattern growth algorithms in general are not limited to a specific exploration order. However, as described in section 2.6.1.2, each poset rank may contain an exponential number of patterns w.r.t. the rank level. As a result, most sequential pattern growth algorithms use the depth-first exploration strategy since it minimizes the number of patterns that need to be kept in memory. Nevertheless, breadth-first or hybrid exploration orders are often utilized to parallelize or distribute the work even for the pattern growth candidate generation as a consequence of the sequential exploration dependency along the DFS tree edges.

[**Cor+09**] Cormen et al. *Introduction to Algorithms, 3rd Edition*. 2009

2.6.2.1.3 Duplicate Elimination All candidate generation strategies can generate isomorphic candidate graphs. For example, in the pattern growth scenario, the Hasse diagram depicted in fig. 2.2 shows multiple rank-increasing level-wise construction paths for a single pattern. For the apriori-based mining algorithm one source of isomorphic candidates (among others), are multiple embeddings of a common subgraph of size $k-1$ such as C and C' in fig. 2.3. To avoid multiple enumerations of the same pattern, isomorphic candidate pattern representatives need to be eliminated during the mining process. In other words, graph pattern mining algorithms usually operate explicitly, i.e., with an well-defined representative function ρ, or implicitly, i.e., by eliminating duplicate patterns on each rank level, on a representative pattern space $[\mathcal{S}]_{\simeq}$. In order to avoid computationally demanding pairwise isomorphism tests on each rank level, canonical forms or graph invariants are commonly used. The duplicate elimination strategies are quite diverse and often strongly linked to the candidate generation and/or graph representation of the specific algorithm. For this reason, the de-duplication strategies are discussed alongside the specific algorithms in section 2.6.2.2.

2.6.2.1.4 Support Calculation To determine if a candidate pattern is frequent the support of the pattern needs to be calculated. The support calculation is typically the major bottleneck of frequent subgraphs mining algorithms even for small datasets [Wör+05]. There exist three major strategies for support calculation: *linear dataset scans*, *transactions lists*, and *embedding lists*.

[**Wör+05**] Wörlein et al., "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston". 2005

Naively, support calculation can be performed as a linear scan over the graph dataset to count the positive subgraph isomorphism relationships between the candidate and the dataset graphs. Keeping in mind, that the subgraph isomorphism problem is NP-hard and the number of frequent subgraphs may be exponential w.r.t. to the graph size (cf., section 2.6.1.2), the above bottleneck statement comes as no surprise. However, there are some common observations, to speed up the practical running time when it comes to linear scans. First, the support does not need to be calculated exactly. The linear scan can be aborted whenever the minimum support is reached

or whenever it becomes unreachable, i.e., the remaining unchecked dataset graphs are not enough to reach the minimum support. Second, one can limit the linear scan to transaction lists of subpatterns. A transaction list is associated with a graph pattern P and stores a reference to all dataset graphs that support P, i.e., $\mathcal{G}_{\sqsupseteq P}$. This is motivated by the anti-monotonicity property (cf., lemma 2.2), which limits the supporting dataset graphs of a pattern candidate $P \in \mathcal{G}_{\sqsubseteq}$ to the supporting dataset graphs of each subpattern $P' \sqsubset P$, i.e., the following relation holds:

$$\mathcal{G}_{\sqsupseteq P} \subseteq \bigcap_{P' \sqsubset P \in \mathcal{G}_{\sqsubseteq}} \mathcal{G}_{\sqsupseteq P'} = \bigcap_{\substack{P' \sqsubset P \in \mathcal{G}_{\sqsubseteq}, \\ |P'|=|P|-1}} \mathcal{G}_{\sqsupseteq P'} =: C_P \tag{2.12}$$

In other words, $\mathcal{G}_{\sqsupseteq P'}$ is an upward closed set w.r.t. to the poset $(\mathcal{G}_{\sqsubseteq}, \sqsubseteq)$. Since the anti-monotonicity property is transitive it is sufficient to intersect only direct subpatterns, i.e., $|P'| = |P| - 1$. This is closely related to the upward closure property (cf., definition 2.14) with the addition, that C_P can serve as a candidate list for $\mathcal{G}_{\sqsupseteq P}$. Thus, we can omit all subgraph isomorphism tests to graphs in $\mathcal{G} \setminus C_P$, since

$$\forall G \in \mathcal{G} \setminus C_P : P \not\sqsubseteq G$$

holds. However, the determination of all direct subpatterns and the associated transaction lists causes additional overhead. First, retrieving the transaction lists of all direct subpatterns of a pattern P requires to remember all transaction lists of frequent patterns of size $|P| - 1$ and a BFS exploration strategy. Second, to retrieve the transaction lists of all subpatterns, it is required to enumerate all subpatterns and match the subpatterns to already mined patterns via isomorphism tests between their representatives. Third, for large transaction lists (or graph datasets respectively) the intersections themselves can be computationally demanding. For these reasons, many DFS FSM algorithms avoid the intersection of multiple transaction lists and only consider the transaction list of the parent pattern (w.r.t. to the DFS tree) to calculate the pattern support.

Contrariwise to transaction lists, which store only the binary subgraph relationships between a pattern and each dataset graph, embedding lists store all embeddings (subgraph isomorphisms) of a pattern representative to each dataset graph. To calculate the support of a pattern, it is thereby sufficient to count all the dataset graphs for which at least one embedding exists. Embedding lists have the advantage, that an extension of a pattern can be seen as an extension of existing embeddings. Thus, given a pattern representative R, an extension vertex v (or two extension vertices u, v in the case of a backward extension; cf., definitions 2.18 and 2.19) and an embedding $\psi : R \to G$ to a dataset graph $G \in \mathcal{G}$, one only needs to consider adjacent vertices of $\psi(v)$ (or edges between $\psi(u)$ and $\psi(v)$) in order to find the embeddings of the extended pattern representative. In other words, it is possible to avoid the re-calculation of partial embeddings for a pattern $P \in \mathcal{G}_{\sqsubseteq}$ that were already calculated for some subpattern $P' \sqsubset P$. On the downside, all embeddings of R to G need to be enumerated and kept in memory, which is an exponential number w.r.t.

the size of $V(G)$ and factorial in $V(R)$ for unrestricted graph classes in the worst case [Wel20]. Contrariwise, transaction lists only need to store a binary relation and the associated subgraph isomorphism tests does not need to enumerate all embeddings (i.e., it can terminate after the first embedding is discovered). In the context of pattern growth algorithms, embeddings lists additionally allow the elimination of some pattern extensions. Given an extension vertex v of a pattern representative R, it is possible to enumerate all incident edges E_{incident} to $\psi(v)$ for all embeddings ψ in the embedding list of R. Only patterns found in E_{incident} need to be considered as possible extensions of R at extension vertex v, i.e., the set of frequent edge patterns can be pruned accordingly.

[**Wel20**] Welke, "Efficient Frequent Subgraph Mining in Transactional Databases". 2020

2.6.2.2 Frequent Subgraph Mining Algorithms

A plethora of different frequent subgraph pattern mining algorithms have been proposed in the last two and a half decades. For this reason, this section will only discuss a selection of FSM algorithms covering the most important algorithmic techniques to solve the FSM mining problem as defined in problem 2.5 (FSM). In particular, this section will not cover early ILP (Inductive Logical Programming) approaches to graph pattern mining, such as presented in [DTK98], which were limited to very small pattern sizes. Frequent graph pattern mining algorithms for restricted pattern classes—such as tree pattern mining algorithms—are not discussed either. Since the algorithm of chapter 4 requires a full enumeration of all frequent patterns and chapter 3 only uses a lightweight sampling method for maximal patterns, this section will also not cover subgraph mining algorithms specifically designed to mine closed or maximal frequent subgraph patterns.

[**DTK98**] Dehaspe, Toivonen, and King, "Finding Frequent Substructures in Chemical Compounds". 1998

Inokuchi, Washio, and Motoda [IWM00] presented the first apriori-based algorithm *Apriori-based Graph Mining* (AGM). It mines integer labeled disconnected induced subgraph patterns with a breadth-first search pattern space exploration and node counting. The algorithm uses normalized adjacency matrices—which are ordered by the vertex labels—for pattern and dataset graph representation. In contrast to many other algorithms, it does not eliminate all isomorphic patterns on each level. Instead, it only normalizes the matrix representation of the pattern and keeps an index subject to isomorphic instances with the help of a canonical matrix code. By doing so, the identification of the common subgraph for the join can be performed by matching the upper left part of the adjacency matrix. However, keeping multiple instances of the same pattern comes at the cost of memory consumption and increases the number of pairwise pattern representations that need to be considered for joining. For support counting of a pattern P, AGM uses linear dataset scans. The upward closure property is used to filter infrequent patterns prior to the linear dataset scan by checking if each subgraph of size $|P|-1$ is frequent, i.e., discovered as frequent pattern of rank $|P|-1$.

[**IWM00**] Inokuchi, Washio, and Motoda "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data". 2000

In 2001 Kuramochi and Karypis [KK01] presented another apriori-based algorithm, named *Frequent Subgraph Discovery* (FSG). It outputs connected non-induced patterns, uses a breadth-first search pattern space exploration, and edge counting. The authors claim a speedup of several orders of magnitudes on chemical datasets in comparison with AGM. However, it should be noted that the tasks of the two algorithms

[**KK01**] Kuramochi and Karypis "Frequent subgraph discovery". 2001

are not identical (induced vs. non-induced patterns). In contrast to AGM, FSG uses a sparse adjacency list representation for the graphs and pattern representatives. Duplicate patterns are eliminated on each rank level using a canonical adjacency list representation based on a node ordering that is obtained via a canonical adjacency matrix code. To identify common subgraphs to join patterns on rank k, a canonical representation for each $k-1$ subpattern of each discovered frequent pattern is computed and kept in memory. For support counting of a pattern P, FSG uses the candidate list C_P as defined in eq. (2.12).

In 2002 Yan and Han [YH02] presented the *Graph-Based Substructure Pattern Mining* (gSpan) algorithm and Borgelt and Berthold [BB02] presented the MoFa algorithm (sometimes called MoSS). A detailed description of the gSpan algorithm is given in paragraph 2.6.2.2.1, since it is used as a base algorithm in chapter 4. Subsequent extensions of the algorithms exist, covering closed graph pattern mining (MoFa: [MBB04a], gSpan: [YH03]) and subgraph pattern mining with fuzzy chains (MoFa: [MBB04b]). Jahn and Kramer [JK05] optimized the running time of the gSpan algorithm in the context of molecular datasets. The two algorithms share several design aspects. Both are pattern growth algorithms for connected non-induced patterns with a depth-first search pattern space exploration and edge counting for labeled connected patterns. While gSpan uses transaction lists for support counting, MoFa resorts to embedding lists. MoFa uses these embedding lists to grow only possible extensions from the embeddings in the dataset graphs as described in paragraph 2.6.2.1.4. Additionally, they restrict the pattern growth step with a heuristic to avoid duplicate enumerations of the same pattern. MoFa orders the vertices of the dataset graphs in an arbitrary but fixed order. Given this ordering, the extensions of embeddings are restricted, such that the extension vertex (cf., definition 2.18) has the lower order in comparison with the adjacent vertex of the extensions edge pattern. Because of the heuristic nature of this duplicate filtering step, MoFa still relies on duplicate filtering on a global level, i.e., each mined pattern has to be compared with already mined patterns. Subsequent extensions of these duplicate patterns cannot be pruned, resulting in partially overlapping search trees in the pattern space. On the contrary, gSpan uses a minimum DFS code pattern canonization, which completely filters duplicate patterns by a local test after each pattern extension. The prefix property of this canonization allows the pruning of the DFS search tree at non-canonical pattern representations.

[YH02] Yan and Han "gSpan: Graph-Based Substructure Pattern Mining". 2002

[BB02] Borgelt and Berthold "Mining Molecular Fragments: Finding Relevant Substructures of Molecules". 2002

[MBB04a] Meinl, Borgelt, and Berthold, "Discriminative Closed Fragment Mining and Perfect Extensions in MoFa". 2004

[YH03] Yan and Han, "CloseGraph: Mining Closed Frequent Graph Patterns". 2003

[MBB04b] Meinl, Borgelt, and Berthold, "Mining Fragments with Fuzzy Chains in Molecular Databases". 2004

[JK05] Jahn and Kramer "Optimizing gSpan for molecular datasets". 2005

Huan, Wang, and Prins [HWP03] presented the *Fast Frequent Subgraph Mining* (FFSM) algorithm in 2003. It uses a hybrid join and extension strategy to explore the search space in breadth-first order, embeddings lists, and a canonical matrix representation of the graph patterns. Similar to the gSpan algorithm, FFSM completely filters duplicate patterns based on a local test on canonical representation. A special property of their canonical matrix representation is, that the submatrix of a connected subpattern is the canonical matrix of the subgraph. As a consequence, the submatrices of a joined or an extended pattern candidate must be frequent canonical representations of already mined patterns. Thus, testing on matrix identity offers the classical downward closure property filtering (as given in eq. (2.12)). At the same time, this procedure filters some of the non-canonical pattern representations. The

[HWP03] Huan, Wang, and Prins "Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism". 2003

remaining candidates then need to be tested, if they are in canonical form. FFSM uses a hybrid pattern-join and pattern growth exploration since their join over canonical submatrices is incomplete and would miss some pattern, otherwise. It should be noted, that—in contrast to the previous algorithms[10]—*FFSM* is only suited for undirected patterns and dataset graphs since their canonical matrix representation strongly depends on its symmetry.

[NK04] Nijssen and Kok "Frequent graph mining and its application to molecular databases". 2004

In 2004 Nijssen and Kok [NK04] presented the Gaston algorithm for undirected graphs. It uses a pattern extension strategy to explore the search space in depth-first order with edge counting and embeddings lists. The major novelty of the Gaston algorithm is, that it splits the discovery of frequent subgraph patterns into three phases. Each phase represents a special class of graph patterns: paths, free trees, and cyclic patterns. After mining all frequent path patterns, it grows the tree patterns from them and subsequently cyclic patterns. For all pattern classes, Gaston uses a distinct canonical form and grow strategy. A special focus was put on the canonization of the trees. The longest path in a tree is called backbone and their tree canonization is based on the identification of such a backbone. Gaston uses the already mined frequent paths from phase one as the backbone of the trees grown in phase two, i.e., extensions of tree patterns which would lead to a different backbone are invalid. With this restriction, Gaston is able to completely avoid the generation of duplicate candidates for tree patterns. The lower complexity for the first two classes w.r.t. to the subgraph isomorphism problem and canonization enable Gaston to avoid costly operations for a portion of the frequent patterns. Nijssen and Kok especially targeted molecular datasets, which have special properties that render the above strategy effective.

[Hua+04] Huan et al. "SPIN: Mining Maximal Frequent Subgraphs from Graph Databases". 2004

In fact, Nijssen and Kok [NK04] were not the only ones that had the idea of splitting the mining into multiple phases based on different pattern classes. In the same year, Huan et al. [Hua+04] presented an algorithm, called *Spanning Tree Based Maximal Graph Mining* (SPIN), which uses a two-phase approach (trees and cyclic patterns). The algorithm is specialized to mine maximal patterns only and thus outside the scope of this thesis.

[BH14] Bhuiyan and Hasan "FSM-H: Frequent Subgraph Mining Algorithm in Hadoop". 2014

[BH15] Bhuiyan and Hasan, "An Iterative MapReduce Based Frequent Subgraph Mining Algorithm". 2015

In addition to the above listed sequential algorithms, multiple distributed algorithms have been proposed as well. Bhuiyan and Hasan [BH14] proposed a distributed MapReduce algorithm *Frequent Subgraph Mining Algorithm in Hadoop* (FSM-H) in 2014. The same algorithm with some additional detail was republished in 2015 ([BH15]). It uses the minimal DFS code of gSpan for canonization, a BFS exploration order, and embedding lists for support calculation. Each poset rank is processed by a separate MapReduce instance and the dataset is partitioned among the mappers. In iteration k each mapper processes all k-sized frequent subgraph patterns which have an absolute support of at least one in the partition element. It calculates all extensions for these patterns and calculates the embeddings lists w.r.t. the assigned dataset graphs. The mapper then emits the local support values with the minimal DFS code as key. The reducers sum up to local support values.

[10] Some of these algorithms are presented for undirected graphs only. However, there exists trivial extensions to the original algorithms to support directed graphs

In the same year Lin, Xiao, and Ghinita [LXG14] presented another MapReduce algorithm that works in three rounds (filter, sort, refine). It uses a canonical adjacency matrix duplicate elimination, and a local mining approach similar to SPIN. The basic idea is, that frequent pattern mining is possible in a single MapReduce round by partitioning the dataset, enumerating the complete subgraph pattern space together with the support values of each subgraph pattern on each partition element in the map phase, and summing up the partial support values in the reduce phase. However, this simple approach has the bottleneck, that the local enumeration cannot apply any support-based pruning and the number of subgraph patterns sent to the reducers is extremely high. For this reason, they calculate the probability for a pattern to be globally infrequent based on the local support. Whenever the probability is low enough the algorithm does not enumerate the pattern and does not send the pattern to the reducers in the filter round. Consequentially, the support values of some patterns are only partially available in the reduce phase. Nevertheless, the chosen approach guarantees, that a frequent pattern is output by at least a single mapper. With this partial information, it is possible to categorize the enumerated patterns in the categories infrequent, probably frequent, and frequent in the sorting round. The exact support values of the probably frequent patterns are then computed in the refinement round by computing the local support on each dataset partition element.

[LXG14] Lin, Xiao, and Ghinita "Large-scale frequent subgraph mining in MapReduce". 2014

Talukder and Zaki [TZ16] presented the frequent subgraph pattern mining algorithm *Parallel Graph Mining with Dynamic Load Balancing* (ParGraph) implemented in MPI and OpenMP. While the algorithm is tailored towards a single input graph, it is easily adjustable to the transaction setting. In contrast to the above presented distributed algorithms, they do not distribute the dataset graphs, but the pattern space exploration among the workers. Consequentially, their algorithm aims to speed up the computation for datasets that still fit in the memory of each worker. While using a DFS-based exploration, they assign subspaces of the minimal DFS code treeified pattern space rooted at some pattern to the workers. Since the subspaces have a skewed distribution w.r.t. the amount of work, they implemented a distributed work-stealing algorithm. Whenever a worker runs out of work, it steals some of the unprocessed patterns of another worker. The unprocessed patterns naturally occur, when a pattern can be extended in multiple ways and only some of these extensions are already processed by the DFS traversal.

[TZ16] Talukder and Zaki "Parallel graph mining with dynamic load balancing". 2016

Petermann, Junghanns, and Rahm [PJR17] presented the DIMSpan algorithm in 2017. Similar to FSM-H, it uses a BFS exploration order and uses the minimum DFS code canonization from gSpan in combination with embedding lists for support counting. Also in alignment with FSM-H they keep a full copy of frequent patterns (which have an absolute support of at least one in the partition element) in each workers memory. However, instead of using the MapReduce distribution framework, it relies on resilient distributed datasets in Spark. This allows the dataset and the patterns to be kept in memory of the workers without the need to write intermediate results to disk. Additionally, DIMSpan optimizes some details, such as using the restricted pattern growth from gSpan in combination with the growth restriction of MoFa to grow only patterns, that are possible w.r.t. the embedding lists. Another

[PJR17] Petermann, Junghanns, and Rahm "DIMSpan: Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems". 2017

optimization is the usage of compressed pattern representations to reduce the memory requirements for each worker and the bandwidth usage in order to communicate them.

[Dia+19] Dias et al. "Fractal: A General-Purpose Graph Pattern Mining System". 2019

Dias et al. [Dia+19] presented a general-purpose graph pattern mining framework called Fractal in 2019. Although the framework is tailored towards a single input graph, it has several interesting conceptual novelties and is therefore listed here. Foremost, it generalizes the pattern space exploration process by allowing the user to specify custom extension, aggregation and filtering primitives. Thus, the framework can be used not only to implement a frequent subgraph mining algorithm, but many other subgraph pattern mining problems, such as motif and clique extraction and counting, subgraph querying, and keyword-based subgraph search. The main goal of the framework is to let the user only specify the logical task at hand and to abstract the technical details about the pattern space exploration, support counting, and load balancing. Internally, the framework uses a gSpan-like pattern space exploration with minimum DFS code canonization. The work is distributed in Spark and implements a work-stealing approach similar to ParGraph. Thus, the framework also shares the limitation of ParGraph to load the complete dataset into each worker's memory.

[YH02] Yan and Han, "gSpan: Graph-Based Substructure Pattern Mining". 2002

[Wör+05] Wörlein et al., "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston". 2005

[Bor07] Borgelt, "Canonical Forms for Frequent Graph Mining". 2007

2.6.2.2.1 gSpan Algorithm In the following, the gSpan algorithm [YH02] will be discussed in more detail, since the implementations of algorithms given in chapter 4 are based on gSpan. The gSpan algorithm is a pattern growth algorithm for undirected labeled graphs with edge counting. However, the adoption to directed graphs is possible with minor changes [Wör+05]. gSpan uses a so-called DFS code as graph representation. As mentioned in paragraph 2.6.2.1.1, a graph can be constructed by adding a single vertex to the empty graph followed by a sequence of forward and backward extensions. This extension sequence aligns with a DFS exploration order of the final graph. The DFS code represents such a construction and DFS exploration order. The following representation is based on [Bor07], which presented a simplified representation compared to the original paper of Yan and Han. The representation has the form:

$$l_{vs}(t_d \overleftarrow{t_s} l_e l_{vd})^m \ , \tag{2.13}$$

where $l_{v\cdot}$ are vertex labels, l_e is an edge label, $t_\cdot$ are DFS exploration timestamps, and m is the number of repeats. The subscripts s and d stand for source and destination vertex, while the source vertex is defined to be the vertex with the lesser DFS exploration timestamp. Thus, l_{vs} is the label of the vertex for which the DFS exploration is started or the first vertex added to the empty graph in the construction process, depending on the viewpoint. Subsequently, each repetition in the parenthesis is a DFS exploration step or an extension. The left arrow above the source timestamp indicates reverse ordering (see below).

An example graph with DFS-exploration vertex timestamps is given in fig. 2.5. A matching DFS code would be (with black edge labels):

● 21●● 32●● 31●● 43●● 42●● 52●● (2.14)

Undoubtedly, there exist multiple DFS codes for a single graph w.r.t. different DFS exploration or construction orders. Even for fixed DFS vertex timestamps, the

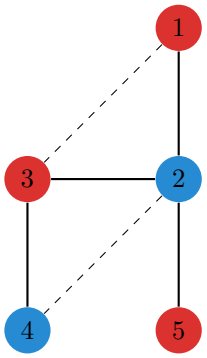


Figure 2.5: Example DFS-exploration of a labeled graph. Vertex colors indicate vertex labels. Vertex numbers indicate DFS exploration timestamps. DFS-tree edges are depicted by straight lines; backward edges are depicted by dashed lines.

backward edges might be added at an arbitrary position after the incident vertices are present. For example, the following DFS code is different to eq. (2.14), but still matches fig. 2.5:

$$\bullet 21 \bullet\bullet 32 \bullet\bullet 43 \bullet\bullet 52 \bullet\bullet 31 \bullet\bullet 42 \bullet\bullet \tag{2.15}$$

On the other hand, a single DFS code might present multiple DFS exploration orders as a consequence of automorphisms.

> **Lemma 2.3 (Canonical DFS Code).** *The lexicographically minimum DFS code is a graph invariant. [YH02]*

[YH02] Yan and Han, "gSpan: Graph-Based Substructure Pattern Mining". 2002

[Bor07] Borgelt "Canonical Forms for Frequent Graph Mining". 2007

Based on the above DFS code definition, Yan and Han [YH02] and Borgelt [Bor07] define a minimal DFS code as a canonical of the represented graph pattern. Given an ordering of the vertex and edge labels, one can sort each repetition in the code to deduce an ordering of DFS codes. While the former definition is a bit more complex in comparing each repetition, the above presented form given by [Bor07] can be simply sorted lexicographically. The source time stamp is sorted in reverse order. The two definitions are equivalent in the sense, that the same DFS exploration orders lead to minimal DFS codes.

> **Lemma 2.4 (Canonical Prefix Property).** *Given a prefix p of a minimal DFS code with m repeats (cf., eq. (2.13)), that aligns with the repetition boundaries (i.e., for some $i \leq m$), p is the canonical DFS code of the represented graph pattern.*

Given the lexicographic ordering of a minimal DFS code, the prefix property follows directly. The minimal DFS code is used in gSpan to eliminate duplicate patterns.

Using a DFS-based pattern growth approach, one can simply check whether the DFS code aligning with the construction sequence is in canonical form. Furthermore, the prefix property guarantees, that no extension of a non-canonical DFS code can be canonical. Consequentially, similar to the support-based pruning, subspaces of the search space rooted in a non-canonical exploration path can be pruned.

To test whether a DFS code is minimal, a backtracking algorithm is used that enumerates different DFS exploration orders. As a consequence of the prefix property, it is possible to prune the search whenever a non-canonical prefix is enumerated. Still, a possibly exponential number of DFS exploration orders might be considered. For example, a complete graph with uniform labels has an exponential number of automorphism, since each vertex can be mapped to every other vertex. Thus, the same DFS code is created for all the exploration orders. Only after adding the last repetition to the DFS code, one can decide, that the generated code is not less than the reference code., i.e., no canonization-based pruning can be applied.

To avoid some of these possibly computationally expensive minimal checks, gSpan restricts the extension in such a way, that some non-canonical forms of DFS codes are not generated in the first place. Especially, the extensions are restricted to generate DFS codes that align with DFS explorations orders (and not e.g., a BFS exploration order). Namely, backward extensions are restricted to the last added vertex, which is called rightmost vertex in the literature. This is justified by the fact, that forward extensions added afterwards will have a higher destination vertex timestamp, i.e., a backward extension has a smaller repetition order. For example, the code in eq. (2.15) cannot be minimal, because the backward extension is added too late. Additionally, gSpan restricts forward extensions to extension vertices on the rightmost path, which is defined as the path in the DFS tree from the DFS tree root to the rightmost vertex.

[**Bor07**] Borgelt, "Canonical Forms for Frequent Graph Mining". 2007

For the sake of completeness, it should be mentioned that [Bor07] also presented a BFS code, that defines the minimal code based on the BFS exploration order. However, even the BFS code can be used in conjunction with a DFS pattern space exploration.

2.6.2.3 Computational Complexity and Practical Performance

[**HBD07**] Horváth, Bringmann, and De Raedt, "Frequent Hypergraph Mining". 2007

[**NM17**] Neumann and Miettinen, "Reductions for Frequency-Based Data Mining Problems". 2017

[**Wel20**] Welke, "Efficient Frequent Subgraph Mining in Transactional Databases". 2020

[**GJ79**] Garey and Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979

[**Lin89**] Lingas, "Subgraph Isomorphism for Biconnected Outerplanar Graphs in Cubic Time". 1989

[**HRW10**] Horváth, Ramon, and Wrobel, "Frequent Subgraph Mining in Outerplanar Graphs". 2010

The general FSM problem is known to the NP-hard [HBD07] while it is polynomial-time solvable for certain constrained pattern spaces [NM17]. Furthermore, the FSM problem and even the frequent tree mining problem—which restricts the pattern space to trees—cannot be solved in output polynomial time unless P = NP [Wel20]. This hardness is closely related to the hardness of the support counting step. The subgraph isomorphism problem has to be solved for the database scan respectively transaction list setting. This problem is NP-complete even for quite restricted graph classes such as outerplanar graphs or disconnected tree patterns [GJ79]. Positive exceptions of this hardness are biconnected outerplanar graphs [Lin89] and connected tree patterns when embedded into forests. Restricted to block and bridge preserving subgraph isomorphisms it is possible to solve the FSM problem in incremental polynomial delay time [HRW10]. Block and bridge preserving subgraph isomorphisms are restricted to graphs which have a tree structure, where tree vertices are allowed to be replaced

by biconnected outerplanar graphs. The corresponding subgraph isomorphism is restricted to map blocks (biconnected outerplanar components) to blocks and bridges (connecting tree paths) to bridges. Embedding lists do not lower the complexity of the support counting step, since they solve the subgraph isomorphism problem for a sequence of pattern extensions. Welke [Wel20] points out, that the size of the embedding lists can be as high as the number of subgraph isomorphisms that are considered by the Ullmann's algorithm [Ull76] for the subgraph isomorphism problem. Ullmann's algorithm operates on a fixed ordering $(v_1, \ldots, v_{|V(R)|})$ for the vertices of a representative R of a pattern P which implies a sequence of induced subgraphs $(R_1 \ldots, R_{|V(R)|})$ where $V(R_i) = \bigcup_1^i v_i$. The number of subgraph isomorphism to a graph G is then bound by:

[Ull76] Ullmann, "An Algorithm for Subgraph Isomorphism". 1976

$$\sum_{i=1}^{|V(R)|} \binom{|V(G)|}{|V(R_i)|} |V(R_i)|!.$$

The sequence of induced subgraphs aligns with an (vertex) extension sequence of a rank-increasing level-wise pattern exploration with a vertex-induced subgraph isomorphism.

Each pattern mining algorithm claims it superiority over its predecessors w.r.t. its practical performance. However, survey papers, such as [Wör+05; JCZ13], indicate, that there is no such dominance of a single algorithm over a broader selection of graph datasets. While it is widely accepted, that the early apriori-based miners—i.e., AGM and FSG—and MoFa are superseded by gSpan, FFSM, and Gaston, the latter have their individual strengths and weaknesses. The major problem is, that it is impossible to predict which algorithm will perform best or even well on a specific dataset. Welke, Horváth, and Wrobel [WHW19] state, that the performance often depends on some graph properties that are not formally captured and are unknown. They especially name Gaston as an example. Embedding lists seem to have a quite unpredictable behavior w.r.t. the number of embeddings. For some datasets, the number of embeddings seems to explode with no apparent reason or predictability [WHW19]. Wörlein et al. concluded, that "Contrary to common belief embedding lists do not considerably speed up the search for frequent fragments" [Wör+05, p. 402]. Instead, some instances become intractable because of memory limitations.

[Wör+05] Wörlein et al., "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston". 2005

[JCZ13] Jiang, Coenen, and Zito, "A survey of frequent subgraph mining algorithms". 2013

[WHW19] Welke, Horváth, and Wrobel "Probabilistic and exact frequent subtree mining in graphs beyond forests". 2019

2.6.2.4 Application to Other Subgraph Pattern Mining Problems

Frequent subgraph pattern mining algorithms are the basis of many algorithms, which solve other subgraph pattern mining problems. This includes representative graph pattern mining (cf., section 4.1), discriminative graph pattern mining, significant graph pattern mining, graph pattern feature selection, and more. The variety of applications is overwhelming, making it impossible to give a full overview of FSM adaptations. Therefore, this subsection lists only a selection of examples in order to present the general idea of adaptations.

On the one hand, some algorithms use frequent subgraph pattern mining simply as a subroutine to generate some structural features, which are then processed in a separate step. For example, Hasan et al. [Has+07] used a two-step *mine-and-select* approach to mine representative subgraph patterns. Sugiyama et al. [Sug+15] mined significant subgraph patterns on binary classified datasets with the help of an arbitrary frequent subgraph pattern mining algorithm. Since they do not know the correct parametrization of their model—especially a fitting minimum support threshold—in advance, they present different search strategies (one-pass, LAMP, and LEAP). Some of them repeatedly run a frequent subgraph pattern mining algorithm with different minimum support values.

[Has+07] Hasan et al. "ORIGAMI: Mining Representative Orthogonal Graph Patterns". 2007

[Sug+15] Sugiyama et al. "Significant Subgraph Mining with Multiple Testing Correction". 2015

On the other hand, many algorithms for superproblems are directly integrated into the pattern space exploration. The commonality here is, that the superproblems objective function has some implication on the minimum support threshold. Thus, they still use the exploration strategy of frequent subgraph pattern mining algorithms with a support-based pruning, but prune subspaces by deducing frequency bounds that are either global or specific to the currently explored subspace. For example, Thoma et al. [Tho+10] used the gSpan algorithm to mine a set F of discriminative subgraph pattern features for graph classification. They modeled an internal feature selection quality criterion called *Correspondence-based Quality Criterion* (CORK) to minimize correspondences, which are pairs of graphs from different graph classes with the same subgraph pattern features regarding F. They iteratively mine single optimal patterns, subsequently adding them to their feature set F. Given a pattern P, they deduced a minimal support threshold for the different graph classes for each subpattern $S \sqsubset P$ based on the number of eliminated correspondences of P subject to F. Pan et al. [Pan+15] presented another feature selection algorithm named *Regularized Loss Minimization Subgraph Selection* which integrated the model learning into the mining process of gSpan in order to determine the performance of the selected features specific to the selected machine learning algorithm. Terada, duVerle, and Tsuda [TdT16] developed a significant pattern mining algorithm with confounding variables that is not specific to graph pattern mining. However, they evaluated their approach by integrating it into the gSpan algorithm deducing a frequency bound. Tsuda and Kudo [TK06] and Tsuda and Kurihara [TK08] extended to gSpan algorithm to support weighted subgraph patterns for clustering purposes.

[Tho+10] Thoma et al. "Discriminative frequent subgraph mining with optimality guarantees". 2010

[Pan+15] Pan et al. "Finding the best not the most: regularized loss minimization subgraph selection for graph classification". 2015

[TdT16] Terada, duVerle, and Tsuda "Significant Pattern Mining with Confounding Variables". 2016

[TK06] Tsuda and Kudo "Clustering graphs by weighted substructure mining". 2006

[TK08] Tsuda and Kurihara "Graph Mining with Variational Dirichlet Process Mixture Models". 2008

2.6.3 Graph Clustering

Clustering is a broadly studied data mining topic with a plethora of variants and specializations. With the variety of methods comes a variety of applications, such as data summarization, stratified sampling, visual analytics, classification, image segmentation, anomaly detection, novelty analysis of datasets, community detection, or recommender systems, just to name a few. Clustering is often used as an intermediate or pre-processing step for other data mining tasks or as a speed up-technique for algorithms that benefit from pre-partitioned data.

Definition 2.20 (Clustering). *In most common terms, clustering refers to a set of problems and methods to group (data) objects of a dataset $\mathcal{X}$, i.e., a multiset of objects, into a family of subsets $\mathcal{C} = \{\!\{C_1, \ldots, C_c\}\!\}$ with $\forall C \in \mathcal{C} : C \subseteq \mathcal{X}$ based on some notion of homogeneity. A family member $C \in \mathcal{C}$ is called cluster and $\mathcal{C}$ is called clustering of $\mathcal{X}$.*

The multiset notion for $\mathcal{C}$ is necessary since hierarchical clustering methods might enumerate the same cluster twice for different hierarchy levels. Some clustering methods annotate each cluster with additional properties, such as cluster descriptions in the form of centroids or cluster features (e.g., [Llo82; ZRL96]). Those cluster descriptions are sometimes given in addition to $\mathcal{C}$ as part of the clustering algorithm's output.

[**Llo82**] Lloyd, "Least squares quantization in PCM". 1982

[**ZRL96**] Zhang, Ramakrishnan, and Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases". 1996

The specific definition of homogeneity differs for different clustering problems or methods. The concept is often closely related to some notion of similarity (or distance cf., paragraph 2.6.3.1.3) or otherwise somehow linked objects. Given some underlying truth of a dataset, clusters should be homogeneous in terms of some (unknown) class labels or generative distributions. As such, clustering is also referred to as unsupervised classification. While supervised classification extracts a classification model—i.e., a model that assigns class labels to objects—based on specific label information given for a training dataset, the absence of such information led to the term unsupervised classification for clustering. As such, clustering can only rely on regularities of the objects in relative terms.

Definition 2.20 is sometimes complemented by a separation criterion, which states that clusters should not only be homogeneous, but separated as well. If not given explicitly or implicitly in the clustering problem definition, the concept of separation is also used to evaluate the quality of clusterings (cf., section 2.6.3.5). Similar to homogeneity, separation is not universally defined and the specific definition strongly depends on the specific clustering method, especially w.r.t. the assumptions about the cluster shape (cf., paragraph 2.6.3.1.4) or the underlying generative model. Separation definitions embrace concepts such as the pairwise distances between cluster centers in relation to the compactness of clusters, the pairwise distances between cluster members of different clusters (e.g., average or minimum), non-dense regions between dense clusters, and others. More generally speaking, sets of objects from different clusters should be less homogeneous than the clusters themselves.

This thesis focuses on graph clustering, i.e., clustering problems and methods that deal with graph data.

Definition 2.21 (Graph (Set) Clustering). *The term graph (set) clustering refers to a specialized clustering problem or method, for which the dataset $\mathcal{X}$ consists of graphs, i.e., is a graph dataset $\mathcal{G}$ as defined in section 2.5.*

It should be noted, that the term graph clustering is ambiguously used in literature. It is also used for clustering methods, which cluster vertices of a single input graph based on intra-graph properties, such as connectivity. To avoid such ambiguity, this

thesis will refer to such types of clusterings as *vertex clusterings* and use the term graph clustering or graph set clustering solely as defined in definition 2.21.

2.6.3.1 Properties of Clustering Methods

Given the broad clustering definition above, specific clustering methods and their results can be categorized with the help of the following properties. This section does not claim to be comprehensive w.r.t. to clustering methods but presents concepts central for the latter discussion of the StruClus algorithm in chapter 3.

2.6.3.1.1 Partitioning, Overlapping, and Fuzzy Clustering Clustering methods can be distinguished by the way objects are assigned to clusters. Partitioning clustering methods follow the mathematical partition definition of a multiset, i.e., a clustering $\mathcal{C} := \{\!\{C_1, \ldots, C_c\}\!\}$ of a graph dataset $\mathcal{G}$ has the properties:

$$\forall i, j \in \{1, \ldots, c\}, i \neq j : C_i \cap C_j = \emptyset \tag{2.16}$$

$$\bigcup_{C \in \mathcal{C}} = \mathcal{G} \tag{2.17}$$

$$\forall i \in \{1, \ldots, c\} : C_i \neq \emptyset \tag{2.18}$$

Prominent examples for partitioning clustering algorithms are the k-Means or Lloyd algorithm [Llo82] and the family of SAHN clustering algorithms [And73; DE84]. In the latter, hierarchical, case, the partitioning nature is restricted to flat clusterings, that can be derived from the hierarchy by cutting the hierarchy at a certain similarity threshold (cf., paragraph 2.6.3.1.2). Overlapping clusterings on the other hand may have non-empty overlaps between clusters, i.e., the property of eq. (2.16) is not guaranteed. For example, the structural clustering algorithm of Seeland et al. [See+10] assigns objects to multiple clusters if different structural features are shared with different clusters. Fuzzy clustering methods do not have a fixed assignment to individual clusters, but a weighted or probabilistic assignment to each cluster, e.g., the fuzzy c-means algorithm [Dun73]. This view is closely related to a specific view on datasets, which can be seen as a result of an underlying generative mixture model. If such a model is constructed of overlapping individual distributions, each data point might be the outcome of multiple individual experiments with certain probabilities.

[Llo82] Lloyd, "Least squares quantization in PCM". 1982

[And73] Anderberg, *Cluster Analysis for Applications*. 1973

[DE84] Day and Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods". 1984

[See+10] Seeland et al. "Online Structural Graph Clustering Using Frequent Subgraph Mining". 2010

[Dun73] Dunn, "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". 1973

2.6.3.1.2 Flat and Hierarchical Clusterings Given the unsupervised nature of clustering problems, it is often a priori unclear which degree of granularity fits best the higher-level task at hand. Even if a flat clustering algorithm provides some parameters to adjust the granularity, the resulting clusterings with different granularities might be quite different in terms of common cluster boundaries and hard to compare to each other. Opposed to flat clustering methods, hierarchical methods thereby calculate not only a single clustering result, but a nested clustering tree hierarchy such that multiple flat clusterings can be obtained on different granularity levels. The hierarchical nature of the results helps to relate individual clusters of different levels to each other.

Common classes of hierarchical clustering methods are agglomerative and divisive methods. The former methods start with individual objects as clusters on the lowest level and successively merge these clusters up to a certain level of granularity. The latter methods start with a single cluster of the complete dataset and successively split the clusters until some abortion criterion is reached. A popular class of agglomerative methods are subsumed under the term SAHN clustering [And73; DE84] (Sequential Agglomerative Hierarchical Non-Overlapping clustering). SAHN methods derive cluster to cluster distances based on distances between individual objects. The resulting hierarchy is a binary tree, which can be visualized by a dendrogram (for an example, see fig. 1.5).

Another approach to hierarchical clustering is the density-based OPTICS clustering algorithm [Ank+99]. Here the hierarchy rank describes the degree of density, which is required to form a cluster. With higher densities, more and more clusters are split by non-dense regions and more objects are classified as noise (cf., paragraph 2.6.3.1.5).

[**Ank+99**] Ankerst et al., "OPTICS: Ordering Points To Identify the Clustering Structure". 1999

2.6.3.1.3 View on the Data Clustering methods utilize specific abstractions of the datasets. Most commonly, data is represented by vectors (e.g., k-Means [Llo82] and CLIQUE [Agr+98]) or pairwise distances (e.g., DBSCAN [Est+96] or Jarvis-Patrick [JP73]).

In the former case, not only vectorial data that is naturally represented in such form (e.g., data points in a predefined vector space such as L^2) is applicable to the clustering algorithm. Various types of data can be mapped to a vectorial representation given a numerical feature extraction method for the objects at hand. Such numerical features can be combined to so-called feature vectors, where each feature has a fixed position or dimension in the vector. The associated vector space is then called feature space. For a more detailed discussion of relevant feature extraction methods in the context of graph data, see section 2.6.3.3. Vector spaces benefit from the possibility to add and scale vectors. For example, this enables explicit computation of centroids, i.e., mean vectors, that can serve as cluster descriptions or identifiers. Furthermore, it is possible to derive topological subspaces of vectors by selecting certain features, dimensions, or linear combinations of them (cf., section 2.6.3.2). Normed vector spaces are equipped with a norm (sometimes called length or magnitude), which maps a vector to an absolute scalar value. Most important this implies a distance function via subtraction of vectors, which in turn can be used to measure homogeneity.

[**Llo82**] Lloyd, "Least squares quantization in PCM". 1982

[**Agr+98**] Agrawal et al., "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". 1998

[**Est+96**] Ester et al., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". 1996

[**JP73**] Jarvis and Patrick, "Clustering Using a Similarity Measure Based on Shared Near Neighbors". 1973

> **Definition 2.22 (Distance Function).** *Given a dataset $\mathcal{X}$, a distance function is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where low values represent similar or homogeneous and high values represent dissimilar or inhomogeneous object pairs.*

A similarity function is of the same kind but reverses the direction of interpretation, i.e., low values represent distant and high values represent similar objects. While clustering methods for vector space may also utilize distances to measure homogeneity, some clustering methods solely rely on pairwise distances. In this case, the dataset provided to the clustering algorithm is either a distance matrix or a multiset of

objects and a distance function. In both cases, the specific distance function is often a parameter of the clustering algorithm and can be used to adapt the clustering algorithm to the user's needs or specific problem instances. The quadratic nature of the distance matrix also leads to a quadratic best-case complexity. However, some algorithms can omit the computation of all pairwise distances, utilizing special properties of distance functions. Maybe the most prominent example is the use of the triangle inequality of distance metrics, e.g., to build index structures that can answer range queries to retrieve similar objects in sublinear time.

Definition 2.23 (Distance Metric). *A distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a distance metric, iff the following properties hold for all $O, Q, T \in \mathcal{X}$:*

$$d(O,Q) = 0 \Leftrightarrow O = Q \qquad \textit{identity of indiscernibles} \qquad (2.19)$$

$$d(O,Q) = d(Q,O) \qquad \textit{symmetry} \qquad (2.20)$$

$$d(O,Q) \leq d(O,T) + d(T,Q) \qquad \textit{triangle inequality} \qquad (2.21)$$

A fourth property can be derived from the above properties:

$$d(O,Q) \geq 0 \qquad \textit{non-negativity} \qquad (2.22)$$

It is important to notice that metrics themselves are often defined on specific forms of data representation or interpretation. For example, the property described in eq. (2.19) applied to some graph objects G and H interpreted as graph pattern representatives does imply an isomorphism between G and H. However, when G and H are interpreted as graphs, they must also have the same representation. Additionally, many distances applied to the graph domain have a lossy intermediate representation, such as feature vectors. In this case, non-isomorphic graphs may map to the same intermediate representation and only the intermediate representation must be equal for a metric distance of 0.

A third data representation, that is sometimes used by clustering algorithms—such as kernel k-Means [Gir02]—is a (graph) kernel. In this case, the dataset domain is mapped to an implicit feature space with possibly infinite dimensions. A kernel is a function, that expresses the inner product in this feature space. The associated kernel matrix must be positive semi-definite and symmetric. An advantage of kernel clustering methods in comparison with classical vector space is, that kernels can be easily combined to transform the feature space, making it very flexible in terms of adaptation to the task at hand. Such transformations enable cluster algorithms that are bound to specific shapes of clusters to overcome this limitation in the original representation. With kernels, it is still possible to express distances to centroids objects without explicitly computing the centroid vector in the feature space, which makes kernel methods superior over pure distances when expressing relative positions.

[Gir02] Girolami, "Mercer kernel-based clustering in feature space". 2002

A fourth view on clustering graph data is the structural view, which is a central concept for chapter 3. In this case, the graph data is directly interpreted in the graph domain with graph-theoretic concepts like subgraph isomorphisms, common

subgraphs, or median graphs[11]. Cluster homogeneity can be expressed by shared structural features like common and frequent subgraphs or in similarity terms given some representative or reference graph object. The avoidance of abstraction regarding data representation makes structural clustering algorithms very transparent in terms of interpretation. If structural commonalities are explicitly part of the clustering result, these commonalities can be interpreted directly in the domain of application. Interpretability is sometimes a problem with intermediate representations. For example, it is usually not possible to map a vector representation back to the graph domain. This makes it hard to interpret cluster descriptions, such as centroids or cluster features. Additionally, the usually lossy transformations to a feature space is a source of inaccuracy w.r.t. to the clustering results.

2.6.3.1.4 Shape of Clusters Clustering methods can be distinguished by the shape of clusters they can find. This shape is strongly connected to the definition of homogeneity a clustering algorithm has.

Many well know clustering methods—such as K-MEANS [Llo82]—assume, that clusters do have a spherical shape under some L^p norm. Thus, cluster objects are assumed to be contained in a circle around some center or at least assigned to the closest cluster center. To some extent, this concept is also applicable to clustering algorithms relying solely on metric distances, where a sphere can be described by objects with a certain distance to a reference object (e.g., median object). Since the K-MEANS objective, to minimize the squared mean euclidean errors w.r.t. cluster centroids, is equivalent to the minimization of cluster variance, this view aligns with the idea that all objects from the same cluster should be pairwise similar. Clustering algorithms that minimize the maximum pairwise distances in a cluster are also said to produce compact clusters.

Of course, one can think of other shapes that form intuitive clusters, e.g., two parallel lines, which never touch or a circle inside another circle. Other shapes are often not explicitly defined by the clustering algorithms. Instead, more general approaches are often used to express arbitrary shapes of clusters. One class of these clustering methods transform the input space, such that spherical clusters in the transformed space are non-spherical in the original space. An example is the KERNEL K-MEANS algorithm [Gir02], which allows the combination of different kernels to emphasize close distances, e.g., by applying a Gaussian kernel. Other clustering algorithms exploit concepts of neighborhood. For example, density-based methods like DBSCAN [Est+96] or OPTICS [Ank+99] use the concept of ϵ-neighborhood (fixed radius) to identify dense regions of a dataset, i.e., regions with a high number of other objects in their reciprocal neighborhoods. Clusters are then defined to be densely connected objects in a transitive manner. Thus, dense clusters are separated by non-dense borders. Also, spectral methods employ the concepts of neighborhoods by constructing neighborhood graphs (e.g., ϵ- or k-nearest neighborhoods) of the

[Llo82] LLOYD, "LEAST SQUARES QUANTIZATION IN PCM". 1982

[Gir02] GIROLAMI, "MERCER KERNEL-BASED CLUSTERING IN FEATURE SPACE". 2002

[Est+96] ESTER ET AL., "A DENSITY-BASED ALGORITHM FOR DISCOVERING CLUSTERS IN LARGE SPATIAL DATABASES WITH NOISE". 1996

[Ank+99] ANKERST ET AL., "OPTICS: ORDERING POINTS TO IDENTIFY THE CLUSTERING STRUCTURE". 1999

[11] Median graphs are defined w.r.t. some distance function. They can be either a set median or general median graph, i.e., the search space is either the cluster objects or the subgraph pattern space

dataset, i.e., sparsening the complete distance graph. This sparse neighborhood graph is then embedded into the euclidean space via spectral analysis and again clustered with a standard clustering method like k-Means, or SAHN.

2.6.3.1.5 Noise In the context of clustering, noise can be defined as undesired random defects in the dataset. This can be observed in form of additive noise w.r.t. numerical features, outliers, or completely random objects that should not be part of the dataset. Some clustering methods are known for their fragility w.r.t. noise. For example, single linkage SAHN clustering can dramatically change its results if otherwise separated clusters are bridged by noise objects. For centroid-based methods, noise objects far away from the real clusters can drag the centroids away from their real cluster center, if they are included in the centroid calculation. However, the degree of noise influence on the clustering results is not only a matter of the used algorithm but a product of the dataset structure and the clustering algorithm. For example, for datasets with almost equidistant characteristics, very small changes to the distances change the clustering results a lot. Contrariwise, well-separated clusters would still be separated from each other given some degree of perturbation. Bilu and Linial [BL10] introduced the concept of α-resilience, which states the robustness of a clustering outcome under a distance perturbation by a maximal factor of α.

[BL10] Bilu and Linial "Are Stable Instances Easy?" 2010

For this reason, perturbation is often used to judge the robustness of clustering methods or results. When the clustering results are not stable under small perturbation, this might be a sign of a weak support for the clustering result.

Clustering methods cannot only be discriminated by their resilience of clustering results under perturbation. In addition to this general perturbation resilience, some clustering methods actively identify noise objects via some explicit definition of noise. For example, the density-based DBSCAN algorithm [Est+96], does specify noise as points in non-dense regions of the dataset. Some other methods create an explicit noise cluster that attracts random objects, such that they are most likely excluded from real clusters. For example, Davé [Dav91] added a noise prototype object with equal distance to all objects to the dataset of (fuzzy) k-Means clustering and fixed this object to be the centroid of an explicit noise cluster.

[Est+96] Ester et al., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". 1996

[Dav91] Davé "Characterization and detection of noise in clustering". 1991

2.6.3.2 High Dimensional Datasets

The dimensionality d of a vector space is defined to be the cardinality of its basis. In clustering literature, however, dimensionality is often referred to as a property of a vector representation, i.e., it is assumed, that a d-component vector is stemming from the vector space $\mathbb{R}^d$ with standard basis. During this thesis, this will be denoted by representational dimension.

There exist several reasons for high-dimensional datasets. For example, feature extraction methods for complex objects, such as graphs, tend to produce a large number of distinct features. Subspace and projected clustering algorithms are particularly designed to address challenges, that arise in the context of high-dimensional datasets. Before these clustering methods are described in paragraph 2.6.3.2.3, the challenges of high dimensional datasets will be subject to discussion.

2.6.3.2.1 Irrelevant Features Due to the unsupervised nature of clustering problems, a clustering algorithm cannot discriminate relevant from irrelevant features for the specific task at hand. Thus, adding irrelevant features to a feature vector is not only superfluous but may actually harm clustering quality. For example, if the property of interest is the aerodynamic behavior of the objects, adding a color feature to a feature space, that otherwise contains only shape features, might cause a split of clusters with homogeneous shape into different colors. Even worse, it may let the color become the dominant feature for the clustering result, overshadowing the true property of interest. Clustering methods are often used to find unknown regularities in datasets with low a priori knowledge about them. For these reasons, it might not be possible to select a small and relevant feature set for the task at hand. Instead, having feature-rich datasets might actually reveal regularities, that would have stayed undiscovered otherwise. Thus, including irrelevant features cannot be ruled out in general.

2.6.3.2.2 Concentration Effect and Intrinsic Dimensionality If features or representational dimensions are statistically independent, most common norms (including L^p with $p > 1$) converge towards the value of the mean vector with an increasing number of features. Consequentially, derived distances between objects also converge towards a common value if more and more independent features are added and the relative contrast $\frac{d_{\max} - d_{\min}}{d_{\min}}$ of a distance d converges towards 0. This effect is called concentration effect [Bey+99; ZSK12] and is associated with a bad clusterability since well-separated clusters are hard to find under these circumstances. Instead, every subset of objects shows a similar homogeneity.

[**Bey+99**] Beyer et al., "When Is "Nearest Neighbor" Meaningful?" 1999

[**ZSK12**] Zimek, Schubert, and Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data". 2012

As mentioned before, the representational dimension of a dataset is not the same as the dimension of a well-defined vector space. However, the underlying formal vector space of the dataset is usually unknown and it is simply assumed to be the most general one, i.e., $\mathbb{R}^d$. Nevertheless, it is possible, that the dataset is contained in a subspace of $\mathbb{R}^d$ with a smaller basis. Thus, the data points might be embedded into another vector space $\mathbb{R}^e$ with $e < d$ via (linear) transformation. Such a transformation can be distance preserving, i.e., the distance of two points is equal in the original and transformed space. In the case that a distance preserving embedding into a lower-dimensional vector space exists, clustering algorithms relying on the relative distances produce the same results for the original and the transformed datasets. Regarding the aforementioned concentration effect, this means that the original dataset has (linearly) dependent dimensions, that are not statistically independent and consequentially do not contribute to the concentration effect. Thus, a dataset with a high number of representational dimensions, for which a distance preserving transformations to a low-dimensional vector space exists, does not suffer from the concentration effect.

However, the notion of distance preserving is quite strict in the sense of the concentration effect. There might also exist embeddings to much lower-dimensional spaces when the distance preserving property is relaxed and the distance in the embedded space does only approximate the original distance. For example, while two vector dimensions in a dataset might be linearly independent, they might still be highly

correlated and dependent in the probabilistic sense. Additionally, some dimensions or linear combinations of them might have a very low variance in comparison with another dimension. In this case, the low-variance dimensions do not contribute much to the relative contrast of the distance as well. This leads to the term intrinsic dimensionality. An intrinsic dimensionality measures the underlying dimensionality of the dataset in a fuzzy way. There exist several definitions of intrinsic dimensionality (e.g., [FO71; Ams+15; TS22]). During this thesis, the definition of Chávez and Navarro [CN01] is used, since it is applicable to general distance metrics and not limited to vector space. Thus, it is also possible to measure the dimensionality of datasets that are solely given in form of pairwise distance.

[**FO71**] Fukunaga and Olsen, "An Algorithm for Finding Intrinsic Dimensionality of Data". 1971

[**Ams+15**] Amsaleg et al., "Estimating Local Intrinsic Dimensionality". 2015

[**TS22**] Thordsen and Schubert, "ABID: Angle Based Intrinsic Dimensionality — Theory and analysis". 2022

[**CN01**] Chávez and Navarro "A Probabilistic Spell for the Curse of Dimensionality". 2001

Definition 2.24 (Intrinsic Dimensionality). *The intrinsic dimension of a dataset in a metric space is* $\mathrm{iD} := \frac{\mu^2}{\sigma^2}$*, where* μ *and* σ^2 *are the mean and variance of its histogram of distances. [CN01]*

The definition directly aligns with the above-discussed concept of relative contrast. If the variance is low compared to the mean value of distances, this means a low relative contrast in a probabilistic sense. Datasets formed by d-component random vectors in L^p space result in an expected intrinsic dimensionality $\mathrm{iD} \in \Theta(d)$

2.6.3.2.3 Subspace and Projected Clustering Methods Clustering methods for high dimensional data need to cope with the above-mentioned challenges, i.e., irrelevant attributes and the concentration effect in high dimensional spaces. Subspace and projected clustering methods identify clusters in subspaces of the original dataset. The idea is, that subspaces of relevant features may exist in which homogeneous clusters can be found. Thereby, each cluster is assigned to a subspace. However, the concept of a global cluster separation can become problematic, since different clusters from different subspaces become incomparable in their cluster description. For example, if clusters are described by centroids in a subspace, the centroids of different clusters contain incomparable dimensions or features. The same is true for separation definitions involving individual cluster elements. For this reason, subspace and projected clustering methods usually solely depend on some cluster local concept of homogeneity or separation, e.g., a fixed radius threshold for centroid-based methods, densely connected regions [e.g., KKK04; Ass+07], or fixed grids [e.g., Agr+98] that are only separated in the given subspace. The difference between subspace and projected clustering methods is, that the former enumerate all clusters in all subspaces, while the latter only assigns a subspace to each cluster and dataset object. Thus, subspace clustering methods can be seen as a special form of overlapping or hierarchical clustering methods, while projected clustering methods partition the data objects. The projected clustering task was introduced by [Agg+99] with the PROCLUS algorithm. Similar to k-Medoid clustering, they try to find subspaces for a set of medoid objects, such that the distance of the dataset objects to these medoid is minimized in the assigned subspace. To cope with the combinatorial explosion of subspaces with the number of considered dimensions, most subspace clustering

[**KKK04**] Kailing, Kriegel, and Kröger, "Density-Connected Subspace Clustering for High-Dimensional Data". 2004

[**Ass+07**] Assent et al., "DUSC: Dimensionality Unbiased Subspace Clustering". 2007

[**Agr+98**] Agrawal et al., "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". 1998

[**Agg+99**] Aggarwal et al., "Fast Algorithms for Projected Clustering". 1999

methods explore subspace combinations in a frequent itemset mining apriori style bottom-up strategy [e.g., KKK04], exploiting monotonic properties of their clustering definition w.r.t. the subspace ordering. Furthermore, most subspace clustering methods are axis-parallel [e.g., CFZ99], which means that only combinations of representative dimensions are considered as subspaces.

[**CFZ99**] Cheng, Fu, and Zhang, "Entropy-based Subspace Clustering for Mining Numerical Data". 1999

2.6.3.3 Graph Feature Extraction Methods

This section will cover the most important feature extraction approaches in the context of drug discovery. Most important, it will cover extraction methods used during the evaluation of the StruClus clustering algorithm in section 3.2.9.

Feature extraction methods for graph data are roughly categorized by the way individual features are extracted. Furthermore, the methods can be distinguished by their type, i.e., whether they encode binary or numerical features. While binary feature vectors most commonly encode the presence or absence of some extracted feature, numerical features are often counts of some kind or encode more abstract domain-specific numerical properties (such as molecular weight).

Structural feature extraction methods encode subgraph pattern features, where each dimension encodes the presence or the number of occurrences in a graph. Instead of enumerating the complete subgraph pattern space, the features are selected with different objectives. For example, frequent, significant, or discriminative patterns that are mined for a particular problem instance (cf., section 2.6.1, [e.g., Tho+10; Sug+15]) can be used as structural features. Other methods resort to the (complete) enumeration of restricted classes of subgraph patterns, such as graphlets (small graph patterns with a maximum size, [e.g., She+09]), paths [e.g., GRB06], or trees [e.g., KKM11]. Furthermore, feature extraction methods with manually selected feature sets exist, especially in the context of drug discovery [e.g., Dur+02], where relevant features for drug-likeliness and other properties are identified w.r.t. domain knowledge.

[**Tho+10**] Thoma et al., "Discriminative frequent subgraph mining with optimality guarantees". 2010

[**Sug+15**] Sugiyama et al., "Significant Subgraph Mining with Multiple Testing Correction". 2015

[**She+09**] Shervashidze et al., "Efficient graphlet kernels for large graph comparison". 2009

[**GRB06**] Gedeck, Rohde, and Bartels, "QSAR - How Good Is It in Practice? Comparison of Descriptor Sets on an Unbiased Cross Section of Corporate Data Sets". 2006

[**KKM11**] Klein, Kriege, and Mutzel, "CT-Index: Fingerprint-based Graph Indexing Combining Cycles and Trees". 2011

[**Dur+02**] Durant et al., "Reoptimization of MDL Keys for Use in Drug Discovery". 2002

[**Stö+19**] Stöcker et al., "Protein Complex Similarity Based on Weisfeiler-Lehman Labeling". 2019

[**She+11**] Shervashidze et al., "Weisfeiler-Lehman Graph Kernels". 2011

[**RH10**] Rogers and Hahn, "Extended-Connectivity Fingerprints". 2010

[**LWH03**] Luo, Wilson, and Hancock, "Spectral embedding of graphs". 2003

Another class of graph feature extraction methods is based on vertex labeling. Probably the most simple form is the plain usage of vertex labels contained in a graph [Stö+19]. However, the absence of structural information limits those approaches. More sophisticated vertex labeling extraction methods encode structural information in the vertex labeling, by encoding properties of the neighborhood into the vertex labels. Maybe the most prominent example are methods based on the Weisfeiler-Lehman isomorphism test, which iteratively derives vertex labels by encoding distinct neighborhoods. Intended as an isomorphism test, these features are commonly used to compare graphs [e.g., She+11; Stö+19]. In the context of drug discovery, similar methods (e.g., extended connectivity fingerprints [RH10]), are known under the term circular fingerprints.

Additionally to the above-mentioned feature extraction methods, there exist some more exotic approaches, which are not fully covered here. For example, spectral analysis of the adjacency matrix [e.g., LWH03] can be used to extract some features from the eigenvectors and values. However, spectral approaches are limited to graph connectivity and cannot encode label information.

2.6.3.4 Graph Distances

As discussed above, feature extraction methods can be used to derive feature vectors for graphs. Distances and similarities for feature-vectors and -sets, e.g., the Jaccard-Coefficient or the L^p-norm, can be applied to them for graph similarity analysis. In addition graph distances, i.e., distances directly applicable to the graph domain, exist, that do not rely on such an intermediate representation.

[SF83] Sanfeliu and Fu, "A Distance measure between attributed relational graphs for pattern recognition." 1983

[BA83] Bunke and Allermann, "Inexact graph matching for structural pattern recognition". 1983

The graph edit distance [SF83; BA83] is a well-known example. It measures similarity by the means of edit operations (e.g., insertions, deletions, or substitutions of vertices and edges) to transform one graph into the other. To derive a distance measure, the minimum number of necessary edit operations is calculated. Multiple follow-up publications exist, e.g., to allow sophisticated weighting of edit operations or other edit operations (such as merging and splitting of vertices).

[Bun97] Bunke, "On a relation between graph edit distance and maximum common subgraph". 1997

[BS98] Bunke and Shearer, "A graph distance metric based on the maximal common subgraph". 1998

[Wal+01] Wallis et al., "Graph distances using graph union". 2001

[FV01] Fernández and Valiente, "A graph distance metric combining maximum common subgraph and minimum common supergraph". 2001

Closely related to the graph edit distance are distance measures based on maximum common subgraphs [Bun97]. Since a common subgraph can be interpreted as structural commonality, it is possible to relate the size of such commonality to the size of the compared graphs. Several metrics [e.g., BS98; Wal+01; FV01] have been proposed which are based on this concept:

$$1 - \frac{|\mathrm{mcs}(G,H)|}{\max\{|G|,|H|\}} \qquad \text{[BS98]}$$

$$1 - \frac{|\mathrm{mcs}(G,H)|}{|G| + |H| - |\mathrm{mcs}(G,H)|} \qquad \text{[Wal+01]}$$

$$|G| + |H| - 2|\mathrm{mcs}(G,H)| \qquad \text{[FV01]}$$

In these definitions, $\mathrm{mcs}(G,H)$ is defined to be a maximum common subgraph between two graphs G and H. An interesting property of these distances is the exchangeability of the used maximum common subgraph concept, i.e., using an induced / non-induced or a connected / disconnected common subgraph isomorphism. Most importantly, this includes domain-specific adoptions of the maximum common subgraph concept such as presented by Droschinsky, Kriege, and Mutzel [DKM18]. However, not every maximum common subgraph definition preserves the metric properties.

[DKM18] Droschinsky, Kriege, and Mutzel "Largest Weight Common Subtree Embeddings with Distance Penalties". 2018

2.6.3.5 Validation Measures

To evaluate a clustering method it is vital to have some validation measure to judge the clustering results in terms of quality. Validation measures can be divided into internal and external validation measures.

Internal validation measures express the quality of a clustering based on some notion of homogeneity (and separation), i.e., some function $m : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$. Consequentially, such an internal evaluation method should reflect the optimization criterion of a clustering method. This includes previously mentioned clustering properties regarding allowed overlap between clusters, the handling of special noise clusters, the shape of clusters, and the applicability to subspaces. The standalone value of an internal validation method is often not helpful to judge the quality of a clustering method on its own, since the homogeneity of a clustering can be only as high as the internal

properties of the underlying dataset allow. However, to compare different clustering methods, their parametrization, or dataset properties, clustering results can be ranked by their internal validation value. One must be very careful with the interpretation of such rankings though. For example, different clustering methods may have different optimization criteria and a single internal validation measure may not fit both methods. The same is true for parametrization. The k-Means clustering algorithm, for example, minimizes intra-cluster variance. This criterion is monotonically increasing with the number of clusters. Thus, comparing two clusterings with a different number of clusters, with an internal validation measure that reflects the k-Means optimization criterion, is not possible based in the raw validation values without taking further aspects into account. It can be concluded, that there is not one internal validation measure that fits all clustering methods and the interpretation of such validation values must include knowledge about the validation measure itself.

External evaluation methods measure the degree of agreement between two clusterings, i.e., some function $m : \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ to compare two families of sets. A typical approach to judge the quality of a clustering with an external validation measure, is the use of a ground truth, i.e., the comparison of the clustering method's results with pre-defined class labels of an evaluation dataset. On the one hand, this elides the necessity to explicitly match the optimization criterion of the clustering method in comparison with internal validation measures. For example, whether the shape of a cluster is spherical or arbitrary shaped does not matter. It is simply the ground truth that reflects this property. On the other hand, this shifts the challenge to the availability and selection of a fitting ground truth. If the agreement of a clustering with a ground truth is high, one can conclude that the clustering produces meaningful results. However, if the agreement is low, it does not necessarily mean that the clustering results are meaningless. As a consequence of the unsupervised nature, the clustering method might just have revealed another aspect of the dataset. This is especially true, if complex datasets are evaluated, where multiple subspaces with meaningful clusters exist. External evaluation methods may also be biased towards some clustering parameters. Most prominently, the number of clusters is known to cause a bias in most external evaluation measures. For example, external evaluation measures often show systemically higher or lower values of agreement for random clusterings with a high number of clusters in comparison with a low number of clusters. For this reason there exist measurements that try to eliminate such effects, e.g., by normalization with some expected value under the random clustering assumption. Still, one needs to be careful when interpreting external clustering values for clusterings which largely differ in the number of clusters, the size distribution, or other parameters of a clustering.

During this thesis, three external validation measures are utilized: (a) The Fowlkes & Mallows Index (FM) [FM83], (b) the Normalized Variation of Information measure (NVI) [Kra+05], (c) and the Purity measure [HZ13]. All measures are normalized to the range $[0, 1]$ (FM, and NVI) or $(0, 1]$ (Purity). NVI is the normalized version of variation of information measure (VI) [Mei07] and is a distance. During evaluation all external validation values are given in form of similarities to avoid confusion, i.e., a value of 1 corresponds to a perfect agreement between the two clusterings and a value

[FM83] Fowlkes and Mallows, "A Method for Comparing Two Hierarchical Clusterings". 1983

[Kra+05] Kraskov et al., "Hierarchical clustering using mutual information". 2005

[HZ13] Hui and Zhongmon, "Clustering Validation Measures". 2013

[Mei07] Meilă, "Comparing clusterings—an information based distance". 2007

of 0 indicates no agreement at all. The similarity measure 1 − NVI is also called joint normalized mutual information measure [NEB10].

[NEB10] Nguyen, Epps, and Bailey, "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". 2010

FM and NVI measures are chosen in order to have some variation in the type of measure. FM follows the counting-pairs approach and NVI is an information-theoretic measure. Counting-pair approaches are defined over pairs of objects (O, Q) and measure whether these pairs are in a common or in different clusters. Each pair can fall into one of three categories: (a) O and Q are in the same cluster in both clusterings, (b) O and Q are in different clusters in both clusterings, and (c) O and Q are in the same cluster in one clustering, but in different clusters in the other clustering. Items a and b are counted as agreement and item c as disagreement. The exact way these counts are combined differs with different measurements in this class. Information-theoretic measures are defined with the help of information-theoretic concepts like conditional, joined, or mutual entropy. Roughly speaking, they measure the amount of information contained of a clustering structure in relation to another clustering.

Another reason to choose FM and NVI measures for evaluation is their robustness w.r.t. clusterings of different sizes. However, FM and NVI are limited to partitioning clustering algorithms. For this reason, the Purity measure complements the selection to compare the StruClus algorithm (cf., chapter 3) to overlapping clustering methods. As the name suggests, the Purity measure computes a weighted average over the pureness of each cluster, where pureness is defined to be the largest fraction of objects stemming from a single cluster in the other clustering. This makes the Purity measure asymmetric. For example, if a single cluster is split in half, the Purity measure will count one cluster with a pureness of 0.5 or two clusters with a pureness of 1 depending on the directions of measurement. During this thesis, the direction of measurement is always the average pureness of the clusters from the evaluated clustering w.r.t. to the class labels of the ground truth.

Chapter 3

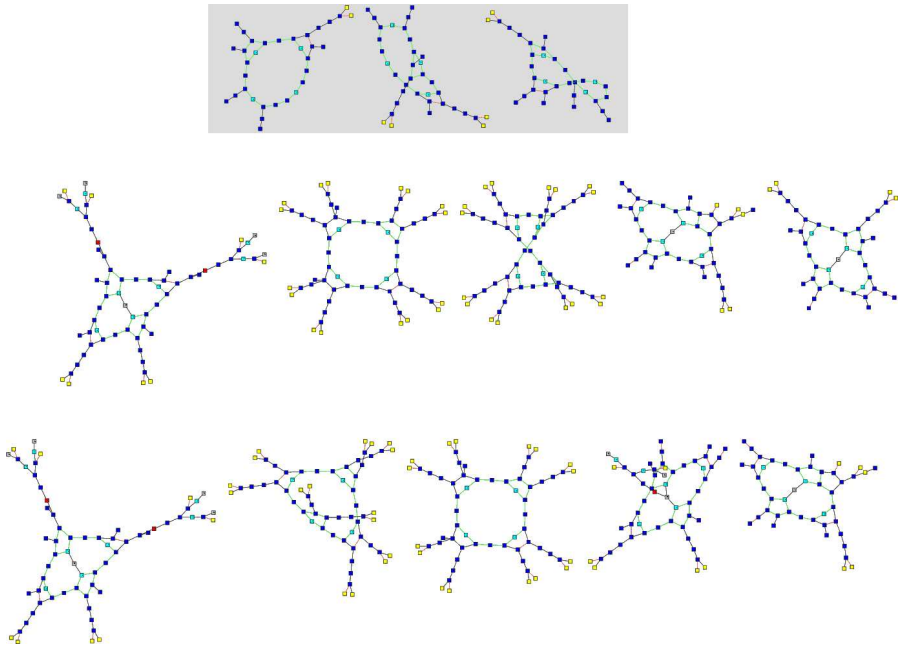
Structural Clustering

As discussed in the introduction (cf., section 1.1.3), cluster analysis plays an important role in drug discovery. It is a central tool to reduce the complexity of large-scale molecular datasets in visual analytics. Furthermore, clustering is used to reason about molecular properties and to infer properties for undiscovered areas of the chemical space.

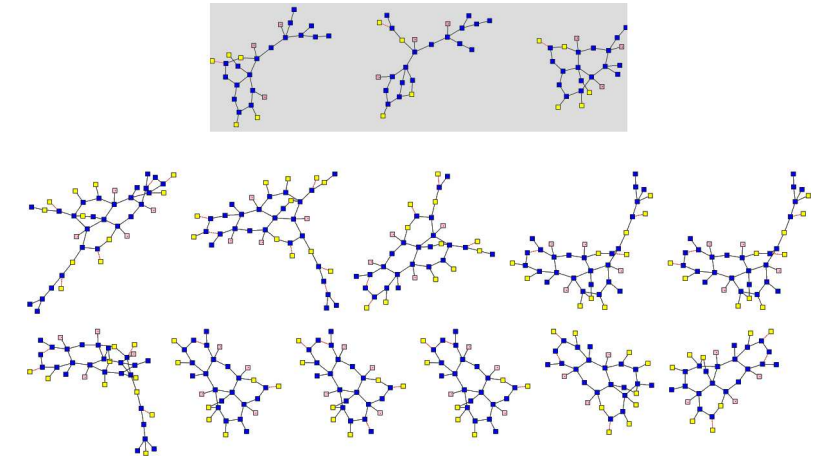
Classical clustering methods rely on data representations in the form of vectors or pairwise distances (cf., paragraph 2.6.3.1.3). For graph datasets, these representations cause limitations in terms of interpretability, since the cluster commonalities cannot be described in the graph domain. Instead, cluster commonalities must be derived manually by observing the cluster content. This somehow contradicts the purpose of complexity reduction. The interpretation of such clusterings often requires the knowledge to interpret vectorial representations and distances, which can be a hard task, especially for a user with no computer science education. The high intrinsic dimensionality of graph datasets [KMS14a] is linked to the so-called concentration effect (cf., section 2.6.3.2) and a bad clusterability. As such, projected or subspace clustering methods are often needed for high-quality clustering results. The transformation to a vectorial representation is usually lossy, which adds an additional source of error in the perturbation-sensitive situation of concentrated distances. Since such lossy vectors cannot unambiguously be transformed back to the graph domain, this is also limiting the interpretability of cluster descriptions like centroids or other cluster features.

[KMS14a] Kriege, Mutzel, and Schäfer, "Practical SAHN Clustering for Very Large Data Sets and Expensive Distance Metrics". 2014

Structural clustering algorithms solve the interpretability issues by describing cluster commonalities in the graph domain. The projection to a selection of structural features can be used for the design of projected structural clustering algorithms. However, their high complexity often limits their applicability to relatively small datasets. Instead, lightweight vectorial algorithms are often the only practical option to cluster large-scale molecular datasets in a reasonable amount of time.



(a) Cluster 1



(b) Cluster 2

Figure 3.1: Structural clusters generated with the StruClus algorithm. Grey boxes show a structural cluster description.

This chapter is based on [SM17] and is dedicated to the scalable structural clustering algorithm StruClus for large datasets of small labeled graphs. Figure 3.1 shows an example of structural clusters generated by StruClus.

[SM17] Schäfer and Mutzel, "StruClus: Scalable Structural Graph Set Clustering with Representative Sampling". 2017

3.1 Related Work

A variety of graph set clustering algorithms have been proposed in recent years. An EM algorithm using a binomial mixture model over very high dimensional feature vectors, which encode all frequent substructures, has been proposed by Tsuda and Kudo [TK06]. Tsuda and Kurihara [TK08] published a follow-up algorithm in 2008 using a Dirichlet Process mixture model. It prunes the set of frequent substructures to reduce the number of representative dimensions of the feature vectors. A k-Median-like graph set clustering algorithm was proposed by Ferrer et al. [Fer+09]. They approximate the discovery of a median graph by embedding graphs into an euclidean vector space. The embedding is performed using the graph edit distance (cf., section 2.6.3.4) to a set of pivot elements. Afterwards, the euclidean median is calculated and an approximate median graph is selected based on the distance to the euclidean median. Seeland et al. [See+10] presented a parallel overlapping graph set clustering algorithm, which greedily adds a graph to an existing cluster whenever a common substructure of a minimum size exists. New clusters are created whenever no existing cluster matches the currently observed graph. Jouili, Tabbone, and Lacroix [JTL10] presented an adoption of the mean shift clustering algorithm to the graph domain. They used the edit path of the graph edit distance, to edit the graphs towards a common median graph. All the previously presented algorithms have in common, that they are not suitable for large-scale datasets. Dataset sizes in the experiments range from a few hundred to a few thousand graphs.

[TK06] Tsuda and Kudo "Clustering graphs by weighted substructure mining". 2006

[TK08] Tsuda and Kurihara "Graph Mining with Variational Dirichlet Process Mixture Models". 2008

[Fer+09] Ferrer et al. "Graph-Based k-Means Clustering: A Comparison of the Set Median versus the Generalized Median Graph". 2009

[See+10] Seeland et al. "Online Structural Graph Clustering Using Frequent Subgraph Mining". 2010

[JTL10] Jouili, Tabbone, and Lacroix "Median Graph Shift: A New Clustering Algorithm for Graph Domain". 2010

XProj [Agg+07] is a scalable structural clustering algorithm for XML documents or, more general, labeled trees. It projects the clusters to sets of approximate frequent subtree patterns with a fixed size. The algorithm requires the enumeration of these size-restricted frequent subtree patterns for each cluster in each iteration. An adoption to general graphs is possible but would result in a large performance impact since many subproblems will become much more complex. Another XML-document clustering framework was presented by [PBM16]. The framework contains a structural variant PathXP, which is based on the enumeration of frequent path patterns. Again, the approach scales well to larger datasets but is restricted to very limited path patterns.

[Agg+07] Aggarwal et al., "Xproj: a framework for projected structural clustering of xml documents". 2007

[PBM16] Piernik, Brzezinski, and Morzy, "Clustering XML documents by patterns". 2016

Some hybrid clustering algorithms try to overcome the structural complexity by assisting the structural clustering with a vector-based representation. The aforementioned algorithm of Ferrer et al. [Fer+09] is one example. The probably most relevant competitor in terms of scalable structural clustering algorithms is a hybrid approach of Seeland, Karwath, and Kramer [SKK14], called SCAP. It applies a fingerprint-based pre-clustering, which partitions the data in a way that aligns with structural properties with a high probability. Afterwards, a structural overlapping clustering algorithm is run on each partition element, to refine the clustering. By this, the superlinear growth of the structural clustering algorithm only applies to the

[SKK14] Seeland, Karwath, and Kramer "Structural clustering of millions of molecular graphs". 2014

smaller partition elements. In addition, they give a shared-memory parallelization to utilize the resources of modern multiprocessor systems.

Both scalable algorithms in the above discussion, i.e., XProj and SCAP, depend on fixed-sized subgraph pattern commonalities. While XProj tries to compensate for this with large sets of partly overlapping commonalities, SCAP must be tried with different parameters to find a suitable size for each dataset.

3.2 StruClus

This section presents the structural StruClus clustering algorithm, which is tailored towards large-scale datasets of small labeled graphs. The design goal of the algorithm was to achieve high-quality clusterings of molecular libraries, although it is not limited to that use case. Besides the structural nature, StruClus is a partitioning flat clustering algorithm with explicit noise handling which adapts the number of clusters to the dataset structure. Furthermore, structural commonalities are not limited to a subclass of patterns and the size of commonalities is dynamically adjusted with the help of cluster homogeneity balancing.

> **Problem 3.1 (SPC).** Structural Partitioning Graph Set Clustering
>
> **Input:** *A graph dataset $\mathcal{G}$*
>
> **Task:** *Find a partition $\mathcal{C} = \{C_1, \ldots, C_c\}$ of $\mathcal{G}$, such that it maximizes cluster homogeneity, minimizes the number of clusters, and respects a minimum cluster separation.*

The problem definition leaves the definition of homogeneity (cf., eq. (3.4)) and separation (cf., eq. (3.13)) open at this point. To define them properly, the used concept of a structural cluster has to be defined first.

Algorithm 2: StruClus Overview

Input: graph dataset $\mathcal{G}$ of non-empty graphs
Output: clustering $\mathcal{C} = \{C_1, \ldots, C_c\}$, cluster representatives $\mathcal{R} = \{\mathcal{R}(C_1), \ldots, \mathcal{R}(C_c)\}$

```
1 apply pre-clustering;
2 while not convergent do
3     split clusters;
4     merge clusters;
5     update cluster representatives;
6     assign graphs to closest cluster representatives;
```

A high-level description of the StruClus algorithm is given in algorithm 2. Initially, StruClus applies a lightweight pre-clustering algorithm. Then, the intermediate clustering result is refined using an optimization loop similar to the k-Means algorithm.

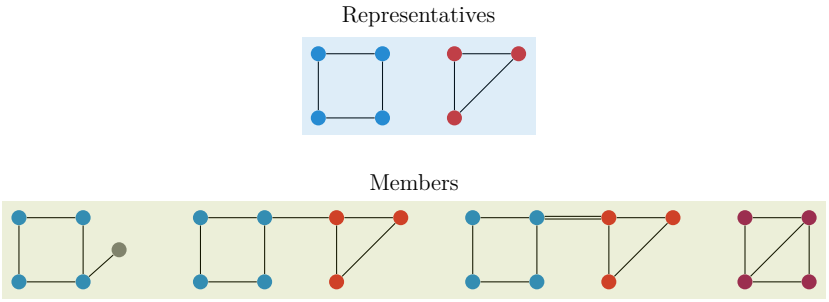


Figure 3.2: Example of a structural cluster with an attached representative set. Vertex colors indicate subgraph isomorphisms between the representatives and members. The purple color indicates that a subgraph isomorphism from the red and blue representatives is mapping to the same vertex.

In order to adapt to the number of clusters present in the dataset and to achieve a good cluster separation, a cluster splitting and merging procedure is applied in each iteration.

3.2.1 Structural Representatives to Express Commonalities in Clusters

As discussed in paragraph 2.6.3.1.3, structural clustering algorithms resolve around cluster commonalities described in the graph domain. The StruClus algorithm is attaching a representative set of structural graph patterns (cf., definition 2.13) to each cluster.

> **Definition 3.1 (Structural Cluster Representative Set).** *Given a cluster C, a structural cluster representative set is defined as a set $\mathcal{R}(C) := \{P_1, \ldots, P_k\} \subseteq C_{\sqsubseteq}$ of connected subgraph patterns, named representatives.*

In the following, the graphs of a cluster will be denoted by the term cluster members to distinguish them from the cluster representatives. With the exception of a single noise cluster (which will be discussed later in more detail), the following invariant holds.

$$\forall C \in \mathcal{C} : \forall G \in C : \exists P \in \mathcal{R}(C) : P \sqsubseteq G \tag{3.1}$$

In other words, the subgraph isomorphism relation is the representative link between the representative patterns and the cluster members. Additionally, the cardinalities of the representative sets are limited to a fixed maximum number $r_{\max}$ of representatives, i.e., $\forall C \in \mathcal{C} : |\mathcal{R}(C)| \leq r_{\max}$.

Figure 3.2 shows an example of a structural cluster with an attached representative set. The vertex colors indicate the mapped vertices of the subgraph isomorphism

relation. Thus, the blue representative is a common subgraph pattern of all cluster members, while the red representative is only subgraph isomorphic to three of the four. For each cluster member, there exists at least one representative that is subgraph isomorphic to it.

As shown in the example, a representative, that is subgraph isomorphic to more than a single member, is a common subgraph of these members and explicitly expresses a structural commonality among them. If the representative set would contain only a single pattern, eq. (3.1) would enforce the pattern to be a common subgraph of all members. Having a larger representative set, on the other hand, allows to represent more complex clusters, where cluster members are a composition of multiple frequent patterns (cf., section 2.6.2) and not every frequent pattern is subgraph isomorphic to each member (see the synthetic datasets of the evaluation in section 3.2.9 for an example). It can be concluded, that one quality criterion for a representative P is its support $\text{supp}_C(P)$ in the cluster C it is associated with. A higher support is preferable in terms of shared commonalities and thereby cluster homogeneity.

However, having an arbitrary common subgraph as a representative for a set of graphs is not a sufficient condition to express a high degree of structural commonalities among the graphs. To be meaningful, the size of the representative is also important. If the size of the representative is small in comparison with the graphs it represents, the shared structural commonality between the graphs is also small in relative terms. In this sense, the representative power of a representative $P \sqsubseteq G$ w.r.t. a single cluster member G will be defined as the relative number of covered edges by the subgraph isomorphism:

$$\text{rpow}(P, G) := \frac{|E(P)|}{|E(G)|} \tag{3.2}$$

For a uniform cluster C—that is, a cluster of isomorphic members—a representative pattern P can be found, that achieves optimal values in both criteria, i.e., $\text{supp}_C(P) = 100\%$ and $\forall G \in C : \text{rpow}(P, G) = 1$. However, if non-isomorphic graphs should be represented by a pattern, the maximum common subgraph is smaller than at least one of the graphs and $\text{rpow}(P, G)$ must be smaller than 1 for at least one graph G. Vice versa, if the minimum support $\text{supp}_{\min}$ for a representative is lowered, it is possible to leave more cluster members unrepresented by the representative and some previously infrequent patterns may become frequent. These are always superpatterns of the frequent patterns with higher support. Thus, there is a chance to increase the representative power for the represented graphs if some graphs are allowed to be unrepresented.

It was discussed above that multiple representatives are beneficial in cases, where a cluster is formed by multiple frequent patterns. However, the connections between different representatives were not addressed explicitly until now. Representative patterns can be related in different ways. First, they can be either structurally similar or dissimilar to each other. Second, they may share a large or small portion of the members they represent. If patterns are similar to each other or they share a large portion of supporting members, cluster members themselves are similar or share common features, respectively. Contrariwise, clusters with dissimilar patterns that

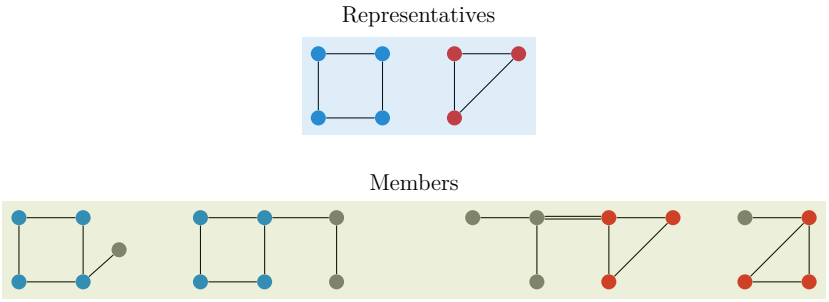


Figure 3.3: Example of a structural cluster that contains two separate subclusters. Vertex colors indicate subgraph isomorphisms between the representatives and members.

represent distinct parts of the cluster members are a problem in terms of member commonalities. Figure 3.3 shows a problematic example cluster C with two representatives P_1 and P_2 that have no overlap in their supporting members, i.e., $C_{\sqsupseteq P_1} \cap C_{\sqsupseteq P_2} = \emptyset$. Consequentially, C actually contains two separate subclusters.

Observation 3.1. *Let P_1 and P_2 be two representative patterns of a cluster C, with $\text{supp}_C(P_1) \geq \text{supp}_{\min}$ and $\text{supp}_C(P_2) \geq \text{supp}_{\min}$. Then, the following inequality holds.*

$$|C_{\sqsupseteq P_1} \cap C_{\sqsupseteq P_2}| \geq 2\,\text{supp}_{\min} - |C| \tag{3.3}$$

In other words, observation 3.1 states, that an overlap between the supporting members of a representative can be enforced by choosing a sufficiently high minimum support threshold and it is possible to avoid situations of separated subclusters in a single real cluster. Representative patterns for StruClus are calculated using a maximal frequent subgraph pattern mining algorithm (will be discussed in more detail below, cf., section 3.2.2). By this, it is possible to enforce a minimum support for each representative pattern.

Under the assumption, that the minimum support of representative patterns is close to 1, cluster homogeneity is then defined as follows:

$$\text{hom}(C) := \frac{\frac{1}{|\mathcal{R}(C)|}\sum_{P \in \mathcal{R}(C)} |E(P)|}{\frac{1}{|C|}\sum_{G \in C} |E(G)|} \tag{3.4}$$

If the minimum support $\text{supp}_{\min}$ is close to 1, we can omit the fact, that some representatives might not represent all cluster members. Thus, Equation (3.4) is simply the average representative power as defined in eq. (3.2) with a low error margin as a result of the relaxed subgraph isomorphism condition. This avoids subgraph isomorphism tests and eases cluster homogeneity comparisons since it is not needed to weigh the number of subgraph isomorphic representatives against the representative power, e.g., for homogeneity balancing (cf., section 3.2.2.3).

By using a structural cluster representative set of common subgraphs, cluster commonalities are now solely measured w.r.t. to patterns in this set. In other words, structural similarities or commonalities between cluster members that are not expressed by the representatives are not considered. Since the cardinality of the representative set is limited, not every common subgraph of the cluster members can be considered. In this sense, structural clustering algorithms with somehow limited representative sets are said to be structural projected clustering algorithms [cf., Agg+07]. This is similar to classical projected clustering algorithms for vector space, which also select a subset of relevant features of a cluster to measure intra-cluster similarities or homogeneity.

[**Agg+07**] Aggarwal et al., "Xproj: a framework for projected structural clustering of xml documents". 2007

3.2.2 Representative Update

As discussed above, a sufficiently high support of representative patterns is crucial in order to have homogeneous clusters and a sufficient overlap between the supported graphs of different representatives. For this reason, the initial calculation and update of cluster representatives in StruClus is based on maximal frequent patterns (cf., definition 2.16). For each cluster, a representative update will be performed by mining maximal frequent patterns in a first step (cf., section 3.2.2.1) and selecting the final representatives by a ranking function in a second step (cf., section 3.2.2.2). The reason to choose maximal frequent (and not simply frequent) patterns has two reasons. First, the usage of maximal frequent patterns in combination with the representative selection criterion (cf., section 3.2.2.2) ensures, that the representative set does not contain patterns that are subgraph isomorphic in one or the other direction. Thus, it helps to diversify the representative set and avoids situations in which the same commonality is selected multiple times just in different sizes. Second, using only maximal patterns can enforce an increase in homogeneity of a cluster, which will play an important role in balancing cluster homogeneity (cf., section 3.2.2.3).

3.2.2.1 Probabilistic Maximal Frequent Subgraph Pattern Sampling

Enumerating all maximal frequent patterns can result in an exponential number of representative candidates (cf., section 2.6.2). The enumeration and ranking of these patterns is usually computationally demanding, especially since this process has to be repeated for each cluster in each iteration of the StruClus algorithm. In order to scale to large datasets with low constant factors (cf., experimental evaluation in section 3.2.9), a twofold sampling method is used to calculate representative candidates in a lightweight fashion.

3.2.2.1.1 Sampling in Pattern Space To avoid the high number of representative candidates due to a complete maximal frequent subgraph enumeration, StruClus draws a random sample $\mathfrak{M}$ from the set of maximal frequent patterns $C_{\max\text{freq}}$ of a cluster C. Algorithmically, it can be achieved with the help of rank increasing random walks on the poset of frequent patterns. Figure 3.4 shows an example of such a random walk. To sample a single maximal frequent subgraph, the algorithm starts

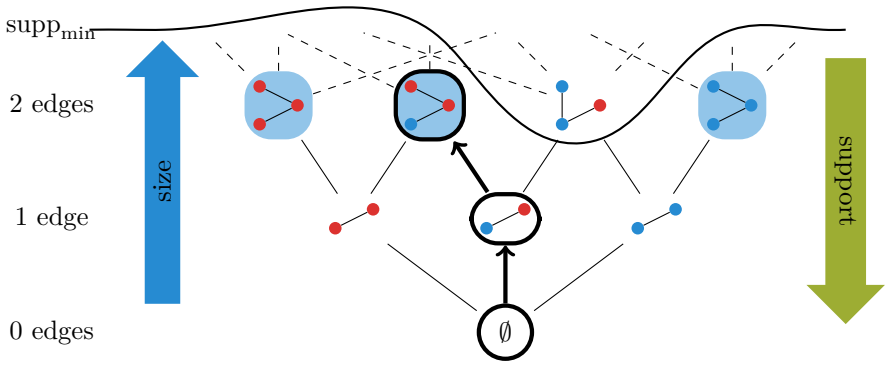


Figure 3.4: Example of a rank increasing random walk in the support pruned labeled connected pattern space with $|\mathcal{L}_V| = 2$, partially ordered by the subgraph isomorphism relation $\sqsubseteq$, and depicted as Hasse diagram. All graph patterns below the $\text{supp}_{\min}$ border are frequent; all above infrequent. Frequent graph patterns with blue background are maximal. Vertex colors indicate vertex labels.

with the enumeration of frequent edges and applies forward and backward extensions (cf., definitions 2.18 and 2.19) until a maximal frequent pattern is reached. Thus, the approach is quite similar to the pattern growth approach of frequent subgraph enumeration algorithms as described in paragraph 2.6.2.1.1. However, instead of branching into multiple possible pattern extensions, a single frequent extension (i.e., an extension that leads to a frequent pattern) is selected in a random fashion. This technique was originally introduced as an intermediate step of the ORIMGAMI representative pattern mining algorithm [Has+07].

[**Has+07**] Hasan et al., "ORIGAMI: Mining Representative Orthogonal Graph Patterns". 2007

A more precise description is given in algorithm 3. The algorithm starts with the enumeration of frequent edge patterns in line 2. This can be performed by scanning all edges in $\mathcal{G}$ to count the distinct vertex-edge-vertex label combinations. Then algorithm 3 constructs m patterns in the loop starting in line 3. Starting with an empty pattern, algorithm 3 enumerates all possible extensions for the current pattern in line 6 to draw a random extension from this set in line 9. Implementation-wise, the enumerated extensions can be stored in a lightweight fashion by referencing a node or node pair (in the case of backward extensions) of P and an associated frequent edge pattern from E_{freq}. The complete pattern can then be constructed in line 9. The determination of an extension as frequent can be performed lazily, i.e., a random extension is drawn and only applied if it is frequent. If the extension is not frequent, algorithm 3 loops in line 8 until a frequent extension is found or the set of enumerated extensions is depleted. In the former case, the first found frequent extension is applied in line 11. In the latter case, P is a maximal frequent pattern and the extension loop of line 5 exits to add the pattern to the output of the algorithm in line 13.

Algorithm 3: Sampling of Maximal Frequent Graphs Patterns

Input: graph dataset $\mathcal{G}$ of non-empty graphs, cardinality $m \in \mathbb{N}^{\geq 1}$ of maximal frequent patterns to sample, minimum support $\text{supp}_{\min}$
Output: random sample of maximal frequent patterns $\mathfrak{M}$

```
1  𝔐 ← ∅;
2  E_freq ← frequentEdges(𝒢);
3  while |𝔐| < m do in parallel
4      P ← ∅;
5      do
6          extensions ← enumExtensions(P, E_freq);
                                          ▶ cf., definitions 2.18 and 2.19
7          Q ← null;
8          while extensions ≠ ∅ and (Q = null or not
            isFrequent(Q, 𝒢, supp_min)) do
9              Q ← extensions.getAndRemoveRandom();
10         if Q ≠ null then
11             P ← Q;
12     while Q ≠ null;
13     𝔐 ← 𝔐 ∪ {P};
14 return 𝔐;
```

3.2.2.1.2 Sampling Cluster Members While the sampling of maximal frequent patterns greatly improves the performance in comparison with enumeration algorithms, the $|C|$ subgraph isomorphism tests for each extension—including failed infrequent extensions—remain a computational bottleneck for StruClus. For this reason, StruClus calculates the support of patterns on random samples of the members in addition to the pattern space sampling discussed above. This helps to reduce the number of subgraph isomorphism tests in order to determine whether a pattern is frequent or not. To limit the sampling-induced error probability, a probabilistic sampling strategy is presented in the following.

Lemma 3.2. *Given a graph dataset $\mathcal{G}$, a uniformly distributed random sample (with replacement) $\mathfrak{S} = (G_1, \ldots, G_N)$ of graphs from $\mathcal{G}$, and a pattern $P \in \mathcal{G}_{\sqsubseteq}$, the random variable $\text{supp}_{\mathfrak{S}}(P) = |\mathfrak{S}_{\sqsupseteq P}|$ follows the binomial distribution $B_{n, \frac{|\mathcal{G}_{\sqsupseteq P}|}{|\mathcal{G}|}}$.*

Proof. Given the sample space $\mathcal{G}$ and a fixed pattern $P \in \mathcal{G}_{\sqsubseteq}$, P is either subgraph isomorphic to a graph $G \in \mathcal{G}$ ($G \in \mathcal{G}_{\sqsupseteq P}$) or not ($G \in \mathcal{G}_{\not\sqsupseteq P} = \mathcal{G} \setminus \mathcal{G}_{\sqsupseteq P}$). When picking a graph G from $\mathcal{G}$ at random, the random variable $X : \mathcal{G} \to \{0, 1\}$ with the events $G \in \mathcal{G}_{\sqsupseteq P}$ (success, value 1) and $G \in \mathcal{G}_{\not\sqsupseteq P}$ (failure, value 0) has a success rate of $p := \frac{|\mathcal{G}_{\sqsupseteq P}|}{|\mathcal{G}|}$ and a failure rate of $\frac{|\mathcal{G}_{\not\sqsupseteq P}|}{|\mathcal{G}|} = \frac{|\mathcal{G}| - |\mathcal{G}_{\sqsupseteq P}|}{|\mathcal{G}|} = 1 - p$. Thus, X follows the Bernoulli distribution $B_{1,p}$. Let $\mathcal{S} := (x_1, \ldots x_N)$ be the random sample representing the observation of the random variables $X(G_1), \ldots X(G_N)$, i.e., N i.i.d. random variables following $B_{1,p}$. Then, $|\mathfrak{S}_{\triangleright P}| = \sum_{x \in \mathcal{S}} x$ is the number of successes and follows the Binomial distribution $B_{N,p}$. □

A similar observation was also made by Lin, Xiao, and Ghinita [LXG14], which used the binomial distribution to filter possible infrequent patterns for distributed frequent pattern mining. With the help of lemma 3.2 it is possible to rephrase the decision problem, to determine if a pattern is frequent or not, as a statistical hypothesis.

[**LXG14**] Lin, Xiao, and Ghinita "Large-scale frequent subgraph mining in MapReduce". 2014

Statistical Hypothesis Test Question 3.1 (Frequent Test).

Input:	*A random sample (with replacement) $\mathfrak{S} = (G_{11}, \ldots, G_{1N})$ of graphs from graph dataset $\mathcal{G}$, a minimum support threshold $\text{supp}_{\min}$, a pattern $P \in \mathcal{G}_{\sqsubseteq}$, and a significance level α.*
Hypothesis:	$H_0 : \text{supp}_{\mathfrak{S}}(P) = \text{supp}_{\min}$
Test Question:	*Can H_0 be rejected w.r.t. the observation $|\mathfrak{S}_{\sqsupseteq P}|$ and significance level α?*

Statistical Hypothesis Test Question 3.1 can be implemented using a binomial test (cf., section 2.3.5.1). While the test hypothesis H_0 is based on the rejection of equality, it is actually necessary to decide if the pattern is above or below the minimum support threshold. However, such a decision can be derived from the test on equality and the computational costs of a second tests can be avoided. Let's assume that it should be determined whether a pattern P is frequent in a cluster

C based on the support $\text{supp}_{\mathfrak{S}}(P)$ in some random sample $\mathfrak{S}$ of C. Then, it is possible to simply assume that $\text{supp}_C(P) < \text{supp}_{\min}$ iff $\text{supp}_{\mathfrak{S}}(P) < |\mathfrak{S}|\frac{\text{supp}_{\min}}{|C|}$ and $\text{supp}_C(P) \geq \text{supp}_{\min}$ iff $\text{supp}_{\mathfrak{S}}(P) \geq |\mathfrak{S}|\frac{\text{supp}_{\min}}{|C|}$ if the hypothesis can be rejected with some reasonable confidence level of $\alpha < 0.5$. This relation between the test on equality and the one-sided tests is justified by the test decision as given in 2.3.5.1 for which the test on equality incorporates both single-sided tests with the half confidence level. Thus, two single-sided tests corrected with the simple Bonferroni method (cf., section 2.3.4.1) would result in the same outcome.

With the above discussed statistical test, it is, therefore, possible to determine if a single pattern P is either frequent or not with a probability of $1 - \alpha$ under the assumption, that H_0 can be rejected with enough confidence. However, it is still necessary to handle situations in which H_0 cannot be rejected, i.e., situations in which the real support $\text{supp}_C(P)$ of P is very close to the minimum support threshold $\text{supp}_{\min}$. Since the cardinality $|\mathfrak{S}|$ of the sample of cluster members $\mathfrak{S}$ used for the candidate test influences the test power, it might be enough to increase the sample cardinality in order to reject H_0. However, choosing a very large sample for all tests contradicts the performance goal of the sampling approach. Since the real support $\text{supp}_C(P)$ of P in C is unknown, it is not possible to automatically select an appropriate sample cardinality according to the situation. For this reason, StruClus uses a sample size doubling strategy. Starting with a small first sample $\mathfrak{S}_1$, the frequency test is executed. If the rejection of the hypothesis fails, a second sample $\mathfrak{S}_2$ with $|\mathfrak{S}_2| = 2|\mathfrak{S}_1|$ is drawn. This process is repeated with $|\mathfrak{S}_i| = 2|\mathfrak{S}_{i-1}|$ until H_0 could be either rejected or the sample cardinality exceeds the cardinality of cluster C. In the latter case, an exact support $\text{supp}_C(P)$ is computed in order to determine if P is frequent or not. Consequentially, the sampling approach is not faster than the exact algorithm in the worst case.

Algorithm 4 shows the probabilistic determination of the classification as frequent or infrequent in more detail. It serves as a drop-in replacement of the function `isFrequent` in algorithm 3. It starts with a minimal sample cardinality in line 2 and doubles the sample cardinality (line 8) as long it is smaller than $|\mathcal{G}|$ and the hypothesis test in line 6 cannot be rejected. If the hypothesis is rejected, the probabilistic answer is directly returned in line 7. Otherwise, the exact decision is returned in line 9. The implementation of the algorithm avoids duplicate subgraph isomorphism tests. In order to execute the hypothesis test, $\text{supp}_{\mathfrak{S}}(P)$ is calculated using subgraph isomorphism tests against $\mathfrak{S}$ in line 5. If multiple tests are necessary, a new independent sample has to be drawn for each test in line 4. Let's assume, that $\mathfrak{S}_A$ and $\mathfrak{S}_B$ are two independent samples of two different iterations. Then it can happen, that $\mathfrak{S}_A \cap \mathfrak{S}_B = I \neq \emptyset$ holds. In order to calculate the support $P \sqsubseteq G$ for some $G \in I$, the second subgraph isomorphism test can be avoided by caching the result of the first one. As a result, algorithm 4 does only execute $|\mathcal{G}|$ subgraph isomorphism tests in the worst case.

The statistical test is repeated for each extension and each sample cardinality doubling. As a consequence, a multiple hypothesis testing correction is necessary to bound the family-wise error for each P to be a maximal frequent pattern of $\mathcal{G}$.

Algorithm 4: Probabilistic Frequency Decision

Input: graph dataset $\mathcal{G}$ of non-empty graphs, pattern P, minimum support $\text{supp}_{\min}$
Fixed Parameterization: minimum dataset sample cardinality $s_{\min}$, maximum error probability α
Output: classification of P into frequent (true) or infrequent (false)
Subprocedure Of: algorithm 3

```
1 procedure isFrequent(P, 𝒢, supp_min)
2     s ← s_min;
3     while s < |𝒢| do
4         𝔖 ← randomSample(𝒢, s);
5         support ← supp_𝔖(P);
6         if binomialTestOnEquality(support, s·supp_min/|𝒢|, α) = rejected then
7             return support ≥ s·supp_min/|𝒢|;
8         s ← 2s;
9     return supp_𝒢(P) ≥ supp_min;
```

Lemma 3.3. *Let $\mathcal{G}$ be a graph dataset, $s_{\min}$ a minimum sample cardinality, E_{freq} the set of all frequent edge patterns in $\mathcal{G}$, $\text{supp}_{\min}$ a minimum support threshold, and $v_{\max}$ the $(1 - \frac{\text{supp}_{\min}}{\mathcal{G}})$-quantile of the sorted (increasing order) vertex counts, i.e., $|V(G)|$ for each $G \in \mathcal{G}$. Then, the maximal number of binomial tests to construct a maximal frequent substructure over $\mathcal{G}$ using algorithms 3 and 4 is bounded by:*

$$\left\lceil \log \frac{|\mathcal{G}|}{s_{\min}} \right\rceil \left(\binom{v_{\max}}{2} + v_{\max} \right) |E_{\text{freq}}| \tag{3.5}$$

Proof. The sample cardinality is doubled at maximum $\left\lceil \log \frac{|\mathcal{G}|}{s_{\min}} \right\rceil$ times if the test never reaches the desired significance level. The vertex count $|V(P)|$ of some $P \in \mathcal{G}_{\max \text{freq}}$ is bounded by the vertex count of each supporting graph $G \in \mathcal{G}_{\sqsupseteq P}$. In the worst case P is supported by the $\text{supp}_{\min}$-largest graphs of $\mathcal{G}$. The vertex count of the smallest supporting graph is then equal to the $(1 - \frac{\text{supp}_{\min}}{\mathcal{G}})$-quantile of the sorted vertex counts in increasing order. The number of applied frequent extensions is bound by the number of edges in the complete graph with $v_{\max}$ vertices times the number of available edge patterns $|E_{\text{freq}}|$. To decide whether P is maximal, a maximum number of $|E_{\text{freq}}|$ additional (infrequent) forward extensions for each vertex need to be performed. □

With the help of lemma 3.3 it is possible to apply a simple Bonferroni correction (cf., section 2.3.4.1) to the significance level of algorithm 4.

Corollary 3.4. *Each subgraph pattern $P \in \mathfrak{M}$ of algorithm 3 in combination with algorithm 4 is a maximal frequent subgraph pattern with a probability of at least $1 - \alpha$ if a Bonferroni correction is applied to the significance level of algorithm 4 according to lemma 3.3.*

Proof. Follows directly from lemma 3.3. □

3.2.2.2 Representative Selection

In the representative update step and during the pre-clustering phase of algorithm 2, representatives have to be calculated for each cluster $C \in \mathcal{C}_i$ of the current iteration i. With the help of algorithm 3, StruClus mines a set of representative candidates $\mathfrak{M}$ with a cardinality of m. Afterwards, the $r_{\max}$ best-ranked candidates are selected as cluster representatives. With the sampling approach, it is possible, that highly ranked previously found commonalities of the clusters are missed by chance, i.e., are not contained in $\mathfrak{M}$. To avoid such situations the selected representatives $\mathcal{R}(C_{-1})$ of the previous iteration $i-1$ are added to the candidate set, i.e., the ranking is performed on the set $\mathfrak{M} \cup \mathcal{R}(C_{-1})$. The ranking function is defined over a representative candidate pattern P, a cluster C, and the complete dataset $\mathcal{G}$ (i.e., the input of algorithm 2).

$$\text{rank}(P, C, \mathcal{G}) := \frac{|C_{\sqsupseteq P}||E(P)|}{\sum_{G \in C_{\sqsupseteq P}} |E(G)|} \left(\frac{\text{supp}_C(P)}{|C|} - \frac{\text{supp}_{\mathcal{G}}(P)}{|\mathcal{G}|} \right) \tag{3.6}$$

As already discussed in section 3.2.1, the goal of StruClus is to maximize cluster homogeneity w.r.t. eq. (3.4). For this reason, the first part of eq. (3.6) does reflect the average representative power of the supporting graphs $C_{\sqsupseteq P}$ of P in C.

Along with the representative power, it is necessary to have a sufficiently high support in order to actually share commonalities among the members and force an overlap between the supporting graphs of the representatives (cf., observation 3.1). This second criterion will be enforced by the $\text{supp}_{\min}$ parameter of algorithm 3. The ultimate value of $\text{supp}_{\min}$ will be selected by the cluster homogeneity balancing (cf., section 3.2.2.3). Nevertheless, it is still possible, that the support values of patterns differ to some degree, as the support of a maximal frequent graph pattern must not be exactly the $\text{supp}_{\min}$ threshold. Additionally, the representative patterns of the previous generation are ranked alongside. They may have been mined with a different minimum support threshold and their support may change with changing cluster members in succeeding iterations. Higher support values are always preferable in terms of shared commonalities, which is the reason why the representative power is complemented with the support of the pattern. Extremely unbalanced cases in which one criterion has a high and the other criterion has a very low value should be avoided, since both criteria are needed to express homogeneous clusters. Thus, the ranking function uses the product of the two criteria.

Finally, the ranking function also guides the representative selection process to prefer patterns that are special to the cluster by subtracting the dataset-wide support

of the pattern from the cluster-specific support. This avoids the development of clusters towards indiscriminate representatives and avoids unnecessary cluster merging operations (cf., section 3.2.4.2), which will be introduced later to guarantee a minimum separation constraint.

3.2.2.3 Homogeneity Balancing

A sufficiently high homogeneity (cf., eq. (3.4)) of a cluster is necessary to produce meaningful clustering results. However, it is not possible to a priori define the desired homogeneity level, since the achievable homogeneity depends on the precise dataset composition. In the one extreme, a dataset is formed of large groups of isomorphic graphs. As such, uniform clusters of isomorphic graphs could be built with an optimal homogeneity of one and a pattern support of 100% (cf., section 3.2.1). On the other extreme are datasets where pairwise graphs only share very small common subgraphs. The size of these common subgraphs then limits the homogeneity criterion too much lower values. As a consequence, it is only possible to push the homogeneity of clusters as far as the dataset allows.

The minimum support threshold $\text{supp}_{\min}$ for representative candidate mining plays an important role in this process. As already discussed in section 3.2.1, a low minimum support leads to a larger maximal frequent pattern in comparison with a higher support as a consequence of the anti-monotonicity property (cf., lemma 2.2). Thus, on the one hand, choosing a relatively low value for $\text{supp}_{\min}$ will increase the homogeneity of a cluster as the size of the representatives in comparison with the size of the members increase. On the other hand, if such a low-support pattern is selected as representative (cf., section 3.2.2.2) this might lead to unrepresented cluster members. These unrepresented members need to be assigned to another cluster since the invariant of eq. (3.1) does no longer hold. In summary, a low minimum support for representative candidate generation will lead to an increase in the cluster's homogeneity and a decrease in the cluster's size. On the contrary, a higher support will slow down this process of decreasing the cluster's size and a value of one will halt it completely.

The process of sorting out graphs needs to be stopped at some point to have clusters of meaningful size. This is guaranteed by the convergence criterion (cf., section 3.2.6). However, some clusters might converge very fast to a good homogeneity and others need more iterations. The usage of a fixed minimum support threshold thereby may lead to a situation where some clusters are relatively inhomogeneous while others are already highly homogeneous. If the convergence criterion stops the algorithm at this point, these inhomogeneous clusters are not meaningful. Conversely, if the algorithm continues the process, homogeneous clusters unnecessarily decrease their size until convergence is reached. As a consequence StruClus chooses the minimum support cluster-specific to balance cluster homogeneity over all clusters. In a first step, the relative homogeneity w.r.t. eq. (3.4) of each cluster C of a clustering $\mathcal{C}$ is calculated.

$$\text{relHom}(C, \mathcal{C}) := \frac{\hom(C)}{\frac{1}{|\mathcal{C}|}\sum_{C' \in \mathcal{C}} \hom(C')} \tag{3.7}$$

Equation (3.7) describes the relative homogeneity of a cluster C and returns a value below 1 if the cluster has a lower homogeneity than the average and a value above 1 in the opposite case. This value is then linearly mapped to a $\text{supp}_{\min}$ value for representative mining.

$$\text{supp}_{\min}(C, \mathcal{C}) := |C| \begin{cases} \text{ls} & \text{if relH}(C, \mathcal{C}) < \text{lr}, \\ \text{hs} & \text{if relH}(C, \mathcal{C}) > \text{hr}, \\ \text{relH}(C, \mathcal{C}) \frac{\text{hs}-\text{ls}}{\text{hr}-\text{lr}} + \left(ls - lr \frac{\text{hs}-\text{ls}}{\text{hr}-\text{lr}} \right) & \text{otherwise.} \end{cases} \tag{3.8}$$

The variables ls, hs, lr, hr represent low and high values for the relative (i.e., fractional) minimum support and relative homogeneity. Thus, the linear function goes through the points $(|C|\text{ls}, \text{lr})$ and $(|C|\text{hs}, \text{hr})$. Below and above these points, the $\text{supp}_{\min}$ values are constantly bound by ls and hs. This helps to ensure the necessity to have a minimum support well above 50% w.r.t. observation 4.12, i.e., $\text{ls} \gg 50\%$. On the other end, the minimum support cannot be above 100% by definition. Additionally, even highly homogeneous clusters should be improved in homogeneity if the loss of very few graphs leads to a significant increase in homogeneity. Thus, hs should be slightly below 100% in practice. In order to not stop the increase of homogeneity for a low homogeneous, but highly balanced intermediate clustering result, it is furthermore necessary to choose hr above one. The convergence criterion (cf., section 3.2.6) will then decide when to stop the process of homogeneity increase.

3.2.3 Cluster Assignment

During cluster assignment, each dataset graph $G \in \mathcal{G}$ is assigned to the cluster with best-fitting representatives, i.e., representatives that maximize the homogeneity as defined in eq. (3.4). Since the size of G is fixed, this boils down to choosing the subgraph isomorphic representatives with maximum size in terms of edges. Besides the representative power, the homogeneity objective does assume a high support for each representative. While it is totally fine to have a few graphs in a cluster that are only represented by a single or few representatives, it is preferable to have as many matching representatives as possible. For this reason, StruClus uses the following assignment similarity for a graph G and a cluster C. To emphasize patterns with large representative power over multiple small patterns, the edge counts of the patterns are squared.

$$\text{aSim}(G, C) := \sum_{\substack{P \in \mathcal{R}(C) \\ P \sqsubseteq G}} |E(P)|^2 \tag{3.9}$$

With the help of this similarity, it is possible to choose the most similar cluster via ranking.

Graphs that do not match any representative of any cluster—i.e., graphs with a similarity of zero to all clusters—are considered noise. StruClus creates a single

noise cluster in each iteration and assigns these graphs to it. Note, that this noise cluster will be considered a regular cluster in the next iteration and only a single noise cluster exists after each iteration. Thus, the invariant of eq. (3.1) is violated at most for a single noise cluster.

3.2.4 Cluster Splitting and Merging

As discussed in the introduction StruClus adapts the number of clusters to the dataset structure, i.e., it does not use a fixed number of clusters, such as k-Means. Up to this point, noise clusters are the only way to add new clusters to the intermediate clustering results. The cluster splitting and merging steps are another way to adapt to the dataset structure and maintain a well-separated clustering result.

3.2.4.1 Cluster Splitting

It was discussed in section 3.2.2.3, that StruClus increases cluster homogeneity by sorting out some unfitting graphs, i.e., choosing a minimum support below 100% during candidate generation. However, since $\text{supp}_{\min}$ is limited from below by $|C|$ls (cf., section 3.2.2.3), it might be impossible to find any representative that increases homogeneity, i.e., the chosen representatives are already the largest frequent pattern for $\text{supp}_{\min} = |C|$ls. For very inhomogeneous clusters it might therefore be impossible to increase the homogeneity to an average level. Additionally, convergence might be very slow if the the required number of additional clusters is high and only a single noise cluster is added in each iteration. A cluster splitting step is used for this reason. It collects all graphs from all clusters with a relative homogeneity (cf., eq. (3.7)) value below an a priori specified threshold $\text{relHom}_{\min}$.

$$\text{splitGraphs} := \{G \in C \in \mathcal{C} \mid \text{relHom}(C, \mathcal{C}) < \text{relHom}_{\min}\} \tag{3.10}$$

Then thees graphs (cf., eq. (3.10)) are re-clustered by the pre-clustering algorithm (cf., section 3.2.5). Thus, it can be seen as separate instance of StruClus with only a single iteration on the subset splitGraphs $\subseteq \mathcal{G}$. By mixing the members of several inhomogeneous clusters it is possible to find inter-cluster commonalities, that would have been infrequent in each of the previous clusters on their own. The resulting clusters are then re-added to the main instance.

3.2.4.2 Cluster Merging

On the contrary to cluster splitting, which focuses on cluster homogeneity, cluster merging ensures a minimum separation between clusters. Although the pre-clustering ensures well-separated initial clusters (cf., section 3.2.5) two clusters may develop towards each other or newly formed clusters (either noise clusters or re-added clusters from the cluster splitting step) maybe similar to existing ones.

In the context of projected clustering algorithms, the separation between clusters of different subspaces is often not considered, since the projected nature of clusters

causes similarities to be cluster-specific. As such, separation constraints that rely on intra-cluster comparisons of cluster members must resort to some set of features or dimensions that do not align with the clusters' specific subspaces. This contradicts the original idea of subspace or projected clustering algorithms, which states, that well-separated clusters may only exist in subspaces of a high dimensional superspace. However, structural projected clustering algorithms as described in section 3.2.1 have a special property. On the one hand, vectorial representations have numerical values for each dimension in the dataset. Thus, two given objects may be close or far away when they are projected to this dimension w.r.t. to the numerical value. On the other hand, structural commonalities are encoded by the presence or absence of structural features. Thus, one can encode such structural features in binary feature vectors of infinite dimensionality (if the size of patterns is not limited), where each representational dimension encodes a structural subgraph pattern. Given this viewpoint, StruClus does only project a cluster to a subgraph pattern feature, if the feature is actually present in the cluster. Consequentially, different clusters with the same representatives also share commonalities w.r.t. the cluster members. This is fundamentally different to classical vector space projected algorithms, where well-separated clusters within the same subspace may exist. Nevertheless, the above described situation of identical structural features is very strict and would not occur often in practice. For this reason, StruClus extends this concept to similar cluster representatives. Recalling the binary feature vector view on structural features, structural features are not independent. As a result of the monotonicity property (cf., lemma 2.2), subgraph patterns of a pattern are always present in the binary feature vector if the pattern itself is present.

Observation 3.5. *Let $C_1 \in \mathcal{C}$ and $C_2 \in \mathcal{C}$ be two clusters of a common clustering $\mathcal{C}$, $P_1 \in \mathcal{R}(C_1)$ and $P_2 \in \mathcal{R}(C_2)$ two representative patterns of these clusters, and Q with $Q \sqsubseteq P_1 \wedge Q \sqsubseteq P_2$ be a common subgraph pattern of P_1 and P_2. Then, Q is a common subgraph pattern of at least all represented graphs of P_1 and P_2, i.e., $C_{1_{\sqsupseteq P_1}} \subseteq C_{1_{\sqsupseteq Q}}$ and $C_{2_{\sqsupseteq P_2}} \subseteq C_{2_{\sqsupseteq Q}}$.*

Observation 3.5 implies, that the support of Q in $C_1 \cup C_2$ is bounded from below given that the support of P_1 and P_2 is bounded in their respective clusters. More precisely it is bound by the weighted average and the following inequality holds.

$$\mathrm{supp}_{C_1 \cup C_2}(Q) \geq \frac{|C_1| \,\mathrm{supp}_{C_1}(P_1) + |C_2| \,\mathrm{supp}_{C_2}(P_2)}{|C_1| + |C_2|} \tag{3.11}$$

Thus, if some common relative minimum support threshold was used to mine P_1 and P_2, Q is a frequent subgraph of the merged cluster. Additionally, if such a common subgraph pattern Q is relatively large in comparison with P_1 and P_2, the representative power of $\mathrm{rpow}(Q, G)$ for some graph $G \in C_{1_{\sqsupseteq P_1}} \cup C_{2_{\sqsupseteq P_2}}$ is also close to the representative power of P_1 or P_2. In such cases, the merging of the two clusters C_1 and C_2 would result in a homogeneous supercluster $C_1 \cup C_2$ with representative Q, assuming $r_{\max} = 1$. From the reverse viewpoint, C_1 and C_2 would not be well-separated in such a situation.

In order to judge the separation of two clusters, it is, therefore, crucial to determine if a relatively large common subgraph pattern of the representatives exists. The following similarity is defined to make this judgment for two representative patterns P_1 and P_2, where $\text{mcs}(P_1, P_2)$ returns an arbitrary common subgraph pattern with maximum edge count.

$$\text{rSim}(P_1, P_2) := \frac{|E(\text{mcs}(P_1, P_2))|}{\max\{|E(P_1)|, |E(P_2)|\}} \tag{3.12}$$

In other words, eq. (3.12) is the maximal factor of loss in representative power $\text{rpow}(Q, G)$ for some graph $G \in C_{1_{\sqsupseteq P_1}} \cup C_{2_{\sqsupseteq P_2}}$ and some common subgraph pattern Q of P_1 and P_2 w.r.t. $\text{rpow}(P_1, G)$ and $\text{rpow}(P_2, G)$. For the sake of completeness, it should be mentioned that $1 - \text{rSim}$ is a well-known distance introduced by Bunke and Shearer [BS98] (cf., section 2.6.3.4).

[BS98] Bunke and Shearer "A graph distance metric based on the maximal common subgraph". 1998

Finally, two clusters C_1 and C_2 are merged, whenever the following minimum separation constraint is violated.

$$|\{(P_1, P_2) \in \mathcal{R}(C_1) \times \mathcal{R}(C_2) \mid \text{rSim}(P_1, P_2) \geq \text{rSim}_{\max}\}| < \frac{|\mathcal{R}(C_1)| + |\mathcal{R}(C_2|}{2} \tag{3.13}$$

For example, if there exists a bipartite matching of similar representatives of two clusters, they would be merged. The representatives of the merged cluster are then chosen in a regular representative update (cf., section 3.2.2) with the addition of the pairwise maximum common subgraphs of the representatives $P_1 \in C_1$ and $P_2 \in C_2$ to the candidate set for representative ranking. The regular update is beneficial since there might exist better representatives as the maximum common subgraph patterns.

3.2.5 Pre-Clustering

The refinement of intermediate results in StruClus is based on the idea of increasing the representative power of representative patterns. To start with this process, it is necessary to have an initial dataset partitioning (i.e., clustering). Choosing such a partitioning at random would have the drawback, that each cluster would be drawn from the same distribution and as a result would have highly similar commonalities. Thus, clusters are expected to be not well-separated. Consequentially, this would result in a high number of clusters to be merged and a very slow convergence of the algorithm. Moreover, the commonalities would most likely have low representative power.

For these reasons, StruClus applies a lightweight pre-clustering to find a better starting point. To pre-cluster a dataset $\mathcal{G}$, a set of maximal frequent subgraphs $\mathfrak{M}$ is sampled from $\mathcal{G}_{\max\text{freq}}$ with the help of algorithm 3 as described in section 3.2.2.1. These maximal frequent graph patterns will serve as candidates for the representatives of the initial clusters. However, $\mathfrak{M}$ may contain very similar representatives. If such similar representatives are used to construct different clusters, the above problem of badly-separated initial clusters would still exist. In order to increase the diversity

[**Cha+08**] Chaoji et al. "ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns". 2008

of the initial representatives, the concept of α-orthogonal graph sets is used. This concept was originally introduced by Chaoji et al. [Cha+08]. Given a set of graph pattern A (in our case $\mathfrak{M}$) and a similarity sim between them, an α-orthogonal subset $B \subseteq A$ of these patterns is defined to be a subset for which the following conditions hold:

$$\forall (P_1, P_2) \in B \times B : \mathrm{sim}(P_1, P_2) \leq \alpha \tag{3.14}$$

$$\forall P_1 \in A \setminus B, \exists P_2 \in B : \mathrm{sim}(P_1, P_2) > \alpha \tag{3.15}$$

In other words, α-orthogonal subsets of A are inclusion maximal sets of pairwise dissimilar patterns. The pre-clustering repeatedly constructs such inclusion maximal sets in random order of $\mathfrak{M}$. The largest α-orthogonal subset of $\mathfrak{M}$ is then selected to construct the initial clusters with a single representative for each cluster. For the sake of well-separated clusters, StruClus reuses rSim of eq. (3.12) and sets α to $\mathrm{rSim}_{\max}$. To assign the graphs of $\mathcal{G}$ to the clusters, the regular assignment similarity aSim (cf., eq. (3.9)) does have the problem, that some graphs may not be represented, i.e., supported, by any pattern in the selected α-orthogonal subset. This is an undesired property for the pre-clustering since this may result in a large noise cluster with the same problems that the pre-clustering should actually avoid. For this reason a softer distance, that does not drop to 0 in case a non-existing subgraph isomorphism is desired. The similarity rSim would make sense in this context. However, the computational complexity of the maximum common subgraph isomorphism is not desirable for a lightweight pre-clustering phase. For this reason, the Jaccard-Similarity is used, utilizing numerical graphlet fingerprints.

3.2.6 Convergence

As stated in problem 3.1 (SPC), the objective of StruClus is to maximize homogeneity and minimize the number of clusters. Furthermore, it was discussed in section 3.2.2.3, that the shrinking of clusters can help to increase cluster homogeneity. Increasing the number of clusters therefore may help to increase overall homogeneity. In the most extreme case, a clustering of singleton clusters would achieve perfect homogeneity, if we dismiss the minimum separation constraint. The other extreme would be a clustering of a single cluster with very low homogeneity. Both extreme cases are undesired since they do not give any insight into the dataset structure. On the contrary, a good clustering would explain or summarize a large portion of the data with high homogeneity and with a simple model, i.e., relatively few clusters. Similar to the idea of the minimum description length principle, StruClus, therefore, balances the model complexity and the explained fraction of the data, i.e., homogeneity, in its convergence criterion.

$$z(\mathcal{C}) := \frac{\sum_{C \in \mathcal{C}} |C| \, \hom(C)}{|\mathcal{C}|} \tag{3.16}$$

The cluster splitting step sometimes introduces new clusters that are then re-merged with existing ones. As a consequence, the convergence criterion fluctuates and contains short-lived optima. The convergence criterion is therefore smoothed over some iterations. Let w be an averaging width and $\mathcal{C}_i$ the intermediate clustering result after the i-th iteration. Then, StruClus will terminate after the first iteration c for which the following conditions hold.

$$c \geq 2w \wedge \frac{\sum_{i=c-w+1}^{c} z(\mathcal{C}_i)}{\sum_{i=c-2w+1}^{c-w} z(\mathcal{C}_i)} \leq 1 + \epsilon \tag{3.17}$$

3.2.7 Computational Complexity

The subgraphs isomorphism and the maximum common subgraphs problem are NP-complete. As such, an exponential worst-case complexity w.r.t. the size of graphs in the dataset $\mathcal{G}$ can be expected as long $NP \neq P$. Nevertheless, these problems can be solved sufficiently fast for small graphs such as molecular structures for drug discovery, which is exactly the setting StruClus was designed for. During the following analysis, the size of a graph is considered to be bound by a constant maximum number of vertices $\mathcal{V}_{\max}$. Let $\mathcal{C}_{\max}$ be the intermediate clustering result with maximum cardinality, i.e., $|\mathcal{C}_{\max}| \geq |\mathcal{C}_i|$ for $1 \leq i \leq c$, where c is the iteration of termination (cf., eq. (3.17)).

Lemma 3.6. *Given a graph dataset $\mathcal{G}$ with $\forall G \in \mathcal{G} : |V(G)| \leq \mathcal{V}_{\max}$, the running time of StruClus is in $\mathcal{O}(c\ |\mathcal{C}_{\max}||\mathcal{G}|)$.*

Proof. It was shown in lemma 3.3, that the number of frequent and infrequent extensions is bound by $\left(\binom{v_{\max}}{2} + v_{\max}\right)\ |E_{\text{freq}}|$ in order to mine a single representative for a cluster C and $v_{\max} \leq \mathcal{V}_{\max}$ being the $(1 - \frac{\text{supp}_{\min}}{|C|})$-quantile of the ordered vertex counts in C. Given a constant maximum vertex count $\mathcal{V}_{\max}$, the term $\left(\binom{v_{\max}}{2} + v_{\max}\right)$ is constant, too. For a minimum support threshold that is bounded by ls (cf., section 3.2.2.3), at least $\lceil \text{ls}|C| \rceil$ edges must be isomorphic to each other in order to be frequent. For a maximum total number of $\binom{\mathcal{V}_{\max}}{2}|C|$ edges in C, at most $\frac{\binom{\mathcal{V}_{\max}}{2}|C|}{\text{ls}|C|} = \frac{\binom{\mathcal{V}_{\max}}{2}}{\text{ls}}$ frequent edge patterns can exist. Thereby, the number of frequent edge patterns $|E_{\text{freq}}|$ is also bounded by $\mathcal{V}_{\max}$. For each of these extensions, the number of subgraph isomorphism tests is in $\mathcal{O}(C)$. Thus, the running time to mine single maximal frequent subgraph pattern is in $\mathcal{O}(C)$ as well.

In order to update the representatives of a cluster, a constant number of maximal frequent subgraph patterns is mined as candidates. To rank a pattern P, the cluster-specific support $\text{supp}_C(P)$, the dataset-wide support $\text{supp}_{\mathcal{G}}(P)$, the set of supporting graphs in the cluster $C_{\sqsupseteq P}$, and the summed edge counts of graphs in $C_{\sqsupseteq P}$ have to be computed. All these values can be calculated in a single pass over $\mathcal{G}$ with constant time operations for each graph $G \in \mathcal{G}$. Since the number of representatives of a cluster is bound by $r_{\max}$, the calculation of the homogeneity values for cluster homogeneity balancing via mapping to a minimum support threshold (cf., section 3.2.2.3) is in $\mathcal{O}(|\mathcal{G}|)$.

The overall complexity of the representative update step for all clusters—including the calculation of the minimum support and the mining, ranking, and selection of representative candidates—is therefore in $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$.

In the assignment step, each dataset graph must be compared with each representative of all clusters w.r.t. the subgraph isomorphism relation. Since the number of representatives for each cluster is again a constant, the complexity of the assignment step is in $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$.

The pre-clustering involves the calculation of a constant number of maximal frequent subgraph patterns as described for the representative update step in $\mathcal{O}(|\mathcal{G}|)$ time. Then a constant number of α-orthogonal subsets of these candidate patterns is constructed. During the greedy construction of these sets, each pattern must be compared to all previously added patterns by the means of a maximum common subgraph isomorphism calculation. Since the candidate sets have a constant size, the number of α-orthogonal set constructions is a constant, and the maximum common subgraph calculations are applied to graphs with a bound vertex count, the combined running time to construct the α-orthogonal sets is also a constant. The dominating running time complexity during pre-clustering is the final assignment phase with a complexity of $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$.

For cluster splitting, the relHom values must be calculated for each cluster as described in the running time analysis of the representative update. Afterwards, the pre-clustering is applied to the graphs, i.e., some multiset $\mathcal{X} \subseteq \mathcal{G}$, of the selected clusters. Both running times were already discussed above. Thus, the overall running time is in $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$.

Cluster merging involves the pairwise comparison over all representative sets, which are bounded in their size. Furthermore, the representatives need to be updated for each merged cluster. The running time of the former step is thereby in $\mathcal{O}(\mathcal{C}_{\max}^2)$ and the representative update complexity is as described above. Since $\mathcal{O}(\mathcal{C}_{\max}^2) \subseteq \mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$ holds, the overall running time of the merging step is in $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$.

The computation of the convergence criterion is again using the well-discussed average homogeneity value, i.e., it is in $\mathcal{O}(\mathcal{G})$.

In summary, every single step of StruClus has a complexity of $\mathcal{O}(|\mathcal{C}_{\max}||\mathcal{G}|)$ and the overall running time is in $\mathcal{O}(c\ |\mathcal{C}_{\max}||\mathcal{G}|)$ for c iterations. □

3.2.8 Implementation Details

3.2.8.1 Subgraph Isomorphism Implementation

[KKM11] Klein, Kriege, and Mutzel, "CT-Index: Fingerprint-based Graph Indexing Combining Cycles and Trees". 2011

Connected subgraph isomorphism tests are performed using a backtracking algorithm described in [KKM11]. The CT-Index fingerprint pre-filtering—stemming from the same publication—is applied, such that the backtracking algorithm is only executed whenever the fingerprint test fails to eliminate the possibility of a subgraph isomorphism. These structural fingerprints are configured to have a length of 2048 bits and contain the presence of subtrees with up to five and rings with eight or less vertices.

3.2.8.2 Maximum Common Subgraph Implementation

A modified variant of the maximum common connected induced subgraph isomorphism algorithm presented by Kriege and Mutzel [KM12] is used in the implementations of this thesis. It computes a product graph and applies a backtracking maximum clique algorithm on top of it. This backtracking algorithm extends cliques by adding further vertices until the clique property is violated. The implementation in this thesis differs from [KM12] in the way the search is pruned. Since any coloring of a graph is an upper bound for the maximum clique size, a greedy coloring can be used to give an upper bound for the number of non-clique vertices that can be added without violating the clique property in the backtracking search. Whenever the current clique size plus the number of additional greedily added colors is below the current maximum clique size, the backtracking search is pruned.

[KM12] Kriege and Mutzel "Subgraph Matching Kernels for Attributed Graphs". 2012

3.2.8.3 Parallelization

The implementation of StruClus is shared-memory parallelized. Parallelization is implemented on the following levels. The mining of maximal frequent patterns is parallelized, such that each random walk is considered a unit of work (cf., line 3 in algorithm 3). The mining is used during the pre-clustering step, which also parallelizes similarity calculations during the construction of the α-orthogonal sets. Furthermore, it is used in the representative update step, which additionally parallelizes the ranking of each representative pattern candidate. The homogeneity calculation for cluster splitting, convergence, and homogeneity balancing is parallelized on the cluster level. Cluster merging is parallelized on the pairwise cluster comparison level. Furthermore, each distance computation during cluster assignment (i.e., the subgraph isomorphism tests of each representative pattern w.r.t. each dataset graph) is done in parallel.

3.2.9 Experimental Evaluation

The following evaluation can be divided into two major parts. Parametrization experiments are presented in the first part to understand the general behavior of the algorithm. In the second part, StruClus is compared to other clustering algorithms in terms of running time and quality, namely SCAP [SKK14], PROCLUS [Agg+99], and Kernel K-Means [Gir02] (cf., sections 2.6.3 and 3.1).

[SKK14] Seeland, Karwath, and Kramer, "Structural clustering of millions of molecular graphs". 2014

[Agg+99] Aggarwal et al., "Fast Algorithms for Projected Clustering". 1999

[Gir02] Girolami, "Mercer kernel-based clustering in feature space". 2002

3.2.9.1 Hardware, Software, and Test Setup

The evaluation was performed on a dual-socket NUMA system with two Intel Xeon E5-2640v3 processors with 8 cores (+ 8 virtual Hyperthreading cores) and 128GiB RAM (i.e., 64GiB per NUMA domain). Turbo Boost was disabled for stable and comparable running times over different runs. Each experiment was fixed to a single NUMA domain (in terms of cores and memory) to avoid random memory effects of cross NUMA memory access during running time analysis.

Experiments were performed using the Ubuntu Server 16.04 operating system. Java was installed in Version 1.8 (Oracle Java HotSpot(TM) 64-Bit Server VM).

Table 3.1: Descriptive statistics of the real-world datasets for StruClus evaluation.

		$\lvert V\rvert$				$\lvert E\rvert$				
Dataset $\mathcal{G}$	$\lvert\mathcal{G}\rvert$	Classes	Min.	Max.	Avg.	Min.	Max.	Avg.	$\lvert\mathcal{L}_V\rvert$	$\lvert\mathcal{L}_E\rvert$
AnchorQuery	65 700	12	11	90	79.19	11	99	86.02	6	5
Heterocyclic	10 000	39	9	69	42.99	10	79	47.35	25	5

StruClus, PROCLUS, and Kernel K-Means are implemented in Java. Java was configured to use a minimum and maximum heap size of 50GiB (`-Xmx` and `-Xms` parameter) to give enough headroom for other system processes and Java Virtual Machine overhead. Setting the minimum heap size avoids, that the Java VM incrementally increases the heap size during runtime, which in turn can cause some avoidable garbage collections. The SCAP algorithm was implemented in C++ and was compiled using the GCC 4.9.3 compiler with O3 optimization level. The implementation of StruClus and SCAP are shared-memory parallelized and configured to use all available cores during experiments. PROCLUS and Kernel K-Means are sequential implementations. Their running times must be interpreted keeping in mind, that a well-scaling parallelization is most probably possible.

All experiments are repeated 30 times if the average running time of the first 15 runs was below 2 hours and 15 times otherwise. As already described in section 2.6.3.5 quality is measured using the Normalized Variation of Information (NVI), Fowlkes & Mellows Indes (FM), and Purity measures for comparison with a ground truth. Since SCAP produces overlapping clustering results, Purity is used for comparisons with SCAP and the other measure are used for comparisons with the partitioning clustering algorithms.

3.2.9.2 Datasets

Datasets with known ground-truth class labeling are used for the evaluation of StruClus.

[Hum+18] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

AnchorQuery and Heterocyclic are two real-world chemical datasets. Both are preliminary versions of subsets of the molecular CH*I*PMUNK [Hum+18] library. Each molecule is the result of a chemical reaction of purchasable building blocks (i.e., smaller molecules). The chemical reaction type during synthesization is used as the ground truth class label. For the AnchorQuery dataset, 12 reaction types from the Anchor Biased Library[1] are utilized. Another 39 distinct reaction types from SciFinder[2] are used for the Heterocyclic dataset.

Table 3.1 shows some descriptive statistics of the real-world datasets. The reaction type has a major influence on the size of the graphs and the label distribution. While the Heterocyclic dataset has an average vertex count of 43 and an average edge count of 47, the numbers for the AnchorQuery dataset are nearly double as high with 79

[1] `http://anchorquery.csb.pitt.edu/reactions/`
[2] `https://scifinder.cas.org`

and 86. The set of edge labels is identical with five bond types: single, double, triple, aromatic, and single or double. With 25 distinct chemical elements, the vertex label set of the Heterocyclic dataset is much more diverse than that of the AnchorQuery dataset with five labels.

In addition to the real-world datasets, synthetic datasets of varying cardinalities are created. The goal was to create datasets of multiple common patterns per cluster. In addition, complete noise graphs and inter-cluster common noise patterns should be added, such that clustering algorithms should prefer clusters of multiple matching patterns over selecting clusters with single pattern commonalities or patterns of low representative power. The synthetic datasets have an average number of ≈ 35 vertices and ≈ 51 edges. They contain 100 clusters and 5% noise graphs that are generated as follows. Each cluster is formed by three cluster-specific seed patterns, which are generated using a Poisson distributed number of vertices with a mean of ten. Vertex labels are drawn from a set of ten labels, with label weights drawn from an exponential distribution. Then, each edge of the complete graph is selected with a probability of 10% and three distinct edge labels are randomly assigned (weighting as above). In addition to the three cluster-specific seed patterns, a common noise seed pool of ten patterns is generated the same way. Finally, each graph of each cluster is generated by a random selection of 3.5 seed patterns on average from the three cluster-specific seeds and two random seeds of the noise pool. The so collected seeds are then combined by connecting them via random edges. The number of edges per seed pair is drawn from a Poisson distribution with a mean of five. The complete dataset is then created by selecting a cluster at random and generating a cluster-specific graph as described above for 95% of the graphs. In a last step, 5% noise graphs are added by randomly creating vertices and randomly connecting them. The average number of vertices and edges are drawn from a Poisson distribution with the same means as the generated cluster graphs. The graph generation can lead to disconnected graphs. They are replaced with connected graphs, generated with the same method.

3.2.9.3 Parametrization of StruClus

If not subject to discussion or explicitly stated otherwise the StruClus algorithms is parameterized as follows. The maximum error probability for algorithm 4 is set to $\alpha = 0.5$. The minimum sample size is set to $s_{\min} = 30$. The number of representative candidates during the representative update step was is to $m = 25$. The maximum number of representative per cluster is set to $r_{\max} = 3$. Too split clusters of low homogeneity the threshold is set to $\text{relHom}_{\min} = 0.6$. The separation threshold for cluster merging is set to $\text{rSim}_{\max} = 0.7$. To map the relative homogeneity of a cluster to a minimum support threshold for representative mining, the parameters ls, hs, lr, and hr are set according to the discussion in section 3.2.2.3. The exact values are ls = 0.7, hs = 0.95, lr = 0.6 (i.e., the the threshold for splitting), and hr = 1.1. Convergence is determined with an averaging with of $w = 3$.

Table 3.2: Comparison of average running times and quality values w.r.t. the cluster member sampling strategy on the synthetic datasets. CV is the max. coef. of variation per column. running times are given in hours.

$\lvert\mathcal{G}\rvert$	StruClus			StruClus (Exact Support)		
	Running Time	NVI	FM	Running Time	NVI	FM
CV	< 0.08	< 0.03	< 0.07	< 0.07	< 0.02	< 0.04
1 000	**0.05**	**0.90**	0.75	**0.05**	**0.90**	**0.77**
5 000	**0.15**	0.94	0.85	0.27	**0.95**	**0.86**
10 000	**0.19**	**0.95**	**0.87**	0.59	0.93	0.85
50 000	**0.33**	**0.94**	**0.87**	2.69	0.92	0.85

3.2.9.4 Influence of Cluster Member Sampling

The described sampling strategy for cluster members in algorithm 4 is bound by a maximum error probability α. In the worst case, this approach is not faster than the exact determination of the support values. However, such a worst-case scenario is unlikely in practice. This section will thereby demonstrate the effect of the sampling strategy on the running time and the quality of the clustering for the synthetic datasets of varying sizes.

The evaluation is set up as follows. StruClus is run in two different configurations. In the first one, it is configured and parameterized as described in section 3.2.9.3. This includes the cluster member sampling strategy with a maximum error probability of $\alpha = 0.5$. A relatively high error is affordable for two reasons. First, the Bonferroni multiple hypothesis testing correction (cf., lemma 3.3) is quite conservative and corrects the FWER for the worst case instead of the expected case. It is assumed, that the test decision for every test is made with a test statistic exactly at the rejection border. In addition, $v_{\max}$ is most likely an overestimation, successful extensions rule out additional extensions between pairs of vertices, and it is extremely unlikely that the worst-case number of sample size doubling steps is necessary. In other words, the real error is most likely much smaller in the expected case. Second, the ranking of frequent patterns (cf., section 3.2.2.2) can filter bad patterns candidates in an independent non-probabilistic step.

In the second configuration, StruClus is run with an exact calculation to determine whether a pattern is frequent or not. Thus, the cluster members are successively tested on subgraph isomorphism to the pattern until either the minimum support threshold is reached or the remaining untested graphs are not enough to reach the threshold.

Table 3.2 shows the results of this evaluation setup. Quality-wise, no significant difference between the two settings can be detected. Differences are always in the range of the standard deviation and wins or losses are almost uniformly distributed between the settings. In terms of running time, no difference can be seen for a dataset

of 5000 graphs. This can be explained by the minimum sample cardinality $s_{\min} = 30$ (cf., algorithm 4). For most clusters, this threshold is about as large as the clusters themselves. With increasing dataset size, the exact setting shows an almost perfect linear scaling behavior. The sampling setting, on the other hand, shows a sublinear growth of running time. For a dataset size of 50 000, the speedup w.r.t. the member sampling strategy was about already above eight and is expected to increase for larger datasets. This shows, that the frequent test is indeed a computational hotspot for the StruClus algorithm and the sampling strategy can make a big running time difference without sacrificing quality.

3.2.9.5 Influence of the Number of Candidates

The number of representative candidate patterns (i.e., parameter m of algorithm 3) that are mined during the representative update step is fixed to an a priori chosen value. The following experiment will evaluate the influence of this parameter on the clustering quality. While more sophisticated selection strategies for this parameter are easy to imagine (e.g., mine more patterns until the ranking function is no longer increasing), the evaluation will show, that such a strategy is not worth the effort.

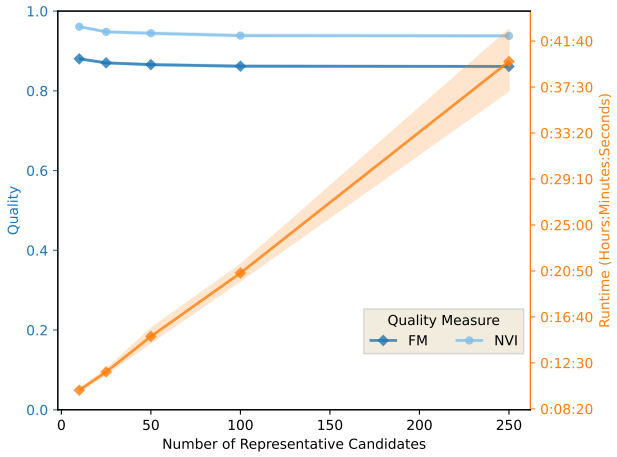
Figure 3.5 shows the quality of the clustering results for different values of m. The clustering quality is largely insensitive in the tested range. The low number of necessary pattern candidates might be a result of the already highly homogeneous clusters, which may cause the number of maximal frequent patterns to be relatively small. Furthermore, missed pattern commonalities might be found in later iterations.

The running time of the algorithm shows a linear increase with the number of candidates. This highlights two aspects w.r.t. the algorithm behavior. First, the representative update is dominating the overall running time. Second, the selection of relatively few representative candidates does not lead to an increase in the number of iterations, e.g., because convergence is slower in such settings. The result of the AnchorQuery dataset, displayed in fig. 3.5c, shows a high variance in running time. This is most probably an effect caused by the much larger representative sizes of 36.25/97 (mean/maximum), compared to 15.42/46 or 9.53/56 for the Heterocyclic and synthetic datasets, respectively. Very large representatives can lead to very hard instances for the subgraph isomorphism and maximum common subgraph algorithms with exponential worst-case complexity, which can dominate the overall running time.

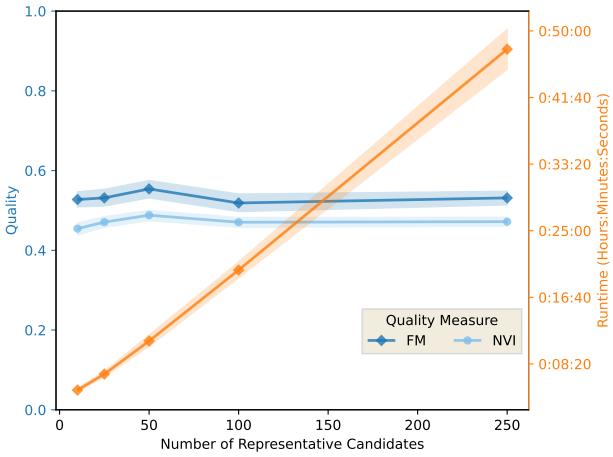
To conclude, a relatively low number of representative pattern candidates can be selected to achieve a high performance without sacrificing quality.

3.2.9.6 Convergence and Iterative Changes

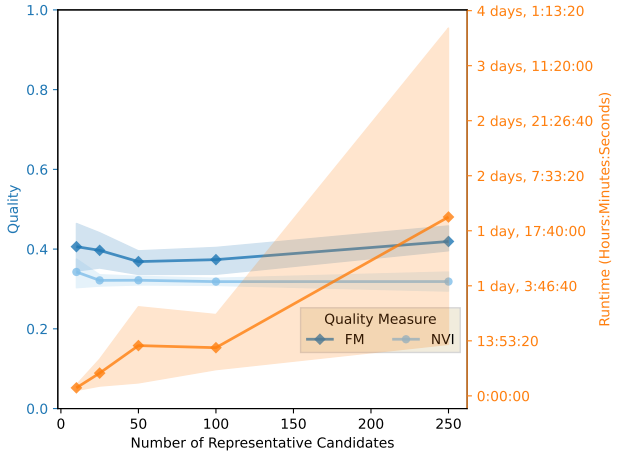
The behavior of StruClus w.r.t. the iterative changes is subject of interest in this section. The experiment was set up as follows. The number of iterations was fixed to 30. Then for each iteration, the number of clusters, the weighted average cluster homogeneity (i.e., the numerator of the convergence criterion, cf., eq. (3.16)), and clustering quality were measured. In addition, the termination iteration c w.r.t. to the termination criterion, i.e., eq. (3.17), was determined.



(a) Synthetic Dataset (size 10 000)

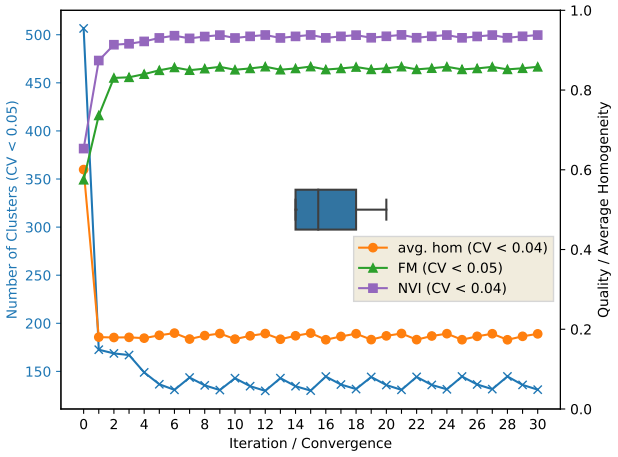


(b) Heterocyclic Dataset

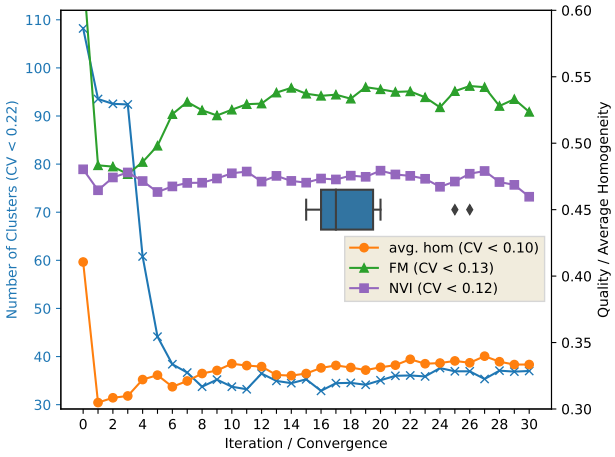


(c) AnchorQuery Dataset

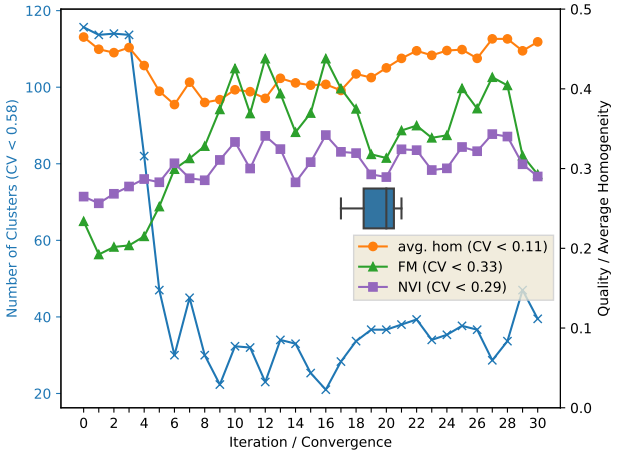
Figure 3.5: Influence of the number of candidate representative patterns on the clustering quality. CV is the max. coef. of variation per measurement category.



(a) Synthetic Dataset (size 10 000)



(b) Heterocyclic Dataset



(c) AnchorQuery Dataset

Figure 3.6: Iterative behavior of StruClus for a fixed number of 30 iterations. The iteration in which the algorithm would terminate is given as box plot on the x-axis. The average homogeneity and the number of clusters, i.e., the two criteria for convergence determination, are given by the orange and blue line plots. In addition, the purple and green line plots represent the NVI and FM quality values for intermediate results.

Figure 3.6 shows the results for the different datasets. It is noticeable, that the pre-clustering step usually creates many more clusters than present in the later iterations. As a result, homogeneity is also high after the pre-clustering. However, the number of clusters quickly drops to a fraction of the initial number and large changes in almost all measurements are observable in the first iterations. In particular, homogeneity decreases as well, but in a less drastic way. Thereby, the convergence criterion is actually increasing. For the synthetic and AnchorQuery datasets, quality increases in both measures with further iteration. Thus, it can be assumed, that the iterative refinement of StruClus is useful w.r.t. to the underlying ground truth. The AnchorQuery increase in quality is less smooth compared to the synthetic dataset, aligning with some fluctuation in the number of clusters. These fluctuations are the result of cluster splitting and merging steps and highlight the necessity to smooth convergence over some iterations. The Heterocyclic dataset shows some differences in comparison to the other two datasets. First, the pre-clustering results has the highest quality w.r.t. the FM quality measure. Thus, the minimum separation constraint might be chosen to strict and a more granular clustering might have led to better results in terms of the ground truth. However, it should be noted that the ground truth is not a universal truth and that other good clusterings, highlighting different aspects, might exist. Second, the correlation between the FM and NVI measures is low. The above effect of a high pre-clustering quality is not observable for the NVI measure. Later iterations do not show any significant change in quality for the NVI measure, while FM increases again after its first drop. This highlights the fact, that quality measures themselves weigh different aspects of clusterings differently. Especially counting pair measures, such as FM, are often sensitive to the number of clusters, i.e., the number of clusters influences the expected value for e.g., random clusterings. On the other hand, FM does seem to align much better with the latter increase in cluster homogeneity.

Convergence usually stops the refinement after a few iterations (Iterations 15 to 21 in our examples with two outliers). As a result of the smoothing, this happens a bit later than desired from a performance point of view. Nevertheless, the observed fluctuations in the number of clusters would make a shorter averaging window less reliable and may cause premature termination.

3.2.9.7 Parallel Scaling

In section 3.2.8.3 a simple parallelization of StruClus was introduced. To evaluate the scaling of the algorithm with the number of cores, the number of cores was varied using the synthetic dataset of cardinality 10 000. To avoid parallel advantages on the java virtual machine level (e.g., through parallel garbage collection) the algorithm instance was also pinned to a fixed set of cores.

Figure 3.7 shows the results of the experiment. With 8 cores StruClus achieves a speedup of 7.15. Including the virtual hyperthreading cores, the speedup increased to 9.11.

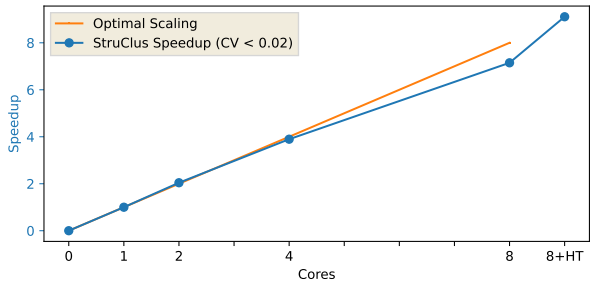


Figure 3.7: Scaling of the running time of StruClus with the number of cores on the synthetic dataset of size 10 000. HT stands for hyperthreading.

3.2.9.8 Comparison with Other Clustering Algorithms

In this section, the performance of StruClus is compared to other clustering algorithms in terms of performance and quality. The synthetic, AnchorQuery, and Heterocyclic datasets are used for evaluation. The competing algorithms are SCAP, PROCLUS, and Kernel K-Means. Since there exists a very high number of clustering algorithms, it is impossible to compare StruClus to a wide range of them. The three comparison algorithms are selected with the following rationale. SCAP is a structural algorithm and, to the best of my knowledge, the only structural clustering algorithm for large-scale datasets that is not limited to simple graph classes. Seeland, Karwath, and Kramer [SKK14] compared SCAP with BIRCH [ZRL96], an expectation-maximization Dirichlet process clustering algorithm for small graph datasets of Tsuda and Kurihara [TK08], and a previous variant of their own algorithm [See+11] which was limited to smaller datasets. SCAP outperformed all competing algorithms in terms of clustering quality. BIRCH was able to succeed in terms of running time. Thus, to some limited extent, the experimental comparison of StruClus with SCAP also compares to the above-mentioned algorithms. PROCLUS is a fast projected clustering algorithm with noise detection. A projected clustering algorithm is included, since the high intrinsic dimensionality of graph datasets as shown by Kriege, Mutzel, and Schäfer [KMS14a] suggests, that the application of an algorithm for high dimensionality is necessary for high-quality results. Furthermore, PROCLUS was studied intensively and performed well in various comparisons of projected clustering algorithms (e.g., [Mül+09; PM06]). The Kernel K-Means algorithm was selected since it allows the applicability of very flexible graph kernels (cf., paragraph 2.6.3.1.4). The parametrization of the comparison algorithms is as follows.

SCAP has two parameters: (a) the minimum vertex count of a shared common subgraph pattern for a cluster, and (b) a similarity threshold for fingerprint-based prepartitioning and representative power of a cluster commonality. During the following experiments, parameter (a) is set to eight, which corresponds to 80% of the average seed size in the synthetic dataset. Also, the common subgraph pattern of the different

[SKK14] Seeland, Karwath, and Kramer "Structural clustering of millions of molecular graphs". 2014

[ZRL96] Zhang, Ramakrishnan, and Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases". 1996

[TK08] Tsuda and Kurihara "Graph Mining with Variational Dirichlet Process Mixture Models". 2008

[See+11] Seeland et al., "Parallel Structural Graph Clustering". 2011

[KMS14a] Kriege, Mutzel, and Schäfer "Practical SAHN Clustering for Very Large Data Sets and Expensive Distance Metrics". 2014

[Mül+09] Müller et al., "Evaluating Clustering in Subspace Projections of High Dimensional Data". 2009

[PM06] Patrikainen and Meila, "Comparing Subspace Clusterings". 2006

reaction types in the AnchorQuery and Heterocyclic datasets are usually above this size. The second parameter (b) was set to 0.2. It is important to understand, that this parameter largely affects the clustering outcome and performance. Selecting a high value will force a high minimal representative power of the cluster representative. If a large value is chosen, smaller commonalities will not be considered during the clustering. For the synthetic dataset, for example, a too high value would rule out the seed patterns as commonalities and lead to a bad clustering. If a low value is chosen, cluster commonalities may be not meaningful (SCAP chooses an arbitrary pattern above this threshold, not necessarily the largest one) and the pre-partitioning will be less effective, which in turn causes an increased running time. The value of 0.2 was the highest value to select a seed pattern in a graph of the synthetic dataset, which is composed of the maximum number of seed patterns (i.e., 5). Again, the selection of a value of 0.2 makes also sense w.r.t. the size of the reaction-type specific substructures of the Heterocyclic and AnchorQuery datasets if compared with the complete synthesized graphs. Thus, both parameters of SCAP are adjusted to fit the ground truths in our datasets in order to get an optimal clustering result.

PROCLUS has also two parameters: (c) the number of clusters, and (d) the mean dimensionality of the cluster subspaces. With the same rationale as above, parameter (c) is set to the ground truth. Parameter (d) was set to 20, for which the best clustering quality was achieved in comparison with the values in $\{10, 20, 30, 40, 50\}$ on the synthetic dataset. In addition to the parametrization, a feature extraction method (cf., section 2.6.3.3) is necessary to convert the graph dataset into a vectorial representation. For the experiments, all subgraph patterns of size 3 where enumerated. The feature vector of a graph G then contains one representational dimension for each pattern and the value of each dimension is the number of distinct subgraphs of size 3 of G that are isomorphic to the associated pattern. In other words, the counting does not include automorphisms. The feature counts for the synthetic datasets were in the range of 7 000 to 10 000. The application to the AnchorQuery and Heterocyclic datasets resulted in much lower feature counts of 274 and 133 features.

Kernel K-Means is parameterized with (e) the number of clusters, and (f) a kernel. Again, parameter (e) is set to the ground truth. Parameter (f) is set to the Graphlet Kernel GK C3 [She+09], which uses the same feature space as the feature extraction method for PROCLUS.

[She+09] Shervashidze et al., "Efficient graphlet kernels for large graph comparison". 2009

Table 3.3 shows the results for the synthetic datasets of varying cardinality. Missing values indicate running times above a two-day time limit. Up to a dataset cardinality of 10 000, SCAP was the fastest algorithm. For cardinalities of 50 000 and above StruClus is the fastest algorithm with an increasing margin. For the dataset of cardinality 500 000, StruClus was already more than 13 times faster than SCAP on average. StruClus was the only algorithm, which was able to cluster the dataset of 1 million graphs in less than two days and was finished in 2.73 hours on average. While SCAP shows an almost quadratic running time scaling behavior, the running time of StruClus was sublinear up to a cardinality of 500 000. The sublinear behavior is caused by the sampling of cluster members (cf., section 3.2.9.4). However, at some point other linear running time aspects (cf., theoretical analysis in section 3.2.7) are becoming more dominant. PROCLUS and Kernel K-Means were much slower.

Table 3.3: Comparison of StruClus with SCAP, PROCLUS, Kernel K-Means on the synthetic datasets. CV is the max. coef. of variation per column. Missing values have running times above two days.

$\|\mathcal{G}\|$	StruClus				SCAP		PROCLUS			Kernel K-Means		
	Running Time	NVI	FM	Purity	Running Time	Purity	Running Time	NVI	FM	Running Time	NVI	FM
CV	< 0.08	< 0.03	< 0.07	< 0.03	< 0.06	< 0.02	< 0.32	< 0.11	< 0.22	< 0.01	< 0.01	< 0.01
1 000	0.05	**0.90**	**0.75**	**0.90**	< **0.01**	0.83	0.13	0.58	0.26	0.02	0.77	0.54
5 000	0.15	**0.94**	**0.85**	**0.98**	< **0.01**	0.84	4.99	0.50	0.24	2.87	0.84	0.67
10 000	0.19	**0.95**	**0.87**	**0.99**	**0.03**	0.83	11.91	0.49	0.24	10.32	0.86	0.78
50 000	**0.33**	0.94	0.87	**0.99**	0.38	0.83	–	–	–	–	–	–
100 000	**0.47**	0.93	0.86	**0.99**	1.21	0.86	–	–	–	–	–	–
500 000	**1.35**	0.93	0.86	**0.99**	18.15	0.83	–	–	–	–	–	–
1 000 000	**2.73**	0.91	0.84	0.98	–	–	–	–	–	–	–	–

Both algorithms are unable to cluster the dataset of cardinality 50 000 in less than two days. Although it should be remembered, that both algorithms are implemented as single-core algorithms, even an optimal speedup with the number of physical cores would not be enough to outperform their competitors. Additionally, both show a clear superlinear growth in running time, while Kernel K-Means shows the sharpest increase with an almost perfect quadratic behavior. Thus, they are not competitive in comparison with StruClus for large-scale datasets.

In terms of quality, all experiments on the synthetic datasets are in favor of StruClus. With an exception of the smallest dataset, all quality values of StruClus are stable and in the range of the standard deviation. For the smallest dataset, StruClus shows a small drop in quality which may have to do with the small number of members per cluster. A visual inspection of the clustering results indicates more clusters formed by noise commonalities. Thus, the overlap between common cluster seed patterns might have been too small to reproduce them reliably. SCAP did not show a similar behavior, but the clustering quality of StruClus was still higher for the small dataset.

The second best quality (Purity values for PROCLUS and Kernel K-Means were all below 0.55) was observed for SCAP with Purity values in the range of $[83, 86]$ compared to $[98, 99]$ for StruClus (excluding the outlier for the smallest dataset). However, the direct comparison must be interpreted with care, since SCAP is an overlapping clustering algorithm. Since the Purity measure cannot detect cluster splits of the ground truth, it is possible that a close reproduction of the ground truth is a subset of the clustering and the other clusters represent noise seed pattern commonalities. Such a situation would be a very good clustering result and would not be reflected by the Purity measure appropriately. However, with 129 clusters on average, this hypothesis does not seem to be very likely since it is very close to the number of clusters in the ground truth. This is especially true when compared to the 176 clusters on average for the StruClus algorithm. Additionally, SCAP does only create new clusters for additional seeds if there exist graphs that cannot be assigned to an existing cluster. Thus, additional noise seed clusters would not be

Table 3.4: Comparison of StruClus with SCAP, PROCLUS, Kernel K-Means on the real-world datasets. CV is the max. coef. of variation per column. Missing values have running times above two days.

dataset	StruClus				SCAP		PROCLUS			Kernel K-Means		
	Running Time	NVI	FM	Purity	Running Time	Purity	Running Time	NVI	FM	Running Time	NVI	FM
CV	< 2.91	< 0.09	< 0.12	< 0.08	< 0.02	< 0.01	< 0.03	< 0.05	< 0.08	< 0.01	< 0.01	< 0.01
AnchorQuery	**2.47**	0.36	0.43	0.85	–	–	–	–	–	–	–	–
Heterocyclic	1.07	**0.46**	**0.53**	**0.66**	**0.01**	0.58	**0.01**	0.29	0.29	*3.03* (Subset)	*0.27*	*0.29*

created if the ground truth would have been found in advance. Therefore, a more likely hypothesis is, that the single pattern commonality approach of SCAP has no chance to differentiate between noise and clusters seed patterns. Thus, whether a noise or cluster seed is selected as cluster commonality is purely determined by chance.

The high purity values for StruClus indicate, that StruClus has split up some ground-truth clusters, but otherwise almost perfectly reproduced them. This over-splitting might also be a result of the dataset noise, which was bundled as a single noise cluster in the ground truth. Random commonalities in this noise might still be interpreted as clusters by StruClus.

While Kernel K-Means still shows reasonable clustering results w.r.t. clustering quality, PROCLUS shows the worst quality with a large margin. Especially the counting pair FM measure indicates a rather small ground truth agreement with a value 0.24.

Table 3.4 shows the comparison of the real-world datasets AnchorQuery and Heterocyclic. It stands out, that only StruClus was able to cluster the AnchorQuery dataset in less than two days in approximately two and a half hours. This is remarkable compared to the running times of the Heterocyclic dataset. While the AnchorQuery dataset is only about six to seven times larger, the running times of *SCAP* and PROCLUS were less than a minute for the Heterocyclic dataset. The larger graph sizes may be an explanation for the bad performance of the SCAP algorithm. However, the feature counts of 274 for the AnchorQuery and 133 for Heterocyclic datasets do not make a huge difference for the PROCLUS algorithm, especially when compared to 7 000 to 10 000 features for the synthetic datasets. The Kernel K-Means algorithm was also surprisingly slow on the Heterocyclic dataset and was unable to finish a single run in less than one day. A random subset of 5000 graphs was therefore given as input for this algorithm. Qualitywise, StruClus outperformed all of its competitors with a good margin. The quality scores for StruClus in compression with PROCLUS and Kernel K-Means are almost doubled. In summary, StruClus demonstrated a more predictable running time behavior and higher quality clusterings for the real-world datasets.

3.2.9.9 Application to Chemical De-Novo Library Novelty Analysis

In this section, a real-world application of StruClus is presented. A summary of joint work with other authors [Hum+18] is given in the following.

[**Hum+18**] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

Chemical de-novo libraries of molecules provide a pool of candidates to identify novel lead structures for drug discovery. In drug discovery, lead structures serve as a starting point in an iterative refinement process, e.g., to minimize side effects or increase the potency of a drug. Today, a lead structure is often identified by a virtual analysis of chemical properties. In a second step, these structures are then evaluated in the wet lab, i.e., synthesized in the real world.

The molecular de-novo library CH*I*PMUNK [Hum+18] was designed for this purpose. In order to contain molecules, that can be synthesized in the real world, CH*I*PMUNK is based on the simulation of chemical reactions which are known to be reproducible in the wet lab. Furthermore, the reactants or educts are taken from libraries that contain purchasable molecules. However, the real-world accessibility of molecules in the library is not the only important property in order to be a tool to find useful lead structures. The novelty of lead structures is equally important to broaden the scope into previously undiscovered fields. As such, the library must contain molecules, which are not already contained in known and exhaustively studied libraries. More precisely, similar structures should also not be contained in other libraries. Such similar structures would most probably show similar characteristics and thereby could serve as an alternative lead structure for the same task. To judge the novelty of a library it is thereby necessary to compare the content of the library to other libraries in terms of structural similarity.

[**Hum+18**] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

StruClus was used for the novelty analysis of CH*I*PMUNK. The test setup was as follows. The CH*I*PMUNK sub-libraries (CH*I*PMUNK Heterocycle, CH*I*PMUNK MCR, CH*I*PMUNK TMC) were clustered together with libraries of purchasable building blocks (namely ZINC [Irw+12], eMolecules[3], and MolPort[4]) and the public bioactivity database ChEMBL [Ben+13]. Each resulting cluster is interpreted as a distinct part of the chemical space (i.e., the search space of possible molecular structures). Now, if some clusters of the clustering are purely covered by graphs from a single library, this can be interpreted as a part of the chemical space that is unique to the specific library. The minimum separation constraint of StruClus guarantees the diversity of different clusters, i.e., that different clusters are not covering a similar part of the chemical space.

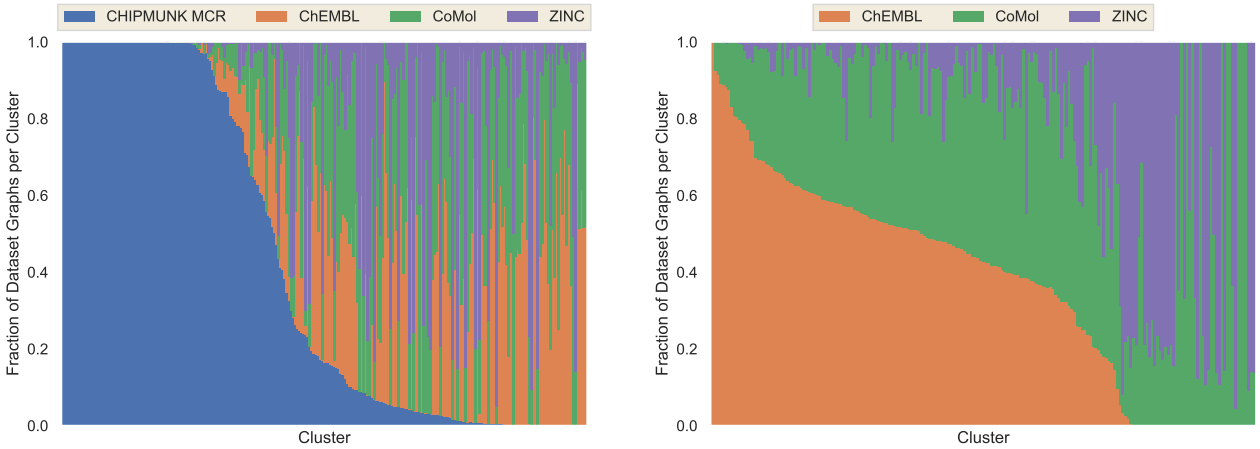
[**Irw+12**] Irwin et al., "ZINC: A Free Tool to Discover Chemistry for Biology". 2012

[**Ben+13**] Bento et al., "The ChEMBL bioactivity database: an update". 2013

The large quantity of molecules in the CH*I*PMUNK and in the comparison libraries stresses the need for scalable clustering algorithms in this field of application. Overall, the clustering algorithm had to cluster 44 123 633, 67 029 742, and 24 254 434 molecules for the novelty analysis of the CH*I*PMUNK MCR, CH*I*PMUNK TMC, and CH*I*PMUNK Heterocycle, respectively.

Figure 3.8a shows the results of this evaluation for CH*I*PMUNK MCR sub-library as an example of the complete analysis. Vertical bars in this plot show the fraction of graphs from a certain molecular library (depicted by colors). Clusters are plotted

[3] `http://www.emolecules.com`
[4] `http://www.molport.com/`



(a) CH*I*PMUNK MCR Evaluation. The x-axis is sorted by the fraction of graphs from the CH*I*PMUNK MCR sub-library.

(b) Comparison with the same analysis conducted without CH*I*PMUNK. The x-axis is sorted by the fraction of graphs from the ChEMBL sub-library.

Figure 3.8: Cluster novelty analysis for the CH*I*PMUNK MCR sub-library. Clusters are displayed on the x-axis. The fraction of graphs of each dataset is displayed as stacked bar for each cluster. CoMol is the combined library of eMolecules and MolPort.

on the x-axis and sorted by the pureness of graphs from the chipmunk library. Thus, most unique clusters of CH*I*PMUNK MCR are shown on the left side of the plot. As the figure shows, there exists a large portion of the clusters which are uniquely covered by the CH*I*PMUNK MCR library. To interpret the significance of the plot, fig. 3.8b shows the same analysis for the ChEMBL library when CH*I*PMUNK MCR is left out. It becomes evident, that the ChEMBL library shows much less uniqueness in comparison with CH*I*PMUNK MCR.

3.3 Summary

A novel structural clustering algorithm, called StruClus, was presented. It scales easily to graph datasets with millions of graphs. With the help of a two-fold sampling strategy with stochastic error bounds, it was possible to keep the number of the costly subgraph isomorphism tests low. In addition to the linear worst-case running time, this leads to low constant factors in the complexity of the algorithm without compromising quality. A cluster merging and splitting step was introduced to achieve well-separated clusters even in the high-dimensional pattern space. In combination with the homogeneity and separation-aware representative selection, this leads to high-quality clustering results. As such, StruClus outperforms existing structural and vectorial approaches in terms of running time and quality. The cardinality constrained set of representatives with dynamic pattern size leads to highly interpretable clustering results in the context of visual analytics.

Chapter 4

Distributed Subgraph Pattern Coverage Maximization

Structural summaries or representatives can play an important role in various contexts. The applications are versatile, ranging from human analysis such as visual analytics (cf., Scaffold Hunter example in section 1.1.3) over the extraction of features for further processing to compressed representations of datasets. Several structural clustering algorithms [e.g., Agg+07], including the StruClus algorithm presented in section 3.2, use representative mining to create cluster descriptions.

[**Agg+07**] Aggarwal et al., "Xproj: a framework for projected structural clustering of xml documents". 2007

This chapter will discuss the problem to mine structural patterns to summarize a graph dataset. In contrast to cluster representatives, these structural summaries are tailored towards diverse and non-homogeneous collections of graphs. As a consequence, the aspect of redundancy becomes important to provide small summaries with a high expressiveness. When selecting representatives by a simple ranking function, as done for the StruClus algorithm, multiple patterns may represent the same portion of graphs. The representative definition in this chapter will address the issue of redundancy explicitly. To cope with the computational complexity of this task, distributed and parallel algorithms are of key interest.

4.1 Related Work

The term *representative pattern mining* is not specific to graph pattern mining. The settings in which it is used vary, and there exists no unique definition. The common goal is to find patterns in a dataset that describe the content of the dataset.

Thus, the mined patterns should describe the dataset close enough to preserve some characteristics. Even the frequent pattern mining problem is sometimes subsumed under the umbrella of representative pattern mining since the frequent patterns characterize a dataset. However, the number of frequent subgraph patterns may be too large to be meaningful, even much larger than the original dataset. Given this observation, some representative graph pattern mining algorithms are simply motivated by the goal to reduce frequent patterns [Has+07]. Consequentially, the cardinality k of the representative set is often restricted by some parameter. This class of pattern mining problems are called *(cardinality) constrained*, said to have a *cardinality budget*, or denoted by *top-k* representative pattern mining problems. In addition to this complexity reduction, many representative pattern mining algorithms include further objectives and quality criterion w.r.t. the domain of application.

[**Has+07**] Hasan et al., "ORIGAMI: Mining Representative Orthogonal Graph Patterns". 2007

Representative pattern mining is used for classified [e.g., Wan+16] and unclassified [e.g., RHS14; NR16] datasets. Similar to the machine learning nomenclature and in alignment with [Mör07], this thesis will use the terms supervised and unsupervised pattern mining for classified and unclassified datasets. In the former case, representative pattern mining is often interchangeably used with discriminative pattern mining. Thus, the objective is the enumeration of class-specific patterns or features, e.g., for later classification in order to enable the classifier to achieve good classification performance or for manual inspection. In the latter case, the objective function is to find a set of patterns or features that best describes the dataset at hand without the knowledge of a supervised setting. In the context of graph pattern mining, unsupervised representative pattern sets are also denoted by structural summaries. The unsupervised setting includes problems, that incorporate some importance function [e.g., NR18], which closes the gap to the supervised setting when the importance function is versatile enough. Graph pattern representatives often represent a subset of graphs from the dataset by the means of some distance function that identifies similar dataset graphs. In this case, the distance function is either combined with a threshold to identify similar graphs [e.g., Has+07; NR16; NR18] or the distance function is used to compare the relative closeness of the representative candidates [e.g., ZYL09] to the dataset graphs. Additionally, some significant graph pattern mining algorithms fall in the category of representative pattern mining. In this domain, solutions independent of a similarity measure can be found. For example, He and Singh [HS06] presented an unsupervised representative mining algorithm, that filters frequent subgraphs of a precise dataset based on their deviation from a randomized stochastic model. Other significant pattern mining algorithms handle classified datasets by finding patterns that significantly differ in their support w.r.t. the class labels [e.g., TdT16].

[**Wan+16**] Wang et al., "RPM: Representative Pattern Mining for Efficient Time Series Classification". 2016

[**RHS14**] Ranu, Hoang, and Singh, "Answering Top-k Representative Queries on Graph Databases". 2014

[**NR16**] Natarajan and Ranu, "A Scalable and Generic Framework to Mine Top-k Representative Subgraph Patterns". 2016

[**Mör07**] Mörchen, "Unsupervised Pattern Mining from Symbolic Temporal Data". 2007

[**NR18**] Natarajan and Ranu, "Resling: a scalable and generic framework to mine top-k representative subgraph patterns". 2018

[**Has+07**] Hasan et al., "ORIGAMI: Mining Representative Orthogonal Graph Patterns". 2007

[**ZYL09**] Zhang, Yang, and Li, "RING: An Integrated Method for Frequent Representative Subgraph Mining". 2009

[**HS06**] He and Singh "GraphRank: Statistical Modeling and Mining of Significant Subgraphs in the Feature Space". 2006

[**TdT16**] Terada, duVerle, and Tsuda, "Significant Pattern Mining with Confounding Variables". 2016

A common quality criterion of representative miners is the diversity or redundancy of the representative set, i.e., the representatives should be either dissimilar to each other or represent dissimilar subsets of the dataset. For example, Hasan et al. [Has+07] mine α-orthogonal sets of representatives, where the pairwise similarities of representatives are bound by a similarity threshold α. Additionally, they introduce the concept of β-representativeness, to minimize the set of unrepresented graphs in the dataset. Li and Wang [LW15] presented an unsupervised representative approximate frequent subgraph mining algorithm, intending to reduce side effects of noisy data by incorporating an

[**LW15**] Li and Wang "REAFUM: Representative Approximate Frequent Subgraph Mining". 2015

approximate support definition. Since the number of approximate frequent subgraph patterns grows with the allowed deviation from the exact support definition, they added a redundancy reduction filtering step into their algorithm. This is related to the observation in the feature selection community, that the top-k ranked features do not necessarily form the best feature set with cardinality k [Cov74], since features might be redundant. A multitude of feature selection publications have identified this redundancy reduction as their core problem besides maximizing feature dependency on the class labels. For example, Peng, Long, and Ding [PLD05] defined a minimum redundancy criterion to minimize the mutual information of features w.r.t. to the datasets class labels in the classified setting.

As a consequence, representatives of many representative mining objectives cannot be judged independent from each other, i.e., the representative power of a representative set usually depends on the combination of representatives. Due to the combinatorial explosion, many of these representative pattern mining problems are NP-hard [e.g., NR18; PLD05] and brute force approaches are usually infeasible. While this is also true for submodular set objective functions (cf., section 2.4), there exist fast approximation algorithms for this subset of objective functions. For this reason, the utilization of submodular objective functions is a common pattern in representative pattern mining [NR16; NR18; Tho+10].

Most representative graph pattern mining algorithms use frequent, closed, or maximal subgraph pattern mining algorithms for candidate generation. While the majority uses the classical bottom-up full enumeration strategies (cf., section 2.6.2.1) there are some exceptions. Hasan et al. [Has+07] sampled maximal frequent subgraphs utilizing random walks with strictly increasing poset ranks. Natarajan and Ranu [NR18] used reinforced walks on the a dynamically extended edit map, where each subgraph pattern is connected to another subgraph pattern if the edit distance (cf., section 2.6.3.4) between the two is one.

[Cov74] Cover, "The Best Two Independent Measurements Are Not the Two Best". 1974

[PLD05] Peng, Long, and Ding "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". 2005

[NR18] Natarajan and Ranu, "Resling: a scalable and generic framework to mine top-k representative subgraph patterns". 2018

[PLD05] Peng, Long, and Ding, "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". 2005

[NR16] Natarajan and Ranu, "A Scalable and Generic Framework to Mine Top-k Representative Subgraph Patterns". 2016

[Tho+10] Thoma et al., "Discriminative frequent subgraph mining with optimality guarantees". 2010

[Has+07] Hasan et al. "ORIGAMI: Mining Representative Orthogonal Graph Patterns". 2007

4.2 Beyond Frequent Pattern Analysis and Towards Big Data Scalability

Almost all of the unsupervised representative graph pattern mining algorithms presented above (more precisely [Has+07; ZYL09; NR16; NR18]), are based on the idea to reduce the cardinality of frequent or important subgraph patterns sets, where important patterns are usually highly scored frequent pattern w.r.t. some ranking function. The only exception is [RHS14], which tries to select graphs from the dataset itself as dataset representative, i.e., no subgraph patterns are considered at all. As such, these approaches try to overcome the problem of frequent pattern analysis, where the number of frequent patterns is usually so large, that it is hard to make sense of them. Consequentially, they try to find a subset of the frequent patterns, such that each frequent pattern is similar to one of the selected patterns, in alignment with classical redundancy-focused feature selection methods.

[ZYL09] Zhang, Yang, and Li, "RING: An Integrated Method for Frequent Representative Subgraph Mining". 2009

[RHS14] Ranu, Hoang, and Singh, "Answering Top-k Representative Queries on Graph Databases". 2014

In contrast, the following work has the different goal of finding subgraph patterns that most appropriately describe the dataset graphs. This is similar to the second-order objective of β-representativeness introduced by Hasan et al. [Has+07]. However, their approach only works well for soft distances like the graph edit distance. If specific requirements for the representative relation between the frequent pattern and the dataset graphs are required—e.g., the pattern must be subgraph isomorphic to the dataset graph or the pattern must retain certain properties such as to not brake circles—their sample of maximal frequent subgraph candidate patterns may not contain fitting ones. In other words, a large fraction of dataset graphs may remain unrepresented and some graphs may even be impossible to represent. For this reason the following work will approach the representative problem from a different angle. Instead of defining a fixed frequency or importance threshold and diversifying the resulting patterns, the first step is the definition of a relation, which defines if a subgraph pattern represents a graph dataset. Then, the goal is to find common subgraph patterns, that are able to represent as many graphs as possible, dropping the pattern-to-pattern diversification criterion of the existing approaches. The rationale is, that some similar subgraph patterns that represent different parts of the dataset should actually be part of a structural summary. During visual analysis of datasets, it can be an important piece of information, that a dataset is composed of similar, but distinct parts.

Furthermore, the previous approaches suffer from the problem, to select a support threshold in advance. If the threshold is chosen too high, it is possible to completely ignore important dataset graphs, that do not contain frequent patterns. Contrariwise, if the threshold is chosen too low, the performance and representative set cardinality is impacted. To overcome the high number of candidate patterns some of the existing approaches also resort to maximal patterns, which increases sensitivity w.r.t. to the support threshold.

In terms of performance, existing approaches are challenged by the classical complexity issues of frequent subgraph pattern mining. More precisely, the number of complex subgraph isomorphism tests is high as a result of the size of the pattern space (cf., section 2.6.1.2). To calculate the support of each pattern, a subgraph isomorphism comparison with each dataset graph needs to be performed. The ORIGAMI algorithm [Has+07] tackles the size of the pattern space by random sampling of maximal patterns. However, they do not provide any quality guarantees for their heuristic. Furthermore, all existing approaches rely on exact methods for the support calculation. This limits the scalability to very large datasets. Last, all previous approaches heavily rely on heuristics, when it comes to selecting the representative set w.r.t. their optimization goal.

[Has+07] HASAN ET AL., "ORIGAMI: MINING REPRESENTATIVE ORTHOGONAL GRAPH PATTERNS". 2007

This chapter will provide a new approach that addresses these issues. To the best of my knowledge, the algorithms presented in the following are the first unsupervised subgraph pattern representative mining solutions with guaranteed approximation ratio w.r.t. to the optimization goal, while providing scalability to datasets with billions of graphs.

4.3 Problem Formalization

The goal of this chapter is to mine a cardinality constraint structural summary S for a given graph dataset $\mathcal{G}$. The following problem definition will focus on binary representative relations, such that each pattern in the structural summary either represents a dataset graph or not. Additionally, a representative will always be subgraph-isomorphic to the represented graph.

Definition 4.1 (Subgraph Pattern Coverage Relation). *Given a graph dataset $\mathcal{G}$, a relation $\triangleleft \subseteq \mathcal{G}_{\sqsubseteq} \times \mathcal{G}$ is called subgraph pattern coverage relation, iff*

$$\forall P \in \mathcal{G}_{\sqsubseteq}, G \in \mathcal{G} : P \triangleleft G \Rightarrow P \sqsubseteq G \wedge \forall G \in \mathcal{G}\ \exists P \in \mathcal{G}_{\sqsubseteq} : P \triangleleft G$$

holds and the evaluation of $P \triangleleft G$ does only depend on P, G and a fixed parametrization of $\triangleleft$. It is said, that P covers G, if $P \triangleleft G$.

It is important for statistical properties of the latter presented algorithms, that the evaluation of $\triangleleft$ does not depend on the other patterns is the structural summary S, the composition of the dataset $\mathcal{G}$, or other factors. The notation $\mathcal{G}_{\triangleright P} := \{\!\{G \in \mathcal{G} \mid P \triangleleft G\}\!\}$ will be used for the subset of covered graphs by a pattern P in a dataset $\mathcal{G}$. The notation $\mathcal{G}_{\triangleright S} := \left\{\!\left\{G \in \mathcal{G} \;\middle|\; \bigcup_{P \in S} \mathcal{G}_{\triangleright P}\right\}\!\right\}$ will be used for the subset of covered graphs by a set of patterns S. Following the above definition, the subgraph isomorphism relation itself is a subgraph pattern coverage relation. However, the representative power is limited, since trivial matching patterns, such as the empty graph or single vertex patterns do not provide a meaningful description of a (larger) covered dataset graph. Thus, additional properties are required to provide a good representative power (cf., representative power definition in eq. (3.2) for the StruClus algorithm presented in chapter 3).

Definition 4.2 (Relative Size Thresholded Subgraph Pattern Coverage Relation). *Given a graph dataset $\mathcal{G}$ and a relative size threshold $t \in (0, 1]$, the relation $\triangleleft_t \subseteq \mathcal{G}_{\sqsubseteq} \times \mathcal{G}$ is called relative size thresholded subgraph pattern coverage relation and is defined as:*

$$\forall P \in \mathcal{G}_{\sqsubseteq}, G \in \mathcal{G} : P \triangleleft_t G \Leftrightarrow P \sqsubseteq G \wedge |E(P)| \geq t\ |E(G)|$$

The relative size thresholded subgraph pattern coverage relation will be the default coverage relation in this chapter. However, the algorithms and problems presented here are suited for any subgraph pattern coverage relation. For example, it is possible to incorporate non-linear size thresholds, weights, constraint to retain certain type os substructures (such as cycles), or label similarities into the most general subgraph pattern coverage relation, i.e., definition 4.1.

Given such a binary coverage relation, the subgraph pattern mining problem is defined as follows.

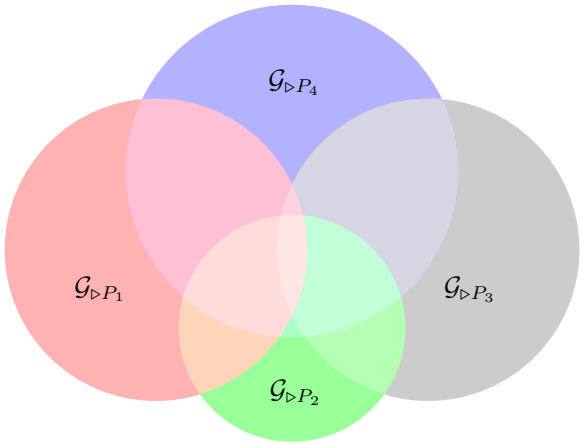


Figure 4.1: Venn diagram of four example graph patterns (P_1 to P_4) covering different portions of a dataset (simplified depiction as a two dimensional plane). Each circle indicates the isolated utility of the associated pattern. Overlapping areas are only counted once for the combined utility.

Problem 4.1 (MAX-CSPC). Constraint Subgraph Pattern Coverage Maximization

Input: *A graph dataset $\mathcal{G}$, a subgraph pattern coverage relation $\triangleleft$, and a positive cardinality constraint k.*

Task: *Solve MAX-SPO with*

$$f_{\mathcal{G},\triangleleft,k}(S) := \begin{cases} |\mathcal{G}_{\triangleright S}| & \text{if } |S| \leq k, \\ -\infty & \text{otherwise.} \end{cases}$$

Figure 4.1 displays a simplified example instance of the problem.

4.4 Problem Properties

Lemma 4.1. *Problem 4.1 (MAX-CSPC) is a special case of problem 2.2 (MAX-Cov).*

Proof. Lemma 4.1 is shown by transforming problem 4.1 (MAX-CSPC) into problem 2.2 (MAX-Cov). MAX-CSPC is defined over problem 2.4 (SPO) with utility function $f_{\mathcal{G}}$ (cf., MAX-CSPC) and can be written as

$$\underset{S \subseteq \mathcal{G}_{\sqsubseteq}}{\arg\max} \; f_{\mathcal{G},\triangleleft,k}(S).$$

Since, the utility $f_{\mathcal{G},\lhd,k}(\emptyset) = 0$ is always greater than the utility of $f_{\mathcal{G},\lhd,k}(S)$ for some S with $|S| > k$ one can add the cardinality constraint to arg max without altering the solution. Subsequently, the second case of $f_{\mathcal{G},\lhd,k}$ can be eliminated, since the arg max cardinality constraint guarantees, that this case is impossible to observe.

$$\underset{\substack{S \subseteq \mathcal{G}_{\sqsubseteq}, \\ |S| \leq k}}{\arg\max} \; f_{\mathcal{G},\lhd}(S) \tag{4.1a}$$

where

$$f_{\mathcal{G},\lhd}(S) := |\mathcal{G}_{\rhd S}| = \left| \bigcup_{P \in S} \mathcal{G}_{\rhd P} \right| \tag{4.1b}$$

Now, it is possible to move the pattern to covered dataset graphs mapping $\mathcal{G}_{\rhd P}$ out of the utility function $f_{\mathcal{G},\lhd}$ by replacing the input set $\mathcal{G}_{\sqsubseteq}$ with $\mathcal{G}_{\sqsubseteq\lhd} := \{\!\{\mathcal{G}_{\rhd P} \mid P \in \mathcal{G}_{\sqsubseteq}\}\!\}$. This results in the task

$$\underset{\substack{S \subseteq \mathcal{G}_{\sqsubseteq\lhd}, \\ |S| \leq k}}{\arg\max} \left| \bigcup_{X \in S} X \right| \quad ,$$

which is identical to MAX-Cov with the input $\mathcal{G}_{\sqsubseteq\lhd}$. □

Strictly speaking, $\mathcal{G}_{\sqsubseteq\lhd}$ is a multiset of multisets, which does not exactly align with the definition of problem 2.2 (MAX-Cov). However, when reducing $\mathcal{G}_{\sqsubseteq\lhd}$ to its support, the solution does not change, since identical selected sets do not add any additional value to the objective. Furthermore, the multisets $\mathcal{G}_{\rhd P}$ for each $P \in \mathcal{G}_{\sqsubseteq}$ can be transformed into indexed sets, where each multiplicity is mapped to a distinct object. The solution of MAX-CSPC can be constructed from the transformed solution by retaining a mapping of each pattern to its matching set in $\mathcal{G}_{\sqsubseteq\lhd}$.

Corollary 4.2. *Problem 4.1 (MAX-CSPC) is a special case of problem 2.1 (MAX-CSSF) and $f_{\mathcal{G},\lhd}$ (cf., eq.* (4.1b)*) is a monotone non-decreasing submodular function.*

Corollary 4.2 follows directly from lemma 4.1, since problem 2.2 (MAX-Cov) has these properties.

Lemma 4.3. *Problem 4.1 (MAX-CSPC) is NP-hard.*

Proof. Lemma 4.1 states, that problem 4.1 (MAX-CSPC) is a special case of the NP-hard problem 2.2 (MAX-Cov). It remains to show, that the family of sets is not restricted to a less complex subset of instances. Let F be an arbitrary family of sets over U and let $\lhd_1$ be a subgraph pattern coverage relation (as defined in definition 4.1) with the additional constraint $|V(P)| = 1$. Lemma 4.3 is proven by constructing a graph dataset $\mathcal{G}$ with a matching graph G_u for each $u \in U$, such that F and $\mathcal{G}_{\sqsubseteq\lhd_1}$ are

isomorphic. One unique vertex label l_S is created for each set $S \in F$. For each $u \in U$ a graph $G_u = (V_u, E_u)$ is added to $\mathcal{G}$. V_u contains one vertex with label l_S for each set S that contains u. Then, only patterns $P \in \mathcal{G}_{\sqsubseteq}$ with a single vertex will have a non-empty set of covered graphs $\mathcal{G}_{\triangleright P}$. Without loss of generality, let l_S be the label of the vertex of some pattern P_S with non-zero utility. Then, $\mathcal{G}_{\triangleright P_S}$ will contain exactly the matched graphs for elements of S, i.e., G_u for each $u \in S$. □

Lemma 4.4. *The utility function $f_{\mathcal{G},\triangleleft}$ is additively decomposable over $\mathcal{G}$.*

Proof. Lemma 4.4 is shown by giving an equivalent form of $f_{\mathcal{G},\triangleleft}$ (cf., eq. (4.1b)) that is a sum over utility functions depending on single dataset graphs. Given the definition of $\mathcal{G}_{\triangleright P}$, a graph $G \in \mathcal{G}$ is contained in the set $\bigcup_{P \in S} \mathcal{G}_{\triangleright P}$ whenever $P \triangleleft G$ is true for at least one pattern $P \in S$. Thus, we can rephrase $f_{\mathcal{G},\triangleleft}$ to sum up all covered graphs:

$$f'_{\mathcal{G},\triangleleft}(S) := \sum_{G \in \mathcal{G}} f_G(S)$$

where

$$f_G(S) := \begin{cases} 1 & \text{if } \exists P \in S : P \triangleleft G, \\ 0 & \text{otherwise.} \end{cases}$$

□

4.5 A Baseline Sequential Greedy Algorithm

Solving problem 4.1 (MAX-CSPC) exactly is unfeasible even for very restricted datasets. This is a result of the computational complexity of problem 2.1 (MAX-CSSF) and the size of the subgraph pattern space (cf., section 2.6.1.2). However, given the greedy approximation algorithm of Nemhauser, Wolsey, and Fisher (cf., section 2.4, algorithm 1) it is straightforward to give an adoption, replacing the recursive formula (cf., algorithm 1, line 4) with eq. (4.2).

$$S_i = S_{i-1} \cup \{ \operatorname*{arg\,max}_{P \in \mathcal{G}_{\sqsubseteq} \setminus S_{i-1}} \Delta_{f_{\mathcal{G},\triangleleft}}(P | S_{i-1}) \} \tag{4.2}$$

$$= S_{i-1} \cup \{ \operatorname*{arg\,max}_{P \in \mathcal{G}_{\sqsubseteq}} f_{\mathcal{G},\triangleleft}(\{P\} \cup S_{i-1}) \} \tag{4.3}$$

Algorithm 5 shows the pseudocode of such an adoption. It successively adds patterns $P_{\max}$ with maximum gain in utility to the representative pattern set S until the cardinality constraint k is reached (cf., lines 3 and 12). This is achieved by looping over all patterns in the subgraph pattern space $\mathcal{G}_{\sqsubseteq}$ in line 5 and determining the pattern $P_{\max}$ in line 7. It should be noted, that the determination of $P_{\max}$ depends on the ordering of $\mathcal{G}_{\sqsubseteq}$ if multiple patterns with maximum utility gain exist. Thus, the algorithm is only deterministic if the ordering of $\mathcal{G}_{\sqsubseteq}$ is fixed. The

Algorithm 5: Sequential Greedy MAX-CSPC

Input: graph dataset $\mathcal{G}$, cardinality constraint $k \in \mathbb{N}^{\geq 1}$, subgraph pattern coverage relation $\triangleleft$

Output: representative pattern set S, utility $\mathsf{utility}_{\max}$

```
1  S ← ∅;
2  utility_max ← 0;
3  while |S| < k do
4      P_max ← null;
5      foreach P ∈ G_⊑ do                    ▶ G_⊑ enumerated by FSM algorithm
6          utility ← f_{G,◁}(S ∪ {P});
7          if utility > utility_max then
8              P_max ← P;
9              utility_max ← utility;
10     if P_max = null then                  ▶ All dataset graphs covered
11         return (S, utility_max);
12     S ← S ∪ {P_max};
13 return (S, utility_max);
```

patterns are computed using a subgraph pattern enumeration method, e.g., a frequent pattern mining algorithm parameterized with an absolute minimum support of 1 (cf., section 2.6.2). Algorithm 5 terminates with $|S| < k$ in line 10 if no additional pattern increases the utility. This situation occurs if and only if some S with $|S| < k$ already covers the complete dataset, i.e., $\mathcal{G}_{\triangleright S} = \mathcal{G}$. Algorithm 5 can be optimized in several ways, which is discussed in the following.

4.5.1 Streaming Pattern Enumeration

The enumeration of $\mathcal{G}_{\sqsubseteq}$ can be performed interleaved with algorithm 5. Precisely, lines 6 to 9 can be integrated as a subroutine in the pattern space exploration. Thus, whenever a pattern is discovered it can be directly determined if this pattern is a newly discovered maximum. This integration has the advantage, that $\mathcal{G}_{\sqsubseteq}$ must not be stored in memory. Instead, the patterns can be processed in a streaming fashion.

4.5.2 Removal of Covered Dataset Graphs

It is not necessary to compute the utility over the complete set $S \cup \{P\}$ in line 6. The overall utility can be written as sum over the utility of S and the marginal gain of P, i.e.,

$$f_{\mathcal{G},\triangleleft}(S \cup \{P\}) = f_{\mathcal{G},\triangleleft}(S) + \Delta_{f_{\mathcal{G},\triangleleft}}(P|S).$$

At this point, the set S and its utility are already fixed. Additionally, S covers a fixed subset $\mathcal{G}_{\triangleright S}$ of the dataset $\mathcal{G}$. Graphs in $\mathcal{G}_{\triangleright S}$ will not contribute to $\Delta_{f_{\mathcal{G},\triangleleft}}(P|S)$, since

each covered graph is only counted once for the overall utility. Furthermore, the set S does not have any utility for the uncovered graphs $\mathcal{G}' := \mathcal{G} \setminus \mathcal{G}_{\triangleright S}$. Thereby it is possible to remember the utility of S and compute $\Delta_{f_{\mathcal{G},\triangleleft}}(P|S)$ on $\mathcal{G}'$ only, i.e.,

$$\Delta_{f_{\mathcal{G},\triangleleft}}(P|S) = \Delta_{f_{\mathcal{G}',\triangleleft}}(P|S) = f_{\mathcal{G}',\triangleleft}(\{P\})$$

holds. Consequentially, it is possible to compute $\mathcal{G}'$ after adding a pattern to S and use $f_{\mathcal{G}',\triangleleft}(\{P\})$ in line 6. This does not only reduce the number of patterns for which $\triangleleft$ must be computed (against each dataset graph) but also reduces the cardinality of the dataset for subsequent iterations.

4.5.3 Support-based Search Space Pruning

It is possible to bound the minimum support threshold $\text{supp}_{\min}$ of the FSM algorithm for a given utility. Since $\triangleleft$ implies the subgraph isomorphism relation (cf., definition 4.1),

$$\forall P \in \mathcal{G}_{\sqsubseteq} : \text{supp}_{\mathcal{G}}(P) \geq f_{\mathcal{G},\triangleleft}(\{P\}) \tag{4.4}$$

holds. Given the optimization from section 4.5.2, it is possible to adjust the $\text{supp}_{\min}$ parameter of the FSM algorithm whenever the utility of the current found maximum increases. The future maximum utility must be larger than the currently found one to fulfill the condition in line 7 of algorithm 5. Thereby, $\text{supp}_{\min}$ can be set to $\Delta_{f_{\mathcal{G},\triangleleft}}(P_{\max}|S) + 1 = \text{utility}_{\max} + 1$ after a change of the currently observed maximal utility. For bottom-up miners (cf., section 2.6.2.1), it is possible to adjust $\text{supp}_{\min}$ during runtime, i.e., it is not necessary to restart the algorithm with the new parameter value. Thus, the pruning power increases for the up to this point undiscovered pattern space.

4.5.4 Transaction List Re-Usage for Utility Computation

An FSM algorithm based on transaction or embedding lists needs to compute the set $\mathcal{G}_{\sqsupseteq P}$ for a given P to calculate the pattern support[1]. Given the optimization from section 4.5.2 one can compute the utility solely over a single pattern P. Since $\mathcal{G}_{\triangleright P} \subseteq \mathcal{G}_{\sqsupseteq P}$ holds, one can reuse the already computed set $\mathcal{G}_{\sqsupseteq P}$ as a filtered input for the computation of the utility over the single pattern P. Additionally, it is possible to avoid the computationally expensive re-evaluation of the subgraph isomorphism relation in the computation of $\mathcal{G}_{\triangleright P}$ since this property is always fulfilled for the input $\mathcal{G}_{\sqsupseteq P}$.

4.5.5 Additional Search Space Pruning for $\triangleleft_t$

For specific subgraph pattern coverage relations, other search space pruning strategies are sometimes possible. For example, the relative size thresholded variant $\triangleleft_t$ can be

[1] For miners that do not completely enumerate $\mathcal{G}_{\sqsupseteq P}$, but abort the calculation when the support threshold is reached (cf., paragraph 2.6.2.1.4), this feature must be disabled.

pruned additionally to the pruning presented in section 4.5.3, since an upper bound for the utility can be calculated for a given multiset of graphs M (e.g., a transaction list of a pattern). The pruning follows the observation that graphs that differ in size more than the size threshold t cannot be covered by the same pattern since a pattern P covering a graph G cannot be larger than G. Thereby, given the ordered sizes $s = (s_1, \ldots, s_n)$ of the supporting graphs of a pattern P, one can compute the maximal with window $(s_i, \ldots, s_j)$ with $1 \leq i \leq j \leq n$, such that $s_i \geq ts_j$. Then, $f_{max} := j - i + 1$ bounds the utility $f_{M,\triangleleft}$ for all superpatterns of P. If $\mathsf{utility}_{\max}$ is already larger than or equal to f_{max}, a bottom-up pattern space exploration can be pruned at P.

4.5.6 The Optimized Algorithm

Algorithm 6: Optimized Sequential Greedy MAX-CSPC

Input: graph dataset $\mathcal{G}$, cardinality constraint $k \in \mathbb{N}^{\geq 1}$, subgraph pattern coverage relation $\triangleleft$, FSM algorithms fsm, pruning strategy $\texttt{pruning}_{\triangleleft}$

Output: representative pattern set S, utility of S

```
1  S ← ∅;
2  utility_sum ← 0;
3  G' ← G;
4  while |S| < k and G' ≠ ∅ do
5      P_max ← null;
6      utility_max ← 0;
7      fsm.init(G', pruning_◁);
8      (P, G'_⊒P) ← fsm.next(1);
9      while P ≠ null do
10         utility ← f_{G'_⊒P,◁}({P});
11         if utility > utility_max then
12             P_max ← P;
13             utility_max ← utility;
14         (P, G'_⊒P) ← fsm.next(utility_max + 1);
15     S ← S ∪ {P_max};
16     utility_sum ← utility_sum + utility_max;
17     G' ← G' \ G'_▷P_max;
18 return (S, utility_sum);
```

Algorithm 6 integrates the above discussed optimizations. The streaming pattern enumeration (cf., section 4.5.1) is realized by an iterator-like interface for the FSM algorithm. It provides two operations, `init` and `next`. The operation `init` does start a new pattern space exploration on a given dataset and a subgraph pattern coverage relation-specific pruning strategy (e.g., as presented in section 4.5.5; possibly

no-op). The operation `next` is parameterized with the minimum support threshold to implement the support-based pruning optimization as given in section 4.5.3. It returns a tuple of the next frequent pattern in the enumeration sequence together with the associated supporting graphs in the dataset (as needed for the transaction list re-usage presented in section 4.5.4). A null pattern is returned, if the enumeration sequence reached its end. With the help of this interface, the foreach loop in line 5 of algorithm 5 is replaced by the while loop in line 9 and two calls of `next` in lines 8 and 14 in algorithm 6. The parameter $\text{utility}_{\text{max}} + 1$ in line 14 also implements the support-based pruning (cf., section 4.5.3).

The removal of the covered dataset graphs as presented in section 4.5.2 is performed in line 17, while $\mathcal{G}'$ represents the filtered dataset. The `init` operation of the FSM algorithm in line 7 is parameterized with $\mathcal{G}'$. The computation of the utility is now performed on $\mathcal{G}'$ and the single pattern P in line 10. As a consequence, the variable $\text{utility}_{\text{max}}$ of algorithm 5 is now split into two variables $\text{utility}_{\text{sum}}$ and $\text{utility}_{\text{max}}$. The former stores the utility of S and the latter stores the marginal gain of P_{max} w.r.t. S, i.e., $\Delta_{f_{\mathcal{G},\triangleleft}}(P|S)$. After computing P_{max} in the loop starting at line 9 $\text{utility}_{\text{max}}$ is added to $\text{utility}_{\text{sum}}$, since P_{max} becomes a member of S. The check in line 10 of algorithm 5 is replaced with a check on $\emptyset$ for $\mathcal{G}'$ in the main loop (line 4). This is correct since it is always possible to find a pattern that covers at least a single dataset graph (e.g., the pattern that contains the dataset graph itself; cf., definition 4.1) as long $\mathcal{G}'$ is not empty.

The transaction list re-usage as presented in section 4.5.4 is realized by computing the marginal utility gain of a pattern P in line 10 over $\mathcal{G}'_{\sqsupseteq P}$ instead of using $\mathcal{G}'$.

4.5.7 Analysis

Algorithm 6 will be subject to analysis in this section w.r.t. computational complexity and approximation quality.

4.5.7.1 Computational Complexity

Since algorithm 6 includes the enumeration of frequent patterns, the complexity of algorithm 6 is not less than the complexity of the FSM algorithm (cf., section 2.6.2.3). Consequentially, a subexponential worst-case running time cannot be expected for algorithm 6 unless P = NP.

Nevertheless, the FSM problem is solvable sufficiently fast for many kinds of datasets (especially datasets of small graphs). It is shown in the following, that algorithm 6 scales linearly with k and that each iteration of the main loop (lines 5 to 17) only adds a constant factor to the running time of the FSM algorithm as long as the relation $P \triangleleft G$ can be evaluated in constant time given that the subgraph isomorphism relation $P \sqsubseteq G$ is known (assumption $\triangleleft$).

The main loop is repeated $\min(k, |\mathcal{G}|)$ times in the worst case, since each iteration adds a pattern to S until $|S| = k$. Additionally, $|S|$ is bound by $|\mathcal{G}|$, since each pattern represents at least one graph in the dataset. Let's assume, that $\texttt{FSM}_{\mathcal{G},\text{supp}_{\text{min}}}$ is the running time for the FSM algorithm given the graph dataset $\mathcal{G}$ and the minimum

support threshold $\text{supp}_{\min}$. Furthermore, let $\mathcal{G}_i$ with $1 \leq i \leq k$ and $\mathcal{G}_1 = \mathcal{G}$ be the graph dataset in iteration i of the main loop (line 4). As long as lines 5 to 17 only add a constant factor to the running time of the FSM algorithm, the worst-case running time of algorithm 6 is in $\mathcal{O}(\sum_{i=1}^{\min(k,|\mathcal{G}|)} \texttt{FSM}_{\mathcal{G}_i,1})$. Since the running time of the FSM algorithm increases monotonically for lower values of $\text{supp}_{\min}$, it is safe to assume a minimum support of 1 in the worst case.

It has yet to be shown, that lines 5 to 17 only add a constant factor to the running time of the FSM algorithm. It is clear, that lines 5 to 6, lines 11 to 13, and lines 12 to 13 are constant-time operations. Each line involving an FSM operation is part of the FSM algorithm and does not contribute to the overhead. Under the above mentioned assumption for $\lhd$ each relation $P \lhd G$ for $G \in \mathcal{G}'_{\sqsupseteq P}$ in line 10 can be evaluated in constant time since $P \sqsubseteq G$ is true by the definition of $\mathcal{G}'_{\sqsupseteq P}$. Thereby, it is possible to relate the costs of each evaluation to a distinct graph in $\mathcal{G}'_{\sqsupseteq P}$. Since each graph is associated with a cost of $\Omega(1)$ in the operation `fsm.next`, line 10 only adds a constant factor to the running time of the FSM algorithm. The set minus operation in line 17 requires a superlinear running time w.r.t. to $\mathcal{G}'$ in the written form. With hashing, it is possible to solve the set minus in expected linear time, but the worst case is still quadratic. However, it is possible to avoid the set minus operation by using the alternative formula $\mathcal{G}' \leftarrow \mathcal{G}'_{\not\sqsupseteq P} \cup \mathcal{G}'_{\sqsupseteq P_{\not\rhd P_{\max}}}$ instead. The multiset $\mathcal{G}'_{\not\sqsupseteq P}$ of graphs that do not support a pattern P can be returned by `fsm.next` with a constant factor overhead, since `fsm.next` already evaluates the subgraph isomorphism relation for each graph in $\mathcal{G}'$. The multiset $\mathcal{G}'_{\sqsupseteq P_{\not\rhd P_{\max}}}$ represents the uncovered graphs in the multiset of supported graphs for a pattern P. It can be computed alongside the computation of the utility in line 10 and the costs can be related to graphs in $\mathcal{G}'_{\sqsupseteq P}$ as shown for line 10.

To conclude, the running time of algorithm 6 is in $\mathcal{O}(\sum_{i=1}^{\min(k,|\mathcal{G}|)} \texttt{FSM}_{\mathcal{G}_i,1})$. Given $\texttt{FSM}_{\max} := \max_{i=1}^{\min(k,|\mathcal{G}|)} \texttt{FSM}_{\mathcal{G}_i,1}$, this can be shortened to $\mathcal{O}(\min(k, |\mathcal{G}|)\ \texttt{FSM}_{\max})$.

4.5.7.2 Approximation Quality

Since algorithm 6 is a straightforward adoption of the greedy algorithm presented by Nemhauser, Wolsey, and Fisher it shares the approximation quality, i.e., an approximation ratio of $1 - (1 - \frac{1}{k})^k > (1 - \frac{1}{e}) \approx 0.63$ in the worst case. As mentioned in section 2.4, this approximation ratio cannot be improved until $P = NP$.

4.5.8 Implementation Details

The algorithm was implemented in Java using gSpan as FSM algorithm. The gSpan algorithm uses the slightly modified code representation given by [Bor07] as described in paragraph 2.6.2.2.1. The implementation is shared-memory parallelized as described for the distributed variant of the algorithm (cf., algorithm 11). The details of the parallelization are omitted at this point to avoid repetitions.

[Bor07] Borgelt, "Canonical Forms for Frequent Graph Mining". 2007

Subgraph isomorphism tests are implemented using the backtracking algorithm and CT-Index fingerprint pre-filtering algorithms from [KKM11]. A more detailed description can be found in the implementation details of StruClus (cf., section 3.2.8.1).

[KKM11] Klein, Kriege, and Mutzel, "CT-Index: Fingerprint-based Graph Indexing Combining Cycles and Trees". 2011

4.5.9 Experimental Evaluation

The goal of this evaluation is to see how the algorithm behaves w.r.t. different parameters and dataset properties. Additionally, the evaluation results serve as a baseline for the later evaluation of the distributed variant (cf., section 4.6.10). Each experiment was repeated 20 times and run on 10 cores in parallel.

4.5.9.1 Computational Environment

4.5.9.1.1 Hardware Computational experiments are performed on the LiDO3[2] cluster and the `cstd01` nodes. Each `cstd01` node is a dual-socket NUMA system with two Intel Xeon E5-2640v4 processors, 20 cores (i.e., 10 per NUMA domain, hyperthreading cores are disabled), and 64 GiB of RAM (i.e., 32 GiB per NUMA domain). The nodes have 2 TiB local storage (Seagate Constellation ES.3 HDD, 7200rpm, SATA 6) and access to a network-attached distributed file system (BeeGFS[3]). The `cstd01` nodes are connected over an Infiniband QDR Interconnect (40 Gbit/s) with a maximum blocking ratio of 1 : 3.

4.5.9.1.2 Software LiDO3 nodes run on CentOS Linux 7 [4] and are managed by the SLUM[5] workload manager. Java is installed in Version 1.8 (Oracle Java HotSpot(TM) 64-Bit Server VM).

4.5.9.1.3 Task Assignment and Environmental Configuration The processor and memory affinity of each computation is restricted to a single NUMA domain. Thus, two non-interfering computations can be launched per `cstd01` node, each having access to 10 cores and 32 GiB of memory. The minimum and maximum heap size of a Java VM instance is set to 25 GiB (`-Xmx` and `-Xms` parameter), such that the heap size plus the overhead of the Java VM always fits in the memory of a single NUMA domain without causing any swapping. Setting the minimum heap size avoids, that the Java VM slowly increases the heap size during runtime, which in turn can cause some avoidable garbage collection overhead. Java VMs are configured to use the parallel garbage collector for the young memory generation (parameter `-XX:+UseParallelGC`).

4.5.9.2 Datasets

Four real-world molecular dataset with different characteristics are used for experimental evaluation: ChemDB[6] [Che+07] (Version 2015-03-31), ChEMBL[7] [Ben+13] (Version 22.1), CH*I*PMUNK Heterocycle CoMol [Hum+18] (publication version, drug-like $500\,\mathrm{g\,mol^{-1}}$ variant), and Protein Interaction [Stö+19]. All datasets were

[Che+07] Chen et al., "ChemDB update—full-text search and virtual chemical space". 2007

[Ben+13] Bento et al., "The ChEMBL bioactivity database: an update". 2013

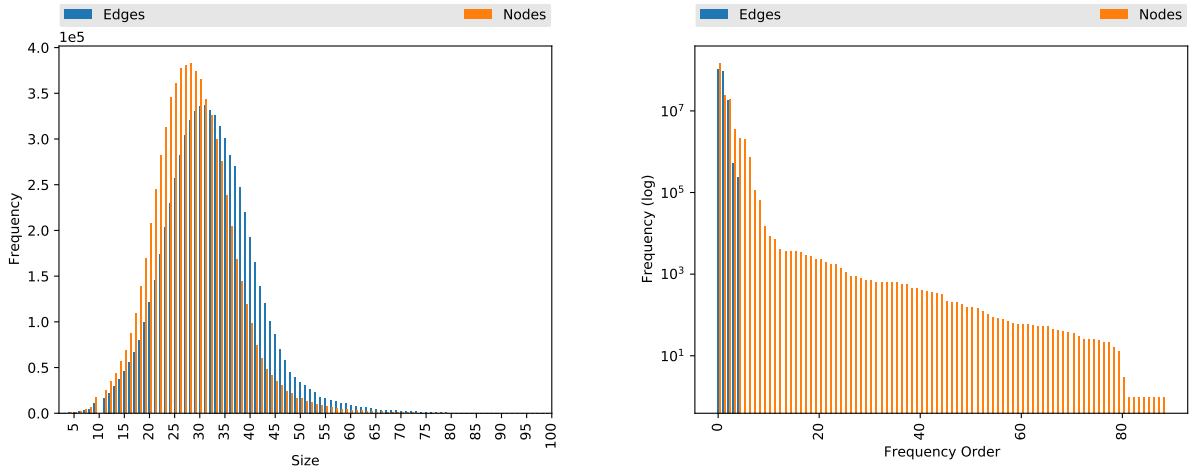
[Hum+18] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

[Stö+19] Stöcker et al., "Protein Complex Similarity Based on Weisfeiler-Lehman Labeling". 2019

[2] `https://www.lido.tu-dortmund.de/cms/de/LiDO3/index.html`
[3] `https://www.beegfs.io`
[4] `https://www.centos.org/`
[5] `https://slurm.schedmd.com/`
[6] `http://cdb.ics.uci.edu/`
[7] https://www.ebi.ac.uk/chembl/

Table 4.1: Descriptive statistics of the evaluation datasets.

		$\lvert V\rvert$				$\lvert E\rvert$					
Dataset $\mathcal{G}$	$\lvert\mathcal{G}\rvert$	Min.	Max.	Avg.	Med.	Min.	Max.	Avg.	Med.	$\lvert\mathcal{L}_V\rvert$	$\lvert\mathcal{L}_E\rvert$
ChemDB	7 100 106	2	435	28.66	28	1	496	31.12	31	89	5
ChEMBL	1 678 393	2	876	29.79	27	1	894	32.18	30	34	5
CHIPMUNK Heterocycle CoMol Subset	4 158 909	4	39	27.56	28	4	46	29.71	30	62	6
Protein Interaction	2 242 972	4	126	5.17	4	3	125	4.17	3	717	1



(a) Size distribution of the ChemDB dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the ChemDB dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

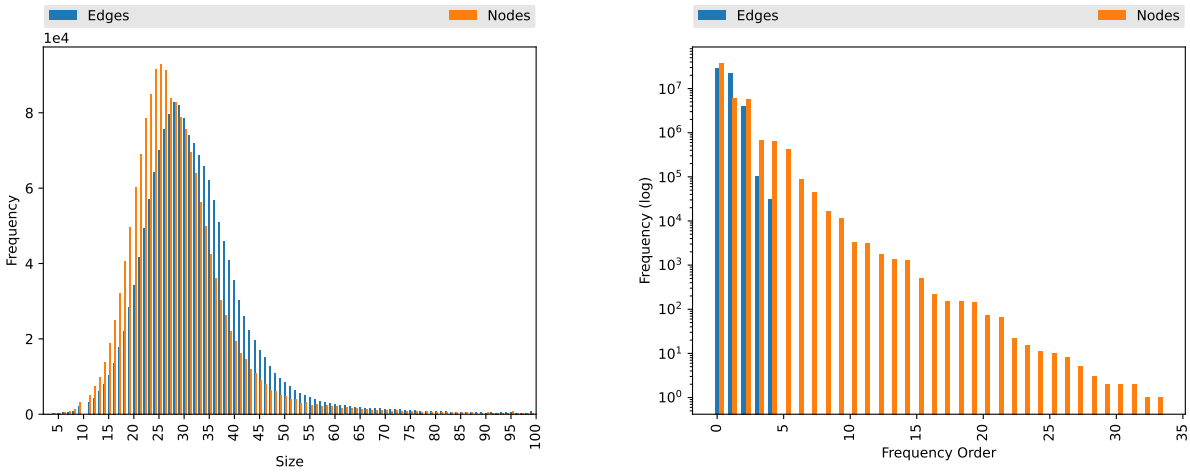
Figure 4.2: Frequency distributions of the ChemDB dataset

pre-processed to remove duplicates and detect aromatic bonds (using CDKHueckelAromaticityDetector[8] of the Chemistry Development Kit [Wil+17]). Table 4.1 shows an overview of some basic dataset statistics regarding dataset cardinality, graph sizes, and label counts.

The ChemDB library is a collection of over 100 different sources, especially commercially available molecules. It is targeting small molecules, such as building blocks for further synthesization. ChemDB was already used in the original StruClus evaluation as a demonstration of the scalability of the algorithm to large-scale datasets. It is the largest dataset used for the non-distributed experiments. Figure 4.2 shows the size and label distributions of the dataset.

[8] `https://cdk.github.io/cdk/1.4/docs/api/index.html?org/openscience/cdk/aromaticity/CDKHueckelAromaticityDetector.html`

[Wil+17] Willighagen et al., "The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching". 2017



(a) Size distribution of the ChEMBL dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the ChEMBL dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

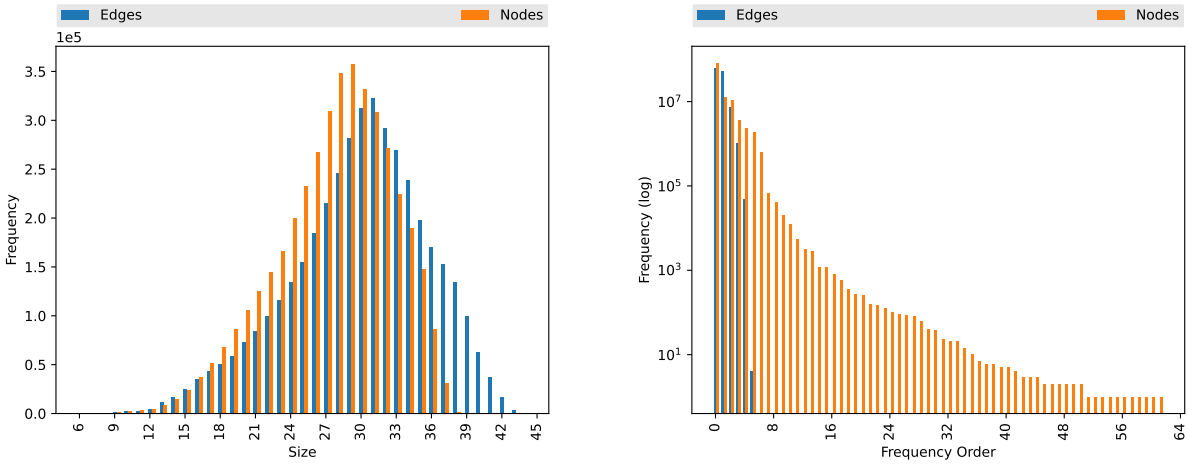
Figure 4.3: Frequency distributions of the ChEMBL dataset

The ChEMBL database is a manually curated public chemical library with a special focus on drug-likeliness. In comparison with ChemDB is therefore a more restricted dataset with different characteristics, e.g., containing fewer chemical elements (i.e., vertex labels). It is considered to have a broad coverage of existing chemical knowledge in the context of drug discovery. The dataset was also used in the novelty analysis of CHIPMUNK (cf., section 3.2.9.9). Figure 4.3 shows the size and label distributions of the dataset.

The molecular de-nove library CHIPMUNK was already discussed in section 3.2.9.9 since the StruClus algorithm was applied for a novelty analysis. In the following experimental evaluation, the CHIPMUNK Heterocycle CoMol subset is selected since it fits in the memory of a single NUMA domain of a cluster node. The used drug-like $500\,\mathrm{g\,mol^{-1}}$ variant of the library shows smaller maximum graph sizes compared to the non-synthetic datasets presented above. Figure 4.4 shows the size and label distributions of the dataset.

[Stö+19] Stöcker et al. "Protein Complex Similarity Based on Weisfeiler-Lehman Labeling". 2019

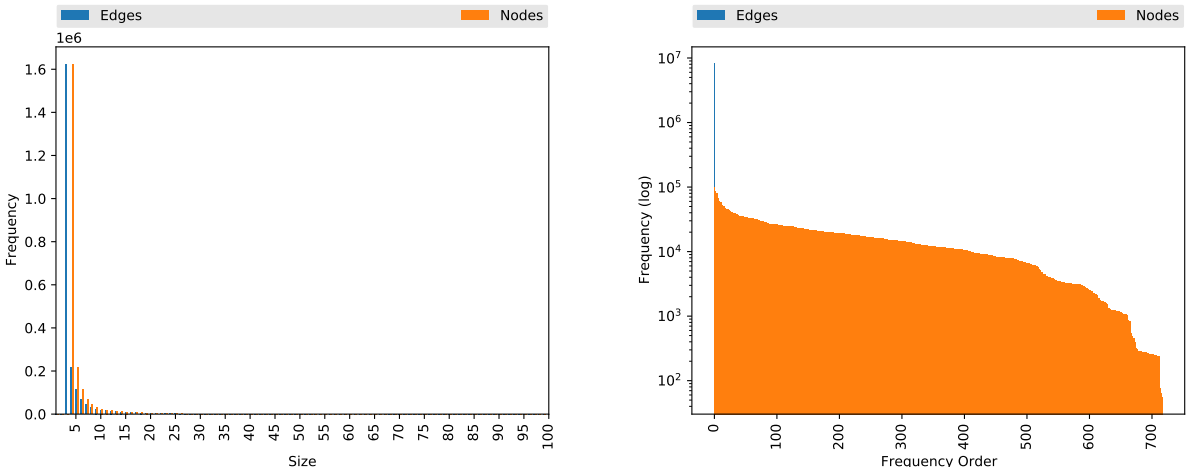
The Protein Interaction dataset is a non-molecular graph dataset presented by Stöcker et al. [Stö+19]. It contains simulated protein-interaction networks, where each vertex is labeled with a descriptor of a protein and the (unlabeled) edges represent physical interactions between these proteins. In comparison with the molecular dataset, the number of distinct vertex labels is very high and the graph sizes are very small with 5.17 vertices 54.17 edges on average. Figure 4.5 shows the size and label distributions of the dataset.



(a) Size distribution of the CH*I*PMUNK Heterocycle CoMol dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the CH*I*PMUNK Heterocycle CoMol dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

Figure 4.4: Frequency distributions of the CH*I*PMUNK Heterocycle CoMol dataset



(a) Size distribution of the Protein Interaction dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the Protein Interaction dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

Figure 4.5: Frequency distributions of the Protein Interaction dataset

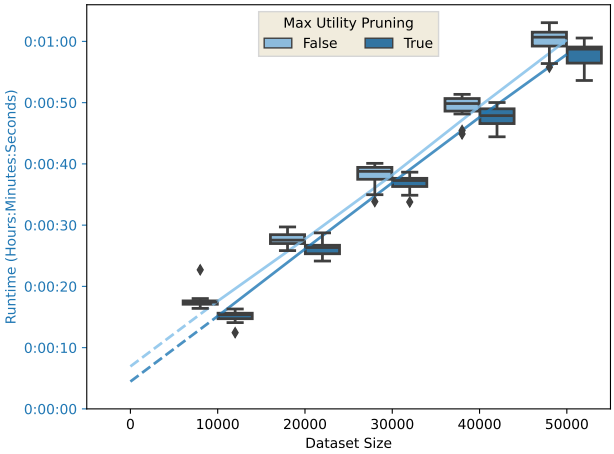
4.5.9.3 Scaling with Isolated Parameters

Before an evaluation w.r.t. different datasets and different iterations of the algorithm is performed in section 4.5.9.4, this section will focus on the running time (wall clock time) effects of algorithm parameters and dataset properties. All experiments are performed on a sample of the ChemDB dataset filtered to a maximum edge count to perform the experiments in an appropriate time span. If the parameters themselves are not suspect to evaluation, the sample cardinality is set to 10 000 and the filtering is set to a maximal edge count of 30. The relative size thresholded subgraph pattern coverage relation was parameterized with the thresholds $t \in \{0.3, 0.5, 0.7\}$.

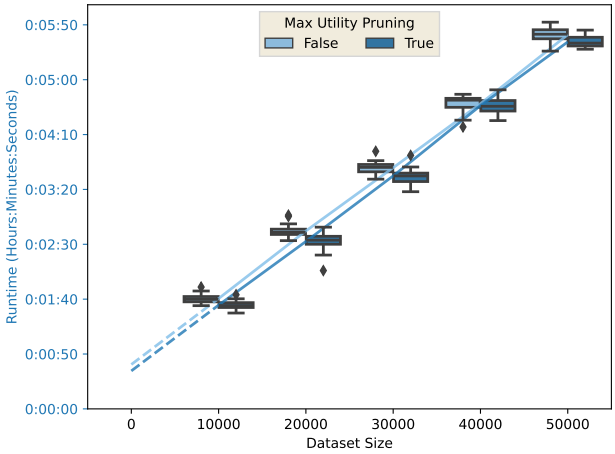
Figure 4.6 focuses on the scaling of the algorithm given different dataset sizes. The random sample of size 10 000 was multiplied to create datasets of sizes 20 000 to 50 000. Multiplying the sample has the advantage, that random effects regarding the pattern space can be ruled out. More precisely, all experiments will be performed on the exact same (pruned) pattern space, since the utility-based pruning bound increases by the same factor as the support values of individual patterns. Figure 4.6 shows an almost perfect linear scaling with the dataset size for the lower values of the relative size threshold parameter t. For $t = 0.7$ there exist some outliers with a high running time, that cause the average running time (lineplot) to be above the 75th percentile of the values. The dashed lines in the plots represent the extension of the average values to a dataset size of 0. While a dataset of size 0 is a meaningless input, the virtual value gives insight into the running time for pattern space exploration and canonization without taking the support calculation into account. Even for the given relatively small dataset sizes, the majority of time is spent performing the subgraph isomorphism tests in the settings $t \in \{0.3, 0.5\}$ and only a small fraction of the time is spent on the canonization and extension of patterns. For $t = 0.7$ the overhead is actually larger than the time for subgraph isomorphism tests for small sample sizes. These observations are also reflected by the implementation of gSpan, which performs the canonization-based search space pruning before the support-based pruning to perform best in most cases and scale well to large dataset sizes.

It can also be seen in fig. 4.6 that the absolute overhead of the pattern space exploration increases with larger values of t. This is a consequence of the fact, that the number of represented graphs (i.e., the utility) by a pattern is monotonically decreasing with larger values of t, i.e., $\forall \mathcal{G}, \forall P \in \mathcal{G}_{\sqsubseteq}, \forall t < t' : \mathcal{G}_{\triangleright_{t'} P} \subseteq \mathcal{G}_{\triangleright_t P}$, which in turn causes the pruning of the search space to be less strict. The effect can also be seen in fig. 4.7, which shows the relation between the relative size threshold and the running time. Additionally, it presents the utility and solution pattern edge count. For larger values of t the running time increases sharply, the utility is decreasing as discussed above, and the edge count of the final pattern increases.

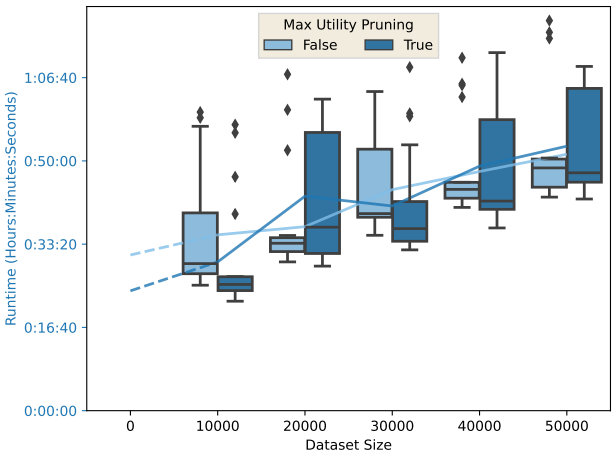
Increasing the maximum edge count of the graphs in the sample has a strong effect on the running time. Figure 4.8 show the running time in relation to a maximum edge count of 20, 30, and 40. The relative amount of graphs in the range of sizes 20 to 30 and 30 to 40 can be extracted from the size distribution of the ChemDB dataset given in fig. 4.2a. This superlinear increase in running time can be attributed to the high complexity of the subgraph isomorphism tests and the canonization test as well



(a) Relative Size Threshold $t = 0.3$



(b) Relative Size Threshold $t = 0.5$



(c) Relative Size Threshold $t = 0.7$

Figure 4.6: Scaling of the running time of algorithm 6 with different dataset sizes given random samples of the ChemDB dataset. The samples of size 20 000 to 50 000 are constructed by multiplication of the 10 000 sample. The dataset is filtered to graphs with an edge count of 30 or less.

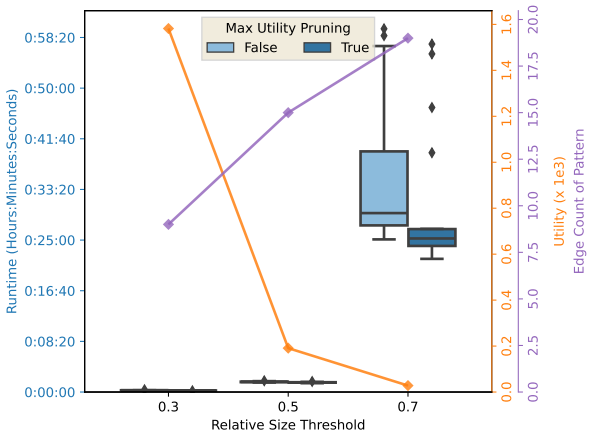


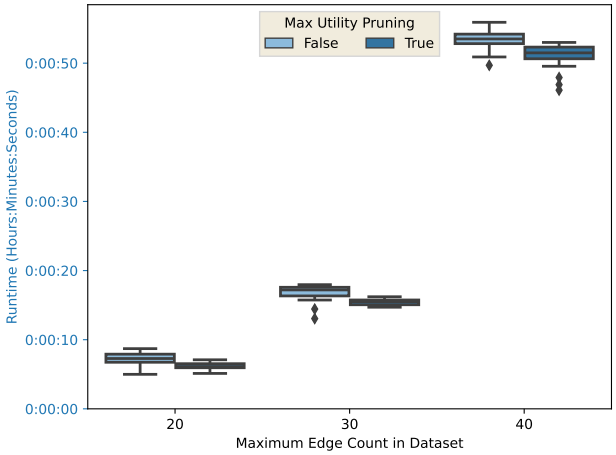
Figure 4.7: Scaling of the running time of algorithm 6 with different values of the relative size threshold t given a random sample of size 10 000 of ChemDB dataset. The dataset is filtered to graphs with an edge count of 30 or less.

as the combinatorial explosion of the graph pattern space w.r.t. the graph size (cf., section 2.6.1.2). The contrast increases for larger values of t, since its causes less strict search space pruning. Thus, the pattern exploration extends to even higher ranks in the pattern space.

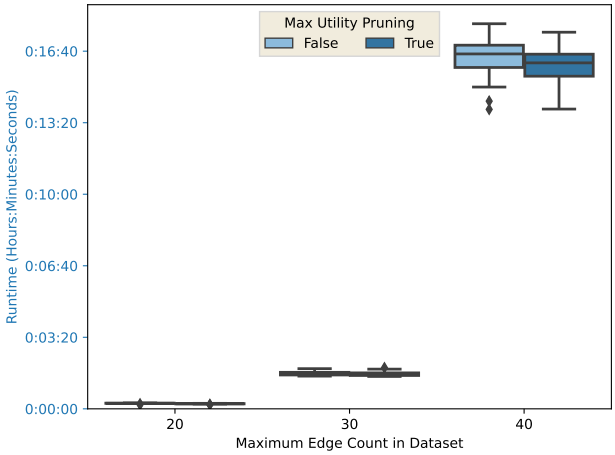
As discussed in section 4.5.5, a maximum utility search space pruning can be applied to $\lhd_t$. Figures 4.6 to 4.8 differentiate between the enabled and disabled state of this pruning strategy. Although the pruning strategy has some overhead to calculate f_{max}, the benefit of the pruning outweighs the overhead. While the speedup is only a few percent on average, the speedup is stable w.r.t. to different settings. Only fig. 4.6c shows some cases where the additional pruning leads to a higher running time on average. However, the variance in this plot is high, such that the larger dataset sizes even outperform smaller sizes in some cases. Thus, the interpretation of the small differences in this setting is not meaningful. As a consequence of the stable behavior, the maximum coverage pruning is enabled for all future experiments.

4.5.9.4 Performance and Key Statistics for Individual Solution Elements

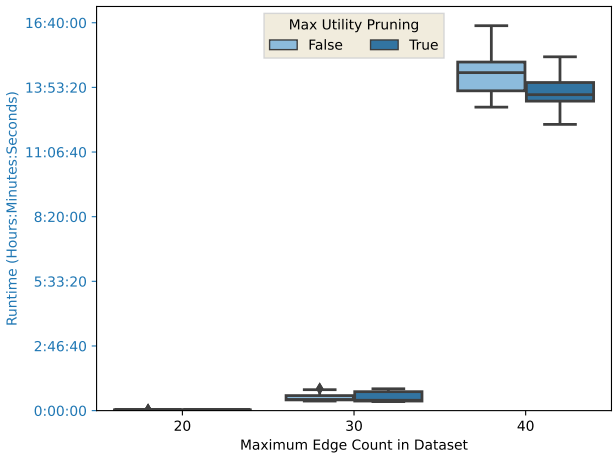
It is evaluated how the performance evolves over the different iterations of the algorithm with increasing solution size in the following. More precisely, the wall clock time, pattern utility, and pattern size will be evaluated for each individual addition to the solution set S, i.e., each iteration of the main loop (line 4) of algorithm 6. The relative size thresholded subgraph pattern coverage relation was parameterized with the thresholds $t \in \{0.3, 0.5, 0.7\}$. With the exception of the Protein Interaction dataset (which has relatively small graphs), the datasets are filtered to graphs with a maximum edge count of 30, since the running time of the algorithm becomes infeasible for larger



(a) Relative Size Threshold $t = 0.3$



(b) Relative Size Threshold $t = 0.5$



(c) Relative Size Threshold $t = 0.7$

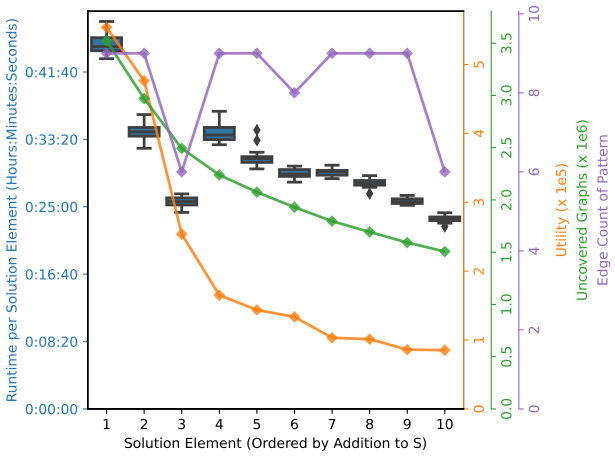
Figure 4.8: Scaling of the running time of algorithm 6 with different graph sizes given a random sample of size 10 000 of ChemDB dataset. The dataset is filtered to graphs with an edge count of 20, 30, and 40 or less.

dataset graphs in combination with larger settings of t, otherwise. Computation was aborted if the total running time (i.e., the summed running time over all performed iterations) exceeds two days. The solution set cardinality $|S|$ is limited to a maximum of ten. Fewer iterations are given in the case of running time abortions.

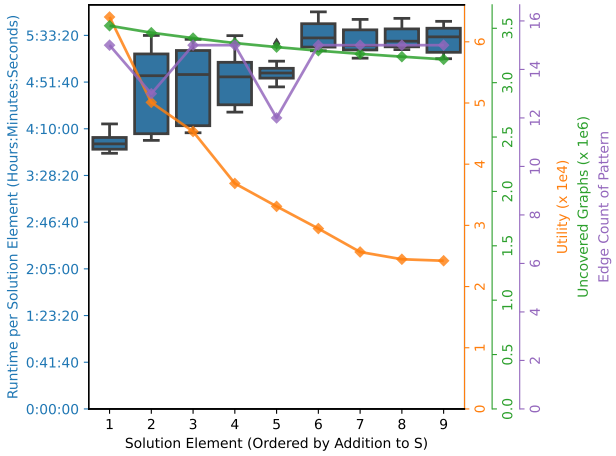
Figures 4.9 to 4.12 show the results for the ChemDB, ChEMBL, CH*I*PMUNK Heterocycle CoMol, and Protein Interaction datasets. The filtered datasets (max edge count 30) have a cardinality of 3 523 595 (ChemDB), 909 321 (ChEMBL), and 1 687 491 (CH*I*PMUNK Heterocycle CoMol), respectively. Since the covered graphs are removed after each iteration of the algorithm, the utility in the plots represents the marginal gain (cf., definition 2.2) w.r.t. the solution set S of the previous iteration. As a result of the submodular nature of the utility function (cf., eq. (2.2)) the marginal gain is monotonically decreasing with each iteration. The uncovered dataset graphs (cf., $\mathcal{G}'$ in algorithm 6) show the other side of the same statistic, that is the dataset size minus the summed utilities of the previous iterations.

Overall, the running times for these filtered datasets are ranging from a few minutes to more than one day per iteration. This confirms the high sensitivity of the running times w.r.t. some algorithm and dataset parameters as discussed in section 4.5.9.3. The running time is influenced by the decreasing dataset size, which is clearly visible in the setting $t = 0.3$ for the datasets ChemDB and CH*I*PMUNK Heterocycle CoMol (figs. 4.9a and 4.11a). This effect is less observable for higher values of t, the ChEMBL dataset, and especially the Protein Interaction dataset. Most likely, other factors of the running time are more relevant in the latter settings. For example, higher values of t cause lower utilities (as shown in the previous section in fig. 4.7), which causes the differences in the dataset sizes to decrease. Additionally, as shown in fig. 4.6 and discussed in section 4.5.9.3, the overhead for the pattern space exploration becomes more relevant for higher values of t. The size of the solution pattern seems to have a small impact on the performance in some situations. This is observable in e.g., fig. 4.9a at iterations 2 to 4, fig. 4.10a at iterations 5 to 7, and fig. 4.10c. However, the effect is sometimes overshadowed by other parameters, such as in fig. 4.10a at iteration 4, where a large drop in utility—w.r.t. iteration 3—-most likely caused a relatively high running time for a small pattern. Furthermore, the Protein Interaction dataset has a very large number of labels in combination with very small graphs. This results in a smaller fraction of the dataset to be covered, fast subgraph isomorphism tests, and shallow pattern space. As a result of the small coverage values, the overall running time is less influenced by the decreasing dataset size, similar to settings with high values for t.

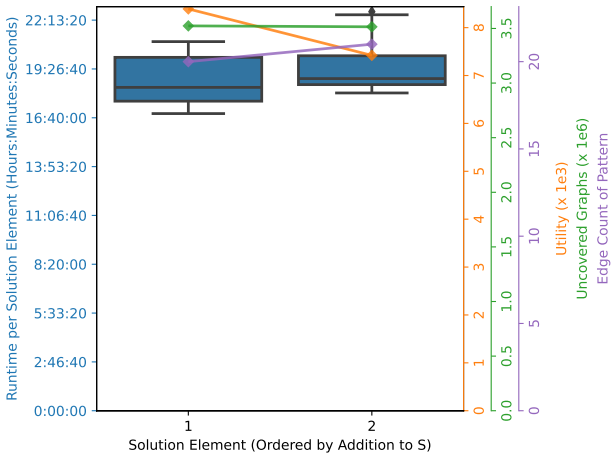
The plots also reveal different characteristics of the datasets themselves. For example, the CH*I*PMUNK Heterocycle CoMol dataset seems to be more homogeneous than the other datasets. With a solution set of cardinality ten and $t = 0.3$ only 18% of the CH*I*PMUNK Heterocycle CoMol remain unrepresented, while 43% and 48% remain unrepresented for the ChemDB and ChEMBL datasets. Again, the Protein Interaction dataset is very different with roughly 96% unrepresented graphs. This observation is most likely a result of the way the datasets are composed. While ChEMBL and ChemDB are broad collections of known molecules, the CH*I*PMUNK datasets are synthesized using a fixed set of virtual reaction types. These reaction



(a) Relative Size Threshold $t = 0.3$

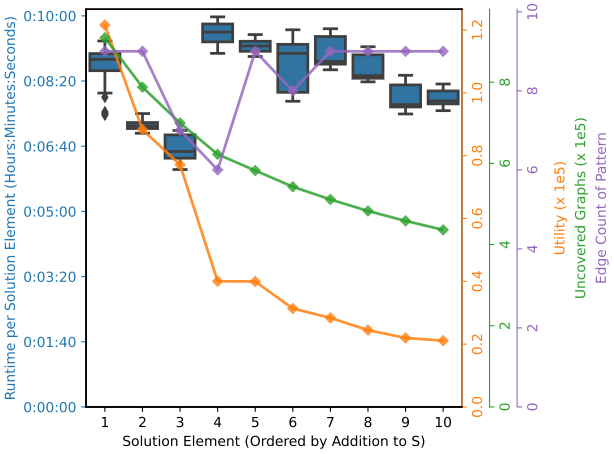


(b) Relative Size Threshold $t = 0.5$

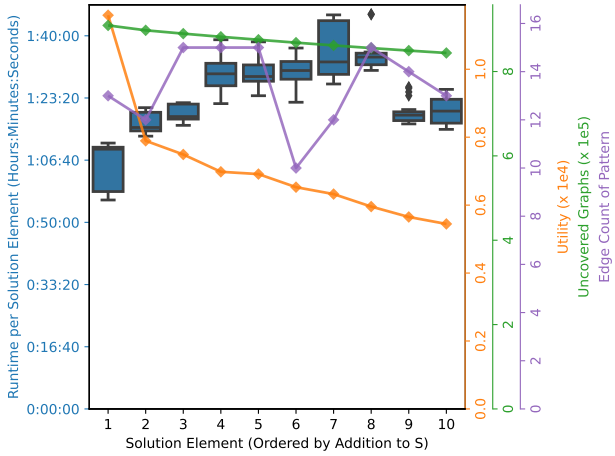


(c) Relative Size Threshold $t = 0.7$

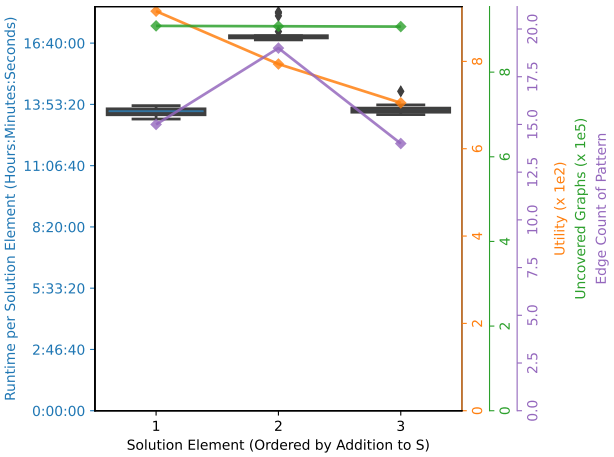
Figure 4.9: Running Time, utility, uncovered graphs, and edge count of $P_{\max}$ for each iteration (i.e., addition to S) of algorithm 6 on the ChemDB dataset. The dataset is filtered to graphs with an edge count of 30 or less. Execution was aborted for each repeat, when the accumulated running time of the individual iterations exceeds 2 days.



(a) Relative Size Threshold $t = 0.3$

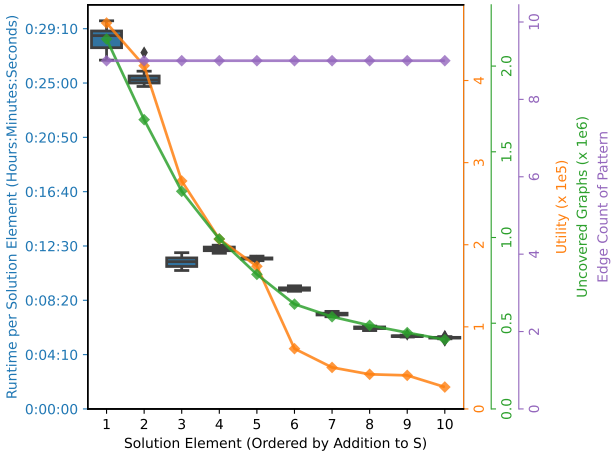


(b) Relative Size Threshold $t = 0.5$

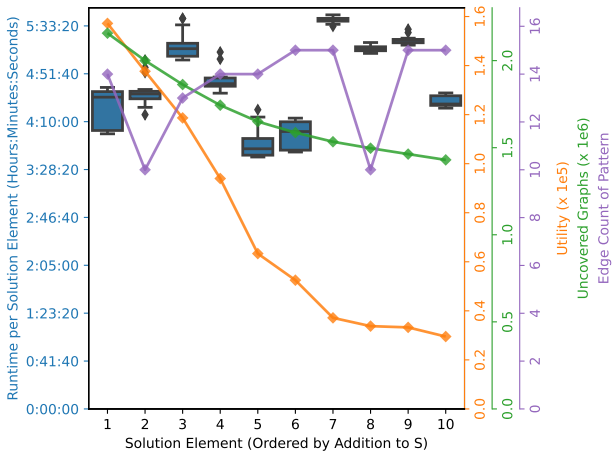


(c) Relative Size Threshold $t = 0.7$

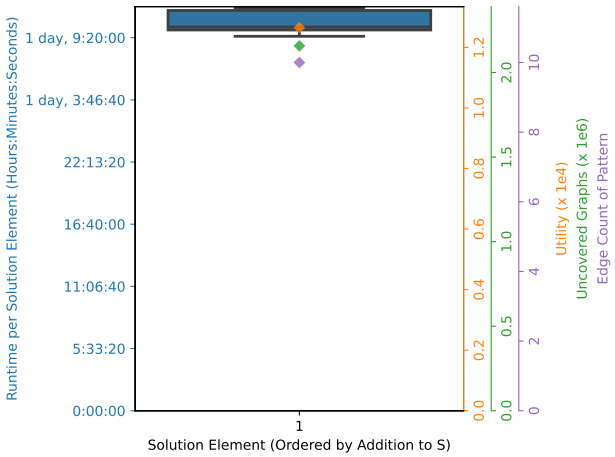
Figure 4.10: Running Time, utility, uncovered graphs, and edge count of $P_{\max}$ for each iteration (i.e., addition to S) of algorithm 6 on the ChEMBL dataset. The dataset is filtered to graphs with an edge count of 30 or less. Execution was aborted for each repeat, when the accumulated running time of the individual iterations exceeds 2 days.



(a) Relative Size Threshold $t = 0.3$

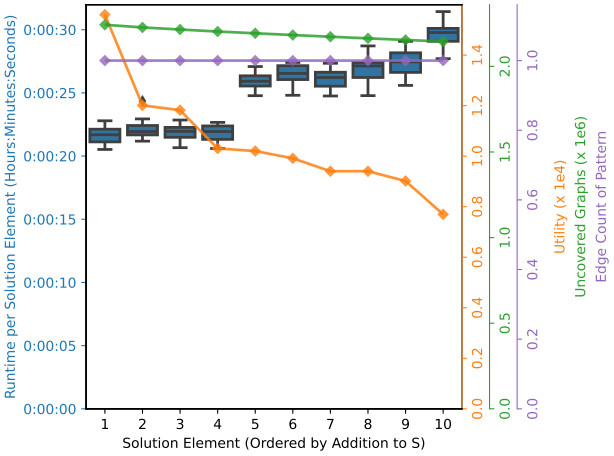


(b) Relative Size Threshold $t = 0.5$

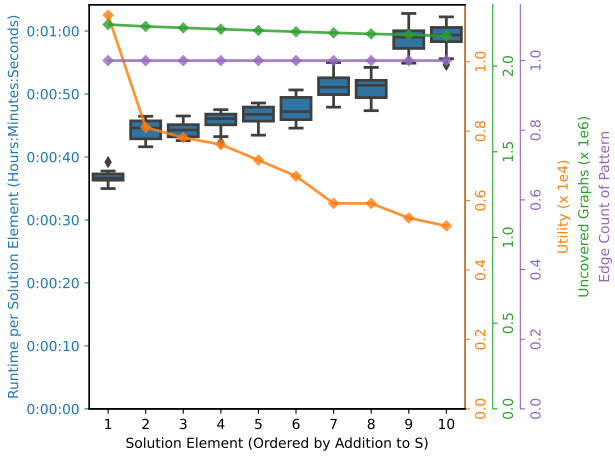


(c) Relative Size Threshold $t = 0.7$

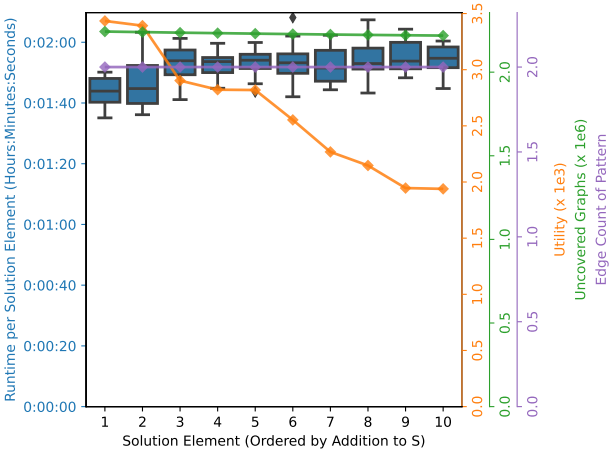
Figure 4.11: Running Time, utility, uncovered graphs, and edge count of $P_{\max}$ for each iteration (i.e., addition to S) of algorithm 6 on the CH*I*PMUNK Heterocycle CoMol dataset. The dataset is filtered to graphs with an edge count of 30 or less. Execution was aborted for each repeat, when the accumulated running time of the individual iterations exceeds 2 days.



(a) Relative Size Threshold $t = 0.3$



(b) Relative Size Threshold $t = 0.5$



(c) Relative Size Threshold $t = 0.7$

Figure 4.12: Running Time, utility, uncovered graphs, and edge count of $P_{\max}$ for each iteration (i.e., addition to S) of algorithm 6 on the Protein Interaction dataset.

types correspond to common substructures in the molecules, which may cause a high coverage for these substructures if the relative size threshold is chosen appropriately. The very high number of labels makes common subgraphs less likely in the case of the Protein Interaction dataset.

4.6 Distributed and Sampling Algorithms

The experimental evaluation of algorithm 6 reveals high practical running times, especially for large datasets with large molecules and high values of the relative size threshold t. The goal of this section is the development of a distributed algorithm utilizing dataset samples to tackle these challenging settings.

4.6.1 Distribution Challenges

Algorithm 6 computes a solution for problem 4.1 (MAX-CSPC) with the best approximation ratio we can hope for (cf., section 2.4), which is $1 - \frac{1}{e} \approx 0.63$. In terms of parallelization, however, the greedy iterations are inherently sequential since the marginal gain w.r.t. the utility depends on the existing solution.

As mentioned in section 2.4, there exist several distributed algorithms for problem 2.1 (MAX-CSSF), i.e., the submodular set function maximization problem in general. However, all of the distributed solutions, that actually parallelize the computation of the solution set S, have lower approximation ratios than the sequential greedy algorithm. The best-known distributed approximation ratio of 0.545 is achieved by Mirrokni and Zadimoghaddam [MZ15].

[MZ15] Mirrokni and Zadimoghaddam "Randomized Composable Core-sets for Distributed Submodular Maximization". 2015

In addition, most distributed algorithms impose further restrictions on the utility function. The GreeDi algorithm of Mirzasoleiman et al. [Mir+13], for example, requires some smoothness of the utility function to have a constant (expected) approximation ratio, that does not degrade with the number of distributed workers. However, the required λ-Lipschitz continuity cannot be guaranteed for problem 4.1 (MAX-CSPC), since minimal modification of a graph pattern may result in an arbitrary high change of utility, by changing the support of the pattern to zero.

[Mir+13] Mirzasoleiman et al. "Distributed Submodular Maximization: Identifying Representative Elements in Massive Data". 2013

A big challenge in the distribution of problem 4.1 (MAX-CSPC) is the dependency of $f_{\mathcal{G},\lhd}$ on the graph dataset $\mathcal{G}$ during its evaluation. Many algorithms, such as the one given in [MZ15], assume that the utility of some object o can be evaluated solely based on S and o. Badanidiyuru et al. [Bad+14] approach this problem in their Sieve-Streaming algorithm with the help of reservoir samples in the presence of additively decomposable submodular set functions (cf., definition 2.4). While this property holds for $f_{\mathcal{G},\lhd}$, the resulting sample cardinalities for good approximation ratios and moderately high cardinality constraints k range in the millions. Furthermore, Sieve-Streaming requires a fixed input set to select the solution from, which is a problem for subgraph pattern mining, where the subgraph patterns also depend on the reservoir sample.

[Bad+14] Badanidiyuru et al. "Streaming Submodular Maximization: Massive Data Summarization on the Fly". 2014

For the above-mentioned reasons, no distributed computation of the solution set S is performed in the following. Instead, distribution is focused on a single greedy round of the sequential greedy approximation algorithm.

4.6.2 A Two-Phase Sampling Approach for a Single Greedy Iteration

As discussed above, a major distribution challenge is the dependency of $f_{\mathcal{G},\triangleleft}$ on the graph dataset $\mathcal{G}$. A method to reduce this dependency to a random sample $\mathfrak{S}$ of $\mathcal{G}$ will be presented in the following. The method will focus on a single iteration (cf., eq. (4.2)) of the greedy approximation described in section 4.5. Given the discussed optimization to remove uncovered graphs in each greedy iteration (cf., section 4.5.2), the following subtask can be extracted from algorithm 6 (lines 5 to 14).

Problem 4.2 (MAX-CSPC-1). Single Subgraph Pattern Coverage Maximization

Input: *A graph dataset $\mathcal{G}$ and a subgraph pattern coverage relation $\triangleleft$.*

Task: *Solve*

$$P_{\max} := \arg\max_{P \in \mathcal{G}_{\sqsubseteq}} f_{\mathcal{G},\triangleleft}(\{P\})$$

For the latter analysis, it should be noted, that $P_{\max}$ is ambiguously defined and multiple patterns with maximal utility might exist.

Algorithm 7: Two-phase sampling framework for problem 4.2 (MAX-CSPC-1)

Input: graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\triangleleft$
Output: graph pattern $P_{\max_{\mathfrak{C}}}$

Phase 1: Compute a set of candidate patterns $\mathfrak{C} \subseteq \mathcal{G}_{\sqsubseteq}$ given a random sample $\mathfrak{S}$ of $\mathcal{G}$.

Phase 2: Select and return a pattern $P_{\max_{\mathfrak{C}}} \in \mathfrak{C}$ with maximal coverage regarding $\mathcal{G}$, i.e., $f_{\mathcal{G},\triangleleft}(\{P_{\max_{\mathfrak{C}}}\}) := \max_{P \in \mathfrak{C}} f_{\mathcal{G},\triangleleft}(\{P\})$.

Algorithm 7 describes a two-phase sampling framework for problem 4.2 (MAX-CSPC-1). This framework will lay the foundation for a distributed computation of problem 4.1 (MAX-CSPC) (cf., section 4.6.8). Additionally, it will lead to a non-distributed sampling and streaming algorithm (cf., section 4.6.6), which only needs to hold a random sample $\mathfrak{S}$ of $\mathcal{G}$ in memory. A central concept of the framework will be a stochastic candidate test (cf., Statistical Hypothesis Test Question 4.1 in section 4.6.3.1) to bound the probability of an incorrect result w.r.t. problem 4.2 (MAX-CSPC-1).

4.6.3 Phase 1 – Candidate Computation

This subsection will discuss the basics to compute the set of candidates $\mathfrak{C}$ in phase 1 of algorithm 7.

4.6.3.1 Stochastic Model

A stochastic model for the coverage utility $f_{\mathcal{G},\lhd}$ is given below. This model will allow the application of statistical hypothesis tests on random samples of $\mathcal{G}$ and is similar to the model presented in lemma 3.2.

Lemma 4.5. *Given a graph dataset $\mathcal{G}$, an equally distributed random sample (with replacement) $\mathfrak{S} = (G_1, \ldots, G_N)$ of graphs from $\mathcal{G}$, a pattern $P \in \mathcal{G}_{\sqsubseteq}$, and a subgraph pattern coverage relation $\lhd$, the random variable $f_{\mathfrak{S},\lhd}(\{P\}) = |\mathfrak{S}_{\rhd P}|$ follows the binomial distribution $B_{n,\frac{|\mathcal{G}_{\rhd P}|}{|\mathcal{G}|}}$.*

Proof. Given the sample space $\mathcal{G}$ and a fixed pattern $P \in \mathcal{G}_{\sqsubseteq}$ a graph $G \in \mathcal{G}$ is either covered by P ($G \in \mathcal{G}_{\rhd P}$) or not ($G \in \mathcal{G}_{\not\rhd P} = \mathcal{G} \setminus \mathcal{G}_{\rhd P}$). When picking a graph G from $\mathcal{G}$ at random, the random variable $X : \mathcal{G} \to \{0, 1\}$ with the events $G \in \mathcal{G}_{\rhd P}$ (success, value 1) and $G \in \mathcal{G}_{\not\rhd P}$ (failure, value 0) has a success rate of $p := \frac{|\mathcal{G}_{\rhd P}|}{|\mathcal{G}|}$ and a failure rate of $\frac{|\mathcal{G}_{\not\rhd P}|}{|\mathcal{G}|} = \frac{|\mathcal{G}|-|\mathcal{G}_{\rhd P}|}{|\mathcal{G}|} = 1 - p$. Thus, X follows the Bernoulli distribution $B_{1,p}$. Let $\mathcal{S} := (x_1, \ldots x_N)$ be the random sample representing the observation of the random variables $X(G_1), \ldots X(G_N)$, i.e., N i.i.d. random variables following $B_{1,p}$. Then, $|\mathfrak{S}_{\rhd P}| = \sum_{x \in \mathcal{S}} x$ is the number of successes and follows the Binomial distribution $B_{N,p}$. □

With the help of lemma 4.5 it is possible to statistically compare the coverage of patterns to each other.

Statistical Hypothesis Test Question 4.1 (Candidate Test).

Input:	*Two equally distributed random samples (with replacement) $\mathfrak{S}_1 = (G_{11}, \ldots, G_{1N})$ and $\mathfrak{S}_2 = (G_{21}, \ldots, G_{2N})$ of graphs from graph dataset $\mathcal{G}$, a reference pattern $R \in \mathcal{G}_{\sqsubseteq}$, a candidate pattern $C \in \mathcal{G}_{\sqsubseteq}$, and a significance level α.*
Hypothesis:	$H_0 : f_{\mathcal{G},\lhd}(\{C\}) = \lvert\mathcal{G}_{\rhd C}\rvert \geq \lvert\mathcal{G}_{\rhd R}\rvert = f_{\mathcal{G},\lhd}(\{R\})$
Test Question:	*Can H_0 be rejected w.r.t. the observations $\lvert\mathfrak{S}_{1_{\rhd C}}\rvert$ and $\lvert\mathfrak{S}_{2_{\rhd R}}\rvert$ and significance level α?*

In other words, given a reference pattern R, the test can identify candidate patterns, that most likely do not have a better utility than R. Consequentially, such patterns most likely do not have an optimal utility.

4.6.3.2 Algorithm Outline for Phase 1

Given the above observation, it is possible to design an algorithm that generates a candidate set $\mathfrak{C}$ by enumerating $\mathcal{G}_{\sqsubseteq}$ and keeping only the patterns for which H_0 cannot be rejected (cf., algorithm 8).

Algorithm 8: Outline for phase 1 of algorithm 7

Input: random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ of graph dataset $\mathcal{G}$, reference pattern $R \in \mathcal{G}_{\sqsubseteq}$, subgraph pattern coverage relation $\triangleleft$, and confidence level α
Output: candidate set $\mathfrak{C}$

```
1 foreach P ∈ G_⊑ do                          ▶ G_⊑ enumerated by FSM algorithm
2     if candidateTest_{S1,S2}(P, R, α) ≠ rejected then
3         C ← C ∪ {P};
4 if C = ∅ then
5     C ← {R};
6 return C;
```

To always return a non-empty candidate set in algorithm 8, R is added to the candidate set $\mathfrak{C}$ in this case. While $R \in \mathcal{G}_{\sqsubseteq}$ holds, i.e., R is tested as a candidate pattern as well, R may get rejected. This is a consequence of the two independent samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$. While algorithm 8 outlines an algorithm idea, several open aspects will be discussed in the following. These aspects involve questions regarding the test realization, the test applicability, the selection of the reference pattern, and performance optimizations from algorithm 6 that are omitted in algorithm 8.

4.6.3.3 Overall Error Bound and Independence of Individual Candidate Tests

The candidate test (cf., Statistical Hypothesis Test Question 4.1) was introduced with the goal to bound the overall error of algorithm 8. More precisely, the aim is to bound the probability of algorithm 8 to return an incorrect result. However, the statistical hypothesis test itself only bounds the statistical error α of an isolated random experiment for a single candidate pattern P. Since the candidate test needs to be performed for each pattern $P \in \mathcal{G}_{\sqsubseteq}$, the following questions arise:

1. Is it necessary to apply a multiple hypothesis testing correction to bound a family-wise error rate?

2. Is it necessary for each test to be statistically independent? In other words, is it sufficient to use the fixed random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ for all tests or are independent samples $\mathfrak{S}_{P,1}$ and $\mathfrak{S}_{P,2}$ needed for each pattern P.

Lemma 4.6. *Algorithm 8 computes a candidate set* $\mathfrak{C}$ *which contains* $P_{\max}$ *with a probability of at least* $1 - \alpha$.

Proof. Without loss of generality, let $P_{\max}$ be an arbitrary, but fixed pattern with maximal utility w.r.t. $\mathcal{G}$ (cf., problem 4.2 (MAX-CSPC-1)). Then, $P_{\max}$ is either R or the parameter of exactly one statistical hypothesis test $\texttt{candidateTest}_{\mathfrak{S}_1,\mathfrak{S}_2}(P_{\max}, R, \alpha)$. If $P_{\max} = R$, R is contained in $\mathfrak{C}$. If $P_{\max} \neq R$, $\texttt{candidateTest}_{\mathfrak{S}_1,\mathfrak{S}_2}(P_{\max}, R, \alpha)$ does reject $P_{\max}$ with a probability of at most α, i.e., the probability of $P_{\max}$ to be contained in $\mathfrak{C}$ is at least $1 - \alpha$. This test is not influenced by the outcome of $\texttt{candidateTest}_{\mathfrak{S}_1,\mathfrak{S}_2}(P, R, \alpha)$ with $P \neq P_{\max}$. □

Both above questions are answered by the proofs observation, that a single test actually bounds to the overall error and it is safe to discard any other test or pattern without any influence on the worst-case error bound of the algorithm. Thus, no multiple hypothesis testing correction must be applied and two fixed random samples are sufficient.

Up to this point it was omitted, that $P_{\max}$ is ambiguously defined in problem 4.2 (MAX-CSPC-1) and that multiple patterns with maximum utility may exist. This does not change the outcome of lemma 4.6, since $P_{\max}$ was defined as an arbitrary, but fixed pattern with maximum utility. However, it is possible that $P_{\max}$ is not returned in phase 2 of algorithm 7.

Corollary 4.7. *Algorithm 7 returns a pattern* $P_{\max_{\mathfrak{C}}}$ *of maximum utility—i.e.,* $f_{\mathcal{G},\triangleleft}(\{P_{\max_{\mathfrak{C}}}\}) = \max_{P \in \mathcal{G}_{\sqsubseteq}} f_{\mathcal{G},\triangleleft}(\{P\}) = f_{\mathcal{G},\triangleleft}(\{P_{\max}\})$*—with a probability of at least* $1 - \alpha$.

Proof. The fixed pattern $P_{\max}$ from lemma 4.6 is contained in $\mathfrak{C}$ with a probability of at least $1-\alpha$. The returned pattern $P_{\max_{\mathfrak{C}}}$ by phase 2 of algorithm 7 is defined to be a pattern from $\mathfrak{C}$ with maximal utility w.r.t. $\mathcal{G}$, i.e., $f_{\mathcal{G},\triangleleft}(\{P_{\max_{\mathfrak{C}}}\}) = \max_{P \in \mathfrak{C}} f_{\mathcal{G},\triangleleft}(\{P\})$. If $P_{\max}$ is contained in $\mathfrak{C}$, the equality $f_{\mathcal{G},\triangleleft}(\{P_{\max_{\mathfrak{C}}}\}) = f_{\mathcal{G},\triangleleft}(\{P_{\max}\})$ holds, since $P_{\max}$ is defined as a pattern with maximum utility from $\mathcal{G}_{\sqsubseteq}$ and there cannot exist a pattern with higher utility in $\mathfrak{C} \subseteq \mathcal{G}_{\sqsubseteq}$. □

Thus, the ambiguity of $P_{\max}$ does not worsen the worst-case error bound of algorithm 7. In contrast, the success rate of algorithm 7 actually increases with each additional pattern $P' \neq P_{\max}$ with maximum utility since P' may be contained in $\mathfrak{C}$ even if the candidate test falsely rejects $P_{\max}$. However, this effect cannot be statistically quantified, since the individual tests are not independent.

4.6.3.4 Statistical Test Realization

Multiple statistical tests for the two-sample binomial testing problem as required for the candidate test from Statistical Hypothesis Test Question 4.1 are available. Exact tests, such as Fisher's, Barnard's or Boschloo's tests, are known to be hard to compute for large sample sizes, since the values of the binomial coefficients become extremely large. As a consequence, the two-sample approximate binomial test on equality (cf.,

section 2.3.5.2) is used in the following. The test imposes several assumptions about the setting in order to be applicable. Besides the obvious assumption of binomially distributed samples, two further assumptions influence the algorithm design. First, the two samples must be independent. This is guaranteed by the test statistics, which are computed over individual random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$. Second, the test only computes results with a reasonable approximation ratio if the absolute number of successes and failures in both samples is larger than a minimum constant. As a statistical rule of thumb (cf., [TK14]) this constant should be not less than five for non-extreme significance levels, i.e., $|\mathfrak{S}_{1_{\triangleright C}}| > 5$, $\left|\mathfrak{S}_{1_{\not\triangleright C}}\right| > 5$, $|\mathfrak{S}_{2_{\triangleright R}}| > 5$, and $\left|\mathfrak{S}_{2_{\not\triangleright R}}\right| > 5$. This limitation can be circumvented using a strategy, which is based on the observation that the rejection region (cf., lemma 4.8) of the test is monotonically increasing with $|\mathfrak{S}_{2_{\triangleright R}}|$ for a fixed sample size. The strategy includes the following rules: (a) If the rejection region of the test is smaller than five, every pattern must be accepted as candidate. (b) If $\left|\mathfrak{S}_{2_{\not\triangleright R}}\right| \leq 5$, use $N - 6$ as test statistic for R. (c) If $\left|\mathfrak{S}_{2_{\not\triangleright C}}\right| \leq 5$, select C as candidate. Selecting a pattern unconditionally will lead to correct results under all circumstances since the pattern will not be missed in phase 2 of algorithm 7. Item a covers the assumption $|\mathfrak{S}_{1_{\triangleright C}}| > 5$, since the rejection region is the threshold for this test statistic. Furthermore, it covers the assumption $|\mathfrak{S}_{2_{\triangleright R}}| > 5$, since the rejections region is always smaller than $|\mathfrak{S}_{2_{\triangleright R}}|$ as long α is smaller than 50% (which is a reasonable assumption in practice and a result of the symmetry of the normal distribution). Item b is justified by the fact that $\left|\mathfrak{S}_{2_{\not\triangleright R}}\right| = x \Leftrightarrow |\mathfrak{S}_{2_{\triangleright R}}| = N - x$, with $0 \leq x \leq N$. As a consequence of the above-mentioned monotonicity of the rejection region, it is safe two lower the test statistic $|\mathfrak{S}_{2_{\triangleright R}}|$ for the reference pattern without missing a pattern as candidate.

[**TK14**] Taeger and Kuhnt, *Statistical Hypothesis Testing with SAS and R*. 2014

4.6.3.5 Subgraph Pattern Space of the Sampled Dataset

Algorithm 8 enumerates the subgraph pattern space $\mathcal{G}_{\sqsubseteq}$. This contradicts the goal to be independent of $\mathcal{G}$. For this reason, the subgraph pattern space $\mathfrak{S}_{1_{\sqsubseteq}}$ of the first random sample $\mathfrak{S}_1$ is used in the final algorithm 9. Using $\mathfrak{S}_1$ instead of $\mathfrak{S}_2$ or $\mathfrak{S}_1 \cup \mathfrak{S}_2$ has the benefit, that it enables a support-based search space pruning (cf., section 4.6.3.6). However, this change does influence the correctness of the algorithm for a corner case. If the absolute maximal coverage is very small, it may happen, that the probability of $P_{\max}$ to be contained in $\mathfrak{S}_{1_{\sqsubseteq}}$ is below $1 - \alpha$. Consequentially, the resulting error is higher than the parameter for the statistical test. On the contrary, the correctness of the overall error is preserved, whenever the rejection region of an exact[9] hypothesis test is 1 or above. In this case, every pattern $P \in \mathcal{G}_{\sqsubseteq} \wedge P \notin \mathfrak{S}_{1_{\sqsubseteq}}$ would be rejected by the candidate test anyway. This observation is important since it makes the detection of such a situation possible, i.e., the algorithm can return the validity of the result w.r.t. to corollary 4.7. The situation of very small utility values is also problematic in the sense, that it typically leads to very large candidate

[9] The large sample restriction of the approximate binomial test on equality requires a rejection region of at least $\mathrm{ex}_{\min}$, where $\mathrm{ex}_{\min}$ is the lower bound of success and failure examples required for sufficient test accuracy (cf., section 2.3.5.2).

sets, possibly $\mathfrak{C} = \mathcal{G}_{\sqsubseteq}$. However, the presented corner case has very limited practical relevance. See section 4.6.10.3 for a detailed analysis of this topic.

4.6.3.6 Deriving a Support-based Pruning Bound

The optimized sequential algorithm 6 for problem 4.1 (MAX-CSPC) prunes the pattern search space based on the minimum support threshold $\text{supp}_{\min}$ as described in section 4.5.3. The central observation is, that the support of a pattern P cannot be smaller than the coverage utility $f_{\mathcal{G},\lhd}\{P\}$ (cf., eq. (4.4)). This inequality does no longer hold for our sampled approach since patterns with a lower support than the reference pattern R might be added to the candidate set as a consequence of the statistical candidate test. However, for fixed sample sizes, a fixed significance level α, and a fixed sample proportion $\hat{p}_2 := |\mathfrak{S}_{2_{\triangleright R}}|$ for a reference pattern R, it is possible to compute the rejection region $[-\infty, \hat{p}_{1l}) \cup (\hat{p}_{1u}, \infty]$ of the two-sample binomial test on equality (cf., section 2.3.5.2) as given in lemma 4.8. It should be noted, that lemma 4.8 is referencing the two-sided test, while Statistical Hypothesis Test Question 4.1 is a one-sided test. However, the rejection threshold $\hat{p}_{1l}$ for the one-sided test can be derived by setting the significance level to $\frac{\alpha}{2}$ (which increases the test power). Thus, the inequality

$$\forall P \in \mathfrak{C} : \text{supp}_{\mathfrak{S}_1}(P) \geq \hat{p}_{1l}. \tag{4.5}$$

holds. Since $\hat{p}_{1l} \in \mathbb{R}$ and $\text{supp}_{\mathfrak{S}_1}(P) \in \mathbb{N}$ the pruning threshold $\text{supp}_{\min}$ of the FSM algorithm can be set to $\text{supp}_{\min} = \lceil \hat{p}_{1l} \rceil$ (the FSM algorithm is applied to $\mathfrak{S}_1$ as described in section 4.6.3.5).

Lemma 4.8 (Rejection Region of the Two-Sample Approximate Binomial Test on Equality). *Given the two-sample approximate binomial test on equality with $H_0 : p_1 = p_2$ (see section 2.3.5.2), a fixed sample size $N = N_1 = N_2$, a fixed sample proportion $\hat{p}_2$, and a fixed significance level α the rejection region for $\hat{p}_1$ is $[-\infty, \hat{p}_{1l}) \cup (\hat{p}_{1u}, \infty]$ with*

$$\begin{aligned}
\hat{p}_{1l} &= a - b \\
\hat{p}_{1u} &= a + b \\
a &= \left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right) \frac{N}{y^2 + 2N} \\
b &= \sqrt{\left(\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right)\left(-\frac{N}{y^2 + 2N}\right)\right)^2 + \left(\frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2\right)\frac{2N}{y^2 + 2N}} \\
y &= \Phi^{-1}\left(\frac{\alpha}{2}\right)
\end{aligned}$$

Proof. We show that $\hat{p}_{1l}$ and $\hat{p}_{1u}$ are the most extreme values for $\hat{p}_1$ for which H_0 cannot be rejected. Thus, we set $\Phi^{-1}(\frac{\alpha}{2}) = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{N_1} + \frac{1}{N_2})}}$, with $\hat{p} = \frac{\hat{p}_1 N_1 + \hat{p}_2 N_2}{N_1 + N_2}$.

Let $y := \Phi^{-1}(\frac{\alpha}{2})$.

$$
\begin{aligned}
& y && = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{N\hat{p}_1+N\hat{p}_2}{2N}(1-\frac{N\hat{p}_1+N\hat{p}_2}{2N})\frac{2}{N}}} \\
\Leftrightarrow\ & \hat{p}_1 - \hat{p}_2 && = y\sqrt{\frac{\hat{p}_1+\hat{p}_2}{2}(1-\frac{\hat{p}_1+\hat{p}_2}{2})\frac{2}{N}} \\
\Leftrightarrow\ & \hat{p}_1 - \hat{p}_2 && = y\sqrt{\frac{2}{N}\frac{\hat{p}_1+\hat{p}_2}{2} + \frac{2}{N}\frac{\hat{p}_1+\hat{p}_2}{2}\left(-\frac{\hat{p}_1+\hat{p}_2}{2}\right)} \\
\Leftrightarrow\ & \hat{p}_1 - \hat{p}_2 && = y\sqrt{\frac{\hat{p}_1}{N} + \frac{\hat{p}_2}{N} - \frac{\hat{p}_1^2}{2N} - 2\frac{\hat{p}_1\hat{p}_2}{2N} - \frac{\hat{p}_2^2}{2N}} \\
\Leftrightarrow\ & (\hat{p}_1 - \hat{p}_2)^2 && = \frac{\hat{p}_1 y^2}{N} + \frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_1^2 y^2}{2N} - \frac{\hat{p}_1\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} \\
\Leftrightarrow\ & (\hat{p}_1 - \hat{p}_2)^2 && = \hat{p}_1\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N}\right) + \frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_1^2 y^2}{2N} - \frac{\hat{p}_2^2 y^2}{2N} \\
\Leftrightarrow\ & 0 && = \hat{p}_1\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N}\right) + \frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_1^2 y^2}{2N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_1^2 + 2\hat{p}_1\hat{p}_2 - \hat{p}_2^2 \\
\Leftrightarrow\ & 0 && = \hat{p}_1\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right) + \hat{p}_1^2\left(-\frac{y^2}{2N} - 1\right) + \frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2 \\
\Leftrightarrow\ & 0 && = \hat{p}_1^2 + \frac{\hat{p}_1\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right) + \frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2}{-\frac{y^2}{2N} - 1} \\
\Leftrightarrow\ & 0 && = \hat{p}_1^2 + \frac{\hat{p}_1\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right)}{-\frac{y^2}{2N} - 1} + \left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right)^2 - \left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right)^2 \\
& && \quad + \frac{\frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2}{-\frac{y^2}{2N} - 1} \\
\Leftrightarrow\ & 0 && = \left(\hat{p}_1 + \left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right)\right)^2 - \left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right)^2 + \frac{\frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2}{-\frac{y^2}{2N} - 1} \\
\Leftrightarrow\ & \hat{p}_1 && = \pm\sqrt{\left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right)^2 - \frac{\frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2}{-\frac{y^2}{2N} - 1}} - \left(\frac{1}{2}\frac{\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2}{-\frac{y^2}{2N} - 1}\right) \\
\Leftrightarrow\ & \hat{p}_1 && = \pm\sqrt{\left(\left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right)\left(-\frac{N}{y^2+2N}\right)\right)^2 + \left(\frac{\hat{p}_2 y^2}{N} - \frac{\hat{p}_2^2 y^2}{2N} - \hat{p}_2^2\right)\frac{2N}{y^2+2N}} \\
& && \quad + \left(\frac{y^2}{N} - \frac{\hat{p}_2 y^2}{N} + 2\hat{p}_2\right)\frac{N}{y^2+2N}
\end{aligned}
$$

□

4.6.3.7 Choosing a Reference Pattern

Up to this point, it was assumed that the reference pattern R is a parameter of the algorithm. Thus, the open question is how R should be chosen. A small candidate

set C is desirable w.r.t. to the computational performance of phase 2 of algorithm 7. Since the rejection region of the statistical test is monotonically increasing for larger values of $|\mathfrak{S}_{2_{\triangleright R}}|$, a strong reference pattern should have a high utility $f_{\mathfrak{S}_2,\lhd}(\{R\})$ in the sample $\mathfrak{S}_2$. Thus, the optimal reference pattern w.r.t. to a small candidate set has maximum utility in $\mathfrak{S}_2$.

$$R_{\mathrm{OPT}} := \underset{P \in \mathfrak{S}_{2_{\sqsubseteq}}}{\arg\max}\, f_{\mathfrak{S}_2,\lhd}(\{P\}). \tag{4.6}$$

Computing R_{OPT} requires problem 4.2 (MAX-CSPC-1) to be solved for $\mathfrak{S}_2$. A straightforward approach would be, to run lines 12 to 14 of algorithm 6 in a preprocessing step for algorithm 8. Consequentially, this approach would require two runs of the FSM algorithm for the subgraph pattern spaces $\mathfrak{S}_{1_{\sqsubseteq}}$ (algorithm 8 with support-based pruning optimization as described in section 4.6.3.6) and $\mathfrak{S}_{2_{\sqsubseteq}}$ (calculation of R_{OPT}). Among other things, this involves the overhead of eliminating isomorphic pattern representations during the mining process (cf., paragraph 2.6.2.1.3). To reduce the overhead, an integrated computation of R, which requires only a single run of the FSM algorithm, is presented in the following. The basic idea is to enumerate the patterns $P \in \mathfrak{S}_{1_{\sqsubseteq}}$ over $\mathfrak{S}_1$ only, but keeping track of the utility $f_{\mathfrak{S}_2,\lhd}(\{P\})$ as well. Starting with an empty reference pattern with utility 0, the algorithm subsequently updates the reference pattern $R_{\max}$ to be the reference pattern with maximum observed utility up to this point (similar to the determination of $P_{\max}$ in algorithm 6). The integrated approach has two major consequences. First, it may happen that R_{OPT} is not enumerated by the FSM algorithm, i.e., $R_{\mathrm{OPT}} \notin \mathfrak{S}_{1_{\sqsubseteq}}$. In combination with the support-based pruning, this event even becomes more likely, since the FSM algorithm will filter patterns P with $\mathrm{supp}_{\mathfrak{S}_1}(P) < \lceil \hat{p}_{1l} \rceil$ (cf., section 4.6.3.6). Still the event, that the pattern with the highest utility in $\mathfrak{S}_2$ has a support less than the rejection region in $\mathfrak{S}_1$, is extremely unlikely. Most important, a non-optimal reference pattern will not influence the correctness of the algorithm, i.e., it will only influence the performance and memory requirements of the algorithm. Second, after updating $R_{\max}$ with $R'_{\max}$ (given $f_{\mathfrak{S}_{2_{\sqsupseteq P}},\lhd}(R_{\max}) < f_{\mathfrak{S}_{2_{\sqsupseteq P}},\lhd}(R'_{\max})$) it may happen, that patterns are contained in the candidate set, which would have been discarded if $R'_{\max}$ would have been used in advance. However, when the utility of a pattern is stored together with the tuple in the candidate set, it is possible to filter the candidate set to contain only patterns which do not fall in the rejection region of $R'_{\max}$. This approach is valid because the rejection region is monotonically increasing and the solution is the same as if we we would have used $R'_{\max}$ right from the start.

4.6.4 A Sequential Algorithm for Phase 1

Algorithm 9 is a fully specified algorithm for the outline given in algorithm 8 and implements the above-discussed aspects. As discussed in section 4.6.3.5, it loops over the patterns $P \in \mathfrak{S}_{1_{\sqsubseteq}}$ of the first random sample (line 7) interleaved with an FSM algorithm. In lines 8 to 9 the utility is computed for each sample. Similar to algorithm 6, the coverage utility for a pattern P is only computed over the supporting

Algorithm 9: Phase 1 of Algorithm 7

Input: random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ of graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$

Fixed Parameterization: pruning strategy $\texttt{pruning}_{\triangleleft}$, FSM algorithm fsm, minimum absolute number of Bernoulli examples $\mathsf{ex}_{\min} \in \mathbb{N}^{>0}$ (success and failure) for the candidate test

Output: candidate set $\mathfrak{C}$ and boolean validity of result

procedure $\texttt{phase1}(\mathfrak{S}_1, \mathfrak{S}_2, \triangleleft, \alpha)$
- $R_{\max} \leftarrow$ **null**;
- $\mathsf{utility}_{R_{\max}} \leftarrow 0$;
- $\mathfrak{C} \leftarrow \emptyset$;
- $\hat{p}_{1l} \leftarrow -\infty$;
- $\mathsf{fsm}.\texttt{init}(\mathfrak{S}_1, \mathfrak{S}_2, \texttt{pruning}_{\triangleleft})$;
  - ▶ Uses $\mathfrak{S}_2$ only for transaction lists computation $\mathfrak{S}_{2_{\sqsupseteq P}}$
- $(P, \mathfrak{S}_{1_{\sqsupseteq P}}, \mathfrak{S}_{2_{\sqsupseteq P}}) \leftarrow \mathsf{fsm}.\texttt{next}(1)$;
- **while** $P \neq$ **null do**
  - $\mathsf{utility}_1 \leftarrow f_{\mathfrak{S}_{1_{\sqsupseteq P}}, \triangleleft}(\{P\})$;
  - $\mathsf{utility}_2 \leftarrow f_{\mathfrak{S}_{2_{\sqsupseteq P}}, \triangleleft}(\{P\})$;
  - **if** $\mathsf{utility}_2 > \mathsf{utility}_{R_{\max}}$; **then**
    - $R_{\max} \leftarrow P$;
    - $\mathsf{utility}_{R_{\max}} \leftarrow \mathsf{utility}_2$;
    - $\mathsf{utility}_{R_{\max},\mathrm{valid}} \leftarrow \min\{\mathsf{utility}_2, |\mathfrak{S}_2| - \mathsf{ex}_{\min} - 1\}$;
    - $\hat{p}_{1l} \leftarrow \texttt{rejectionThresh}(\mathsf{utility}_{R_{\max},\mathrm{valid}}, |\mathfrak{S}_2|, \alpha)$;
      - ▶ cf., lemma 4.8
    - $\mathfrak{C} \leftarrow \{(C, \mathsf{utility}_C) \in \mathfrak{C} \mid \mathsf{utility}_C \geq \hat{p}_{1l}\}$;
  - **if** $\mathsf{utility}_1 \geq \hat{p}_{1l}$ **or** $\hat{p}_{1l} < \mathsf{ex}_{\min}$ **or** $\mathsf{utility}_1 \geq |\mathfrak{S}_1| - \mathsf{ex}_{\min}$ **then**
    - $\mathfrak{C} \leftarrow \mathfrak{C} \cup (P, \mathsf{utility}_1)$;
  - $(P, \mathfrak{S}_{1_{\sqsupseteq P}}, \mathfrak{S}_{2_{\sqsupseteq P}}) \leftarrow \mathsf{fsm}.\texttt{next}(\max\{1, \lfloor \hat{p}_{1l} + 1 \rfloor\})$;
- $\mathsf{validity} \leftarrow \hat{p}_{1l} \geq \mathsf{ex}_{\min}$;
- **if** $\mathfrak{C} = \emptyset$ **then**
  - $\mathfrak{C} \leftarrow \{(R_{\max}, 0)\}$;
- **return** $(\{C \mid (C, \mathsf{utility}_C) \in \mathfrak{C}\}, \mathsf{validity})$;

graphs in the sample. However, since the FSM algorithm is applied to $\mathfrak{S}_1$, only $\mathfrak{S}_{1_{\sqsupseteq P}}$ would be computed by an unmodified FSM algorithm. As discussed in the preliminaries, the downward closure property enables a very efficient transaction list computation (cf., paragraph 2.6.2.1.4) for a bottom-up exploration of the pattern space. For this reason, the FSM algorithm is slightly modified to compute the transactions list for the second random sample $\mathfrak{S}_2$, even if these lists are not used by the FSM algorithm itself. This enables `fsm.next` to additionally return $\mathfrak{S}_{2_{\sqsupseteq P}}$ and to use the optimized coverage utility computation, which skips the subgraph isomorphism test (cf., section 4.5.4). The conditional block starting in line 10 handles the case in which a new pattern with maximum utility is found in $\mathfrak{S}_2$, i.e., $R_{\max}$ has to be updated as described in section 4.6.3.7. Besides the update of $R_{\max}$ itself (line 11) and its utility (line 12), this basically means to update the rejection threshold (line 14) and to filter the candidate set to remove obsolete items (line 15) as described in sections 4.6.3.6 and 4.6.3.7. To have a valid statistical test setting, line 13 safeguards non-valid parameters of `rejectionThresh` (cf., section 4.6.3.4, item b). The conditional block in line 16 finally adds patterns to the candidate set $\mathfrak{C}$. Additional to the rejection threshold-based filtering it also implements the safeguards described in items a and c from section 4.6.3.4. The statistical validity of the algorithm as described in section 4.6.3.5 is determined in line 19.

4.6.4.1 Computational Complexity of the Sequential Algorithm for Phase 1

Algorithm 9 utilizes an FSM algorithm to enumerate the patterns for the candidate check. In alignment with the complexity analysis of algorithm 6, this analysis will be orientated towards the overhead, that is caused by algorithm 9 for each pattern returned by `fsm.next`.

At the very core, the FSM algorithm itself was modified to compute two separate transaction lists. While the expected size of the second transaction list $\mathfrak{S}_{2_{\sqsupseteq P}}$ for a pattern P should be the same as for $\mathfrak{S}_{1_{\sqsupseteq P}}$, it might be much larger in the worst case. The most extreme case, i.e., $\left|\mathfrak{S}_{1_{\sqsupseteq P}}\right| = 1$ and $\left|\mathfrak{S}_{2_{\sqsupseteq P}}\right| = |\mathfrak{S}_2|$, is possible. Additionally, $\mathfrak{S}_2$ might contain graphs of a different size, which is a parameter for the worst-case asymptotic running time of the subgraph isomorphism tests. Thus, we cannot assume the running time of our modified algorithm to have only a constant overhead w.r.t. $\texttt{FSM}_{\mathfrak{S}_1,\text{supp}_{\min}}$ Furthermore, the analysis in section 4.5.7.1 assigns costs to graph is the transactions lists of a pattern. For these reasons, the running time $\texttt{FSM}_{\mathfrak{S}_{1\cup 2},\text{supp}_{\min}}$ will be used as a reference in the following analysis, where $\text{supp}_{\min}$ is the support-based pruning bound inflicted by the original instance and $\mathfrak{S}_{1\cup 2}$ is the union of the two samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$. For the associated FSM instance

$$\begin{aligned}\mathfrak{S}_{1\cup 2_{\sqsupseteq P}} &= \{\!\{G \in \mathfrak{S}_{1\cup 2} \mid P \sqsubseteq G\}\!\} \\ &= \{\!\{G \in \mathfrak{S}_1 \mid P \sqsubseteq G\}\!\} \cup \{\!\{G \in \mathfrak{S}_2 \mid P \sqsubseteq G\}\!\} \\ &= \mathfrak{S}_{1_{\sqsupseteq P}} \cup \mathfrak{S}_{1_{\sqsupseteq P}}\end{aligned}$$

holds. Additionally, the frequent patterns in $\mathfrak{S}_{1\cup 2}$ will be a superset of the frequent patterns in $\mathfrak{S}_1$ for a fixed absolute minimum support $\text{supp}_{\min}$. Thus, the resulting

complexity is not less than the original instance. As described in section 4.5.7.1, a fixed $\text{supp}_{\min}$ value cannot be assumed. Instead, the upper bound $\texttt{FSM}_{\mathfrak{S}_{1\cup 2},1}$ is used for the analysis.

Given the above reasoning, we have the following costs assigned to each operation of algorithm 9. Lines 2 to 6 are either constant-time operations or part of the FSM algorithm. The costs to compute the utility in lines 8 and 9 can be assigned to the transaction lists of the first and second sample as described in section 4.5.7.1. Lines 12 to 14 are again constant-time operations. The filtering of the candidate set $\mathfrak{C}$ in line 15 requires some additional rationale in order to be a constant time operation per pattern. The algorithm can resort to a utility sorted array of references to candidate lists where each candidate hast the utility of the reference index. Since $\text{utility}_C \in [1, |\mathfrak{S}_1|]$ holds, we can apply a bucket sort with $\mathcal{O}(1)$ insertion time, which requires only $O(|\mathfrak{S}_1|)$ additional memory in comparison with a flat list representation. The filtering step then only needs to discard all buckets with an index smaller than $\hat{p}_{1l}$. In theory, this filtering step could be also placed before the return statement of the procedure (line 22). However, this can result in a higher memory utilization, which is a critical resource in practice. Lines 16 to 19 are again constant-time operations. The mapping of the tuples $(C, \text{utility}_C) \to C$ in line 22 requires a linear scan of the candidates, which is linear in the number of observed patterns.

Overall, this results in a computational complexity of $\mathcal{O}(\texttt{FSM}_{\mathfrak{S}_{1\cup 2},1})$.

4.6.5 A Shared-Memory Parallel Algorithm for Phase 1

In this section, a shared-memory parallelized algorithm will be given for phase 1, which is based on Algorithm 9. It incorporates a parallelized enumeration and candidate evaluation of frequent patterns.

> **Observation 4.9.** *Given a fixed rejection threshold $\hat{p}_{1l}$ each pattern's candidate status can be evaluated independently.*

Since the greedy approximation algorithm of Nemhauser, Wolsey, and Fisher (cf., algorithm 1) does not require a deterministic tie-breaking rule to determine an object with maximum utility (if multiple of such objects are present), the ordering of patterns processed by the main loop in line 7 of algorithm 9 does not influence the correctness of the algorithm. However, the premise of a fixed rejection threshold is not true for an on the fly selection of a reference pattern (cf., section 4.6.3.7). Nevertheless, in a shared memory environment, it is possible to share the variables and datastructures $R_{\max}$, $\text{utility}_{\max}$, $\hat{p}_{1l}$, and $\mathfrak{C}$ and perform synchronized updates on them.

A thread-safe candidate evaluation is given by the procedure `candidateEval` in algorithm 10, which is based on lines 8 to 17 of algorithm 9. The synchronized update of $R_{\max}$, $\text{utility}_{\max}$, and $\hat{p}_{1l}$ has a duplicate nested if-clause to check if $\text{utility}_2 > \text{utility}_{R_{\max}}$. This avoids a synchronized check for each pattern, which can cause lock contention. Instead, only real updates are performed in a synchronized manner. This requires a second check to guarantee the correctness in cases where

Algorithm 10: Shared-Memory Parallelized Candidate Evaluation

Input: subgraph pattern $P \in \mathcal{G}_{\sqsubseteq}$ of a graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\triangleleft$, supporting graphs $\mathfrak{S}_{1_{\sqsupseteq P}}$ and $\mathfrak{S}_{2_{\sqsupseteq P}}$ of the random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ drawn from $\mathcal{G}$, confidence level $\alpha \in [0, 0.5]$

Fixed Parameterization: minimum absolute number of Bernoulli examples $\mathsf{ex}_{\min} \in \mathbb{N}^{>0}$ (success and failure) for the candidate test

Shared-Memory Variables: $R_{\max}, \mathsf{utility}_{R_{\max}}, \mathfrak{C}, \hat{p}_{1l}$

Subprocedure Of: algorithm 11

```
1  procedure candidateEval(P, 𝔖1_⊒P, 𝔖2_⊒P, ◁, α)
2      utility_1 ← f_{𝔖1_⊒P, ◁}({P});
3      utility_2 ← f_{𝔖2_⊒P, ◁}({P});
4      if utility_2 > utility_{R_max}; then
5          synchronized
6              if utility_2 > utility_{R_max}; then
7                  R_max ← P;
8                  utility_{R_max} ← utility_2;
9                  utility_{R_max,valid} ← min{utility_2, |𝔖2| − ex_min − 1};
10                 p̂_1l ← rejectionThresh(utility_{R_max,valid}, |𝔖2|, α);
                                                  ▶ cf., lemma 4.8
11             synchronized ℭ
12                 ℭ ← {(C, utility_C) ∈ ℭ | utility_C ≥ p̂_1l};
13     if utility_1 ≥ p̂_1l or p̂_1l < ex_min or utility_1 ≥ |𝔖1| − ex_min then
14         synchronized ℭ
15             ℭ ← ℭ ∪ (P, utility_1);
```

utility$_{R_{\max}}$ was changed before the synchronized block is entered,i.e., several updates are attempted in parallel.

It is required to parallelize the FSM algorithm itself to scale with the number of processors since the frequent pattern enumeration of a sequential FSM algorithm is not fast enough to saturate a parallel pipeline of candidate evaluations as given in algorithm 10. As described in section 4.6.4.1 algorithm 9 only performs lightweight and graph size-independent operations with a constant overhead w.r.t. enumerated patterns and graphs in the transaction lists. The major computational work is actually done in `fsm.next`, which involves the subgraph isomorphisms tests between the pattern and each graph in the transaction lists as well as the canonical representation test (cf., section 2.6.2). These operations have an exponential worst-case complexity w.r.t. to the graph size and are not cheap in practice, even for moderately sized graphs.

[**Bor07**] Borgelt, "Canonical Forms for Frequent Graph Mining". 2007

The specific implementation of the FSM algorithm becomes relevant at this point. This thesis will use the gSpan algorithm with a DFS code variant of [Bor07] (cf., paragraph 2.6.2.2.1) for the enumeration of frequent subgraph patterns. The gSpan algorithm was chosen since it uses transaction lists. Embedding lists are known to have an unpredictable behavior w.r.t. to memory consumption and performance, i.e., some instance perform exceptionally bad while others are fast in comparison with the transaction list setting (cf., section 2.6.2.3). Furthermore, the search space canonization of gSpan is easy to parallelize, which will be shown in the following.

As described in section 2.6.1.1 the subgraph pattern space can be partially ordered by the subgraph isomorphism relation and forms a poset ranked by the graph size. The gSpan algorithm treeifies the search space by eliminating pattern representations with non-minimal, i.e., non-canonical, DFS codes.

Definition 4.3. *Let $\mathcal{G}$ be a graph dataset, let the vertex and edge labels in $\mathcal{G}$ have an arbitrary fixed linear ordering, and let* $\text{mincode} : \mathcal{G}_{\sqsubseteq} \to l_{vs}(t_d t_s l_e l_{vd})^m$ *(cf., eq.* (2.13)*) be a representative function (of the patterns equivalence class), that returns the lexicographically minimum DFS code of a graph pattern in $\mathcal{G}_{\sqsubseteq}$. Furthermore, let* $\text{len} : l_{vs}(t_d t_s l_e l_{vd})^m \to \mathbb{N}$ *return the length of a minimum DFS code, i.e., zero in the case of the empty pattern $[\emptyset]$ and $m+1$, otherwise. Then, $\textsc{MinDfsTree}_{\mathcal{G}} = (V, E)$ is called minimum DFS code tree iff*

- $V = \left\{ \text{mincode}(P) \;\middle|\; P \in [\mathcal{G}]_{\sqsubseteq} \right\}$
- $E = \{\{v, w\} \in V \times V \mid v \text{ is prefix of } w \wedge \text{len}(v) + 1 = \text{len}(w)\}$

$\textsc{MinDfsTree}_{\mathcal{G}}$ is the representative subgraph pattern space as constructed by the gSpan algorithm when no support-based pruning is applied. It is rooted in the empty pattern representative. Starting with this the root vertex, gSpan constructs $\textsc{MinDfsTree}_{\mathcal{G}}$ adding frequent vertices and applying pattern extensions (cf., definitions 2.18 and 2.19) on them. Since the prefix property (cf., lemma 2.4) holds, this construction sequence aligns with the connectivity of $\textsc{MinDfsTree}_{\mathcal{G}}$. However, the recursion tree of gSpan contains additional vertices, since non-canonical pattern representatives are constructed by the pattern extension and pruned in a second step.

Definition 4.4. *Let* $\textsc{MinDfsTree}_{\mathcal{G}} = (V, E)$ *be a minimum DFS code tree. Then,* $\textsc{gSpanRecTree}_{\mathcal{G}} = (V', E')$ *is called (unpruned)* gSpan *recursion tree iff*

- $V' = V \cup \{\text{fw}(G, \cdot, \cdot) \mid G \in V\} \cup \{\text{bw}(G, \cdot, \cdot, \cdot) \mid G \in V\}$, *where* $\text{fw}(G, \cdot, \cdot)$ *and* $\text{bw}(G, \cdot, \cdot, \cdot)$ *are all restricted forward and backward extensions that are possible for a graph representative* G *(cf., definitions 2.18 and 2.19).*
- $E' = \{\{v, w\} \in V' \times V' \mid v \text{ is prefix of } w \wedge \text{len}(v) + 1 = \text{len}(w)\}$

Observation 4.10. *Given a gSpan recursion tree* $\textsc{gSpanRecTree}_{\mathcal{G}}$*, the construction of a pattern representative* code_P *depends on the constructions of ancestor pattern representatives, i.e., patterns that lay on the path from the root to* code_P *itself. Thus, the patterns lineage exploration is inherently sequential. Patterns in different branches of* $\textsc{gSpanRecTree}_{\mathcal{G}}$ *are independent and can be explored in parallel up to their common ancestor(s).*

Given this insight, a straightforward implementation could exploit the inherit parallelism of a BFS-like exploration. However, as discussed in paragraph 2.6.2.1.2, each poset rank of the pattern space may contain an exponential number of patterns w.r.t. to the rank level. Keeping all patterns of a rank together with the associated transaction lists in memory might be prohibitive for higher ranks. For this reason, the following shared-memory parallelization will use a mixed exploration strategy, that explores only as many branches as necessary to exploit a sufficient parallelism and tries to keep the DFS-like fashion as much as possible.

Algorithm 11 implements a shared-memory parallelized version of algorithm 9. It uses a priority queue `priorityWorkQueue` to store patterns in form of DFS codes, i.e., vertices of the recursion tree $\textsc{gSpanRecTree}_{\mathfrak{S}_1}$, together with a reference to the transaction lists of the parent. This queue holds the work, that can be processed in parallel with the procedure `processPattern` (line 17). More precisely, each pattern's support calculation, canonical test, pruning, extension, and evaluation is considered as an atomic unit of work. Whenever a pattern is extended, the extension is directly enqueued into the work queue (line 24). At the other end, the loop in line 7 dequeues these patterns to process them in parallel. The queue is prioritized by the size of the patterns to prefer a DFS-like exploration. The algorithm keeps track of the number of active processes (lines 10 and 26) for two reasons. First, to limit the maximum number of parallel executions. For this, the algorithm waits in line 12 until the number of active processes is below the maximum parallelism $p_{\max}$. Second, the work queue can be empty while instances of the procedure `processPattern` are still running. Thus, additional patterns might be added to the work queue in the future. In this situation, it is necessary to wait (line 12) until either all processes have terminated or additional work is added to the queue. Otherwise, the abortion criterion of the loop in line 7 may exit to early. The procedure `processPattern` is structured as follows. The transaction lists for the first and second sample ($\mathfrak{S}_{1_{\sqsupseteq P}}$ and $\mathfrak{S}_{2_{\sqsupseteq P}}$) are calculated in lines 19 to 20 based on the transaction list of the parent pattern ($\mathfrak{S}_{1_{\sqsupseteq P\downarrow}}$

and $\mathfrak{S}_{2_{\sqsupseteq P\downarrow}}$) in gSpanRecTree$_{\mathfrak{S}_1}$. The pattern exploration is pruned, if the support of a pattern is below the rejection threshold, if it is not in canonical (i.e., minimal) form, or if the subgraph pattern coverage relation specific pruning strategy returns true (line 21). Then, the rightmost extensions of the pattern (cf., paragraph 2.6.2.2.1) are calculated and added to the work queue in lines 22 to 24. Finally the pattern is evaluated w.r.t. to algorithm 10 in line 25.

4.6.6 A Non-Distributed Streaming Algorithm for MAX-CSPC

The goal of this chapter is the development of a distributed algorithm. However, the discussed sampling approach also leads to a non-distributed algorithm (cf., algorithm 12), which will be presented in this section. Besides its inherent value in a non-distributed setting, it will be utilized as a baseline to evaluate the distributed algorithm in the latter experimental evaluation (cf., section 4.6.10). Furthermore, it is useful for the evaluation of some basic properties of the sampling approach.

Algorithm 12 is an adoption of algorithm 6 for which the determination of $P_{\max}$ is replaced by the sampling approach. Thus, only the adjusted lines 5 to 12 will be discussed here. For each pattern added to the solution set S, i.e., for each iteration of the main loop, a random sample $\mathfrak{S}$ (with replacement) of size $2s$ is drawn and split into the two samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$. In comparison with two separate sampling calls for $\mathfrak{S}_1$ and $\mathfrak{S}_2$, the splitting approach requires only a single pass of $\mathcal{G}'$. Next, algorithm 9 is called in line 7 to determine the candidate set. The latter lines 8 to 11 calculate the coverage for each determined candidate with a single pass over $\mathcal{G}'$. Last, $P_{\max}$ is determined by selecting the candidate with the highest coverage in line 12.

Algorithm 12 is shared-memory parallelized. The parallelization of phase 1 was already discussed in section 4.6.5. Phase 2, or more precisely lines 9 to 11, is straightforward to parallelize by running each evaluation of $\triangleleft$ in parallel. Given a maximum parallelism $p_{\max}$, the streaming fashion over $\mathcal{G}'$ is preserved by ordering the tuples (G, C) lexicographically by the orders $\mathcal{G}'$ and $\mathfrak{C}$ are kept in memory. Thus, only the minimum number of graphs from $\mathcal{G}'$ are kept in memory that are necessary to have a sufficient parallelism.

To not clutter the code, algorithm 12 ignores the validity returned by algorithm 9 in line 7. Handling of invalid situations would be possible, e.g., by adjusting the sample size dynamically. However, the corner case of invalid results is very uncommon in practice. During the following evaluation in section 4.6.10, no single instance returned an invalid result. As such, handling this situation is a rather theoretical discussion and will be omitted in the following.

4.6.7 A Distributed Algorithm for Phase 1

This section will present a distributed algorithm for phase 1 of algorithm 7.

The discussion related to the shared-memory parallelization (cf., section 4.6.5) made clear that a distribution of the pattern space enumeration is necessary to scale with a higher number of processors or workers. As already discussed for the procedure `processPattern` of algorithm 11, each pattern's support calculation, canonical test,

Algorithm 11: Shared-Memory Parallelized Phase 1 of Algorithm 7

Input: random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ of graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$
Fixed Parameterization: pruning strategy $\texttt{pruning}_{\triangleleft}$, minimum absolute number of Bernoulli examples $\mathsf{ex}_{\min} \in \mathbb{N}^{>0}$ (success and failure) for the candidate test, maximum parallelism $p_{\max}$
Output: candidate set $\mathfrak{C}$ and boolean validity of result
Shared-Memory Variables: $R_{\max}$, $\mathsf{utility}_{R_{\max}}$, $\mathfrak{C}$, $\hat{p}_{1l}$, priorityWorkQueue
Subprocedure Of: algorithm 12

```
1  procedure phase1Parallel(𝔖1, 𝔖2, ◁, α)
2      R_max ← null;
3      utility_{R_max} ← 0;
4      𝔠 ← ∅;
5      p̂_1l ← −∞;
6      priorityWorkQueue.enqueue(((), 𝔖1, 𝔖2));
7      while priorityWorkQueue ≠ ∅ do
8          synchronized priorityWorkQueue
9              (code_P, 𝔖1_{⊒P↓}, 𝔖2_{⊒P↓}) ← priorityWorkQueue.dequeue();
                                              ► Prioritize large patterns
10         activeProcesses.atomicIncrement();
11         async processPattern(code_P, 𝔖1_{⊒P↓}, 𝔖2_{⊒P↓}, ◁, α);
12         wait until activeProcesses = 0 or (activeProcesses < p_max and
             priorityWorkQueue ≠ ∅);
13     validity ← p̂_1l ≥ ex_min;
14     if 𝔠 = ∅ then
15         𝔠 ← {(R_max, 0)};
16     return ({C | (C, utility_C) ∈ 𝔠}, validity);
17 procedure processPattern(code_P, 𝔖1_{⊒P↓}, 𝔖1_{⊒P↓}, ◁, α)
18     P ← [code_P];
19     𝔖1_{⊒P} ← {{G ∈ 𝔖1_{⊒P↓} | G ⊑ P}};
20     𝔖2_{⊒P} ← {{G ∈ 𝔖2_{⊒P↓} | G ⊑ P}};
21     if |𝔖1_{⊒P}| ≥ ⌊p̂_1l + 1⌋ and isMinimal(code_P) and not pruning_◁(P)
         then
22         foreach rightmost extension code_{P↑} of code_P do
23             synchronized priorityWorkQueue
24                 priorityWorkQueue.enqueue((code_{P↑}, 𝔖1_{⊒P}, 𝔖2_{⊒P}));
25         candidateEval(P, 𝔖1_{⊒P}, 𝔖2_{⊒P}, ◁, α);      ► cf., algorithm 10
26         activeProcesses.atomicDecrement();
```

Algorithm 12: Streaming MAX-CSPC

Input: graph dataset $\mathcal{G}$, cardinality constraint $k \in \mathbb{N}^{>0}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$, sample size $s \in \mathbb{N}^{>0}$
Output: representative pattern set S, utility of S

```
1  S ← ∅;
2  utility_sum ← 0;
3  G' ← G;
4  while |S| < k and G' ≠ ∅ do
5      𝔖 ← randomSample(G', 2s);                          ▶ With replacement
6      (𝔖_1, 𝔖_2) ← split(𝔖);                              ▶ Split in half
7      (𝔈, ·) ←phase1Parallel(𝔖_1, 𝔖_2, ◁, α);             ▶ cf., algorithm 11
8      utility[] ← (0, ..., 0);                  ▶ Initialize utilities with 0
9      foreach (G, C) ∈ G' × 𝔈 do in parallel              ▶ Stream over G'
10         if C ◁ G then
11             utility[C].atomicIncrement();
12     P_max ← arg max_{C∈𝔈} utility[C];
13     S ← S ∪ {P_max};
14     utility_sum ← utility_sum + utility[P_max];
15     G' ← G' \ G'_{▷P_max};
16 return (S, utility_sum);
```

pruning, extension, and evaluation can be bundled as a unit of work w.r.t. work distribution. However, in contrast to the shared-memory parallelization, it is expensive to share synchronized datastructures in a distributed environment. Maintaining a shared work queue as done in algorithm 11 would not scale well among multiple workers. Consequentially, it is beneficial to define a larger amount of work that can be processed independently (cf., observation 4.10) without a per-pattern workload balancing.

4.6.7.1 Related Distributed Frequent Subgraph Mining Algorithms

The above-described problem was already discussed in the literature and several distributed algorithms for problem 2.5 (FSM) are described in section 2.6.2.2. While most of these algorithms are compatible to the above-discussed sampling approach, i.e., phase 1 of algorithm 7, the algorithm presented in [LXG14] requires a predefined support threshold for the filtering phase. This is problematic since the pruning threshold $\hat{p}_{1l}$ is adjusted on the fly (cf., sections 4.6.3.6 and 4.6.3.7) and a minimum support threshold cannot be specified a priori. The other algorithms in the transactional setting—i.e., FSM-H and DIMSpan [BH15; PJR17]—partition the dataset and use a BFS exploration strategy that distributes all patterns of the current rank to all the workers. This strategy offers a large amount of work that can be processed independently on each subset of the dataset without inter-worker communication, which makes such an approach well suited for the data-centric MapReduce and Spark frameworks. Furthermore, if the dataset is partitioned randomly, the expected work per partition element is equally distributed. However, as discussed in the preliminaries (cf., paragraph 2.6.2.1.2), the combinatorial explosion of patterns on each poset rank, has led to a preference of DFS based algorithms in the sequential setting and the large number of frequent patterns on each poset rank does also limit the scalability of such distributed approaches. More precisely, a large number of workers results in a large communication overhead for pattern distribution. Additionally, each worker's memory must be large enough to hold the frequent patterns of the current rank. The second drawback could be mitigated by splitting the computation into multiple rounds, where each round processes a subset of the patterns. However, such a mitigation precludes an embedding list support counting as used by FSM-H and DIMSpan (which requires to track the embeddings of the patterns for each dataset graph) and requires a transaction list approach.

These problems are not shared by ParGraph and Fractal [TZ16; PJR17], which distribute the patterns and not the dataset among the workers. Thus, several DFS explorations are run on subsets of the pattern space similar to the shared-memory parallelization in algorithm 11. Since the partitioning of the search space into fixed subspaces does lead to an unbalanced work distribution, a work-stealing rebalancing approach is used by both algorithms. While they are proposed for the single graph setting, an adoption to the transactional setting is possible. Nevertheless, the advantage to avoid the communication of the complete pattern space for distributed support counting comes at the cost of holding the complete dataset in each worker's memory.

[LXG14] Lin, Xiao, and Ghinita, "Large-scale frequent subgraph mining in MapReduce". 2014

[BH15] Bhuiyan and Hasan, "An Iterative MapReduce Based Frequent Subgraph Mining Algorithm". 2015

[PJR17] Petermann, Junghanns, and Rahm, "DIMSpan: Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems". 2017

[TZ16] Talukder and Zaki, "Parallel graph mining with dynamic load balancing". 2016

In summary, existing approaches can be divided into two categories. They either distribute the pattern space enumeration or the dataset graphs, but not both. As discussed above, a major goal of the sampling approach for problem 4.2 (MAX-CSPC-1) is the removal of the computational dependency of $f_{\mathcal{G},\triangleleft}$ on the complete dataset $\mathcal{G}$. Instead only a sample $\mathfrak{S}$ of $\mathcal{G}$ is required to be held in memory. This makes the above-discussed pattern distribution approaches applicable to a subset of the dataset graphs and enables the distribution of work among workers which cannot hold the complete dataset in memory. To the best of my knowledge, the following representative mining algorithm is the first distributed subgraph miner, that distributes the pattern space alongside the dataset itself. The precise implementation is given in the following.

4.6.7.2 Search Space Partitioning

> **Definition 4.5.** *Let* $\textsc{gSpanRecTree}_{\mathcal{G}}(\text{code}_P)$ *be the subtree of* $\textsc{gSpanRecTree}_{\mathcal{G}}$ *rooted in* code_P*. Then,* $\textsc{gSpanRecTree}_{\mathcal{G}}(\text{code}_P)$ *is called work package rooted in graph pattern representative* code_P*.*

The recursion tree $\textsc{gSpanRecTree}_{\mathcal{G}}$ of gSpan was already subject to discussion in section 4.6.5. A work package is a connected component of the recursion tree. Thus, given the root pattern representative, the containing pattern representatives can be constructed consecutively with a DFS- or BFS-based exploration.

> **Definition 4.6.** *The work packages for two pattern representatives* code_P *and* $\text{code}_{P'}$ *are said to be independent if* $\textsc{gSpanRecTree}_{\mathcal{G}}(\text{code}_P)$ *does not contain* $\text{code}_{P'}$ *and vice versa.*

Thus, similar to observation 4.10, two independent work packages do not overlap and define a distinct amount of work that can be processed independently w.r.t. pattern exploration. Given a bottom-up pattern exploration, the exploration starts with a single (empty) pattern. Consequentially, a large enough set of independent work packages have to be calculated in a preprocessing step to distribute the work among the workers. From an algorithmic point of view, this can be done with a pruned bottom-up exploration (e.g., DFS or BFS) of $\textsc{gSpanRecTree}_{\mathcal{G}}$ starting at the root of the recursion tree. Then, the pruned branches are independent work packages. Figure 4.13 shows the partitioning of the search tree given such a pruned exploration.

4.6.7.3 A Simple Distributed Algorithm for Phase 1 with Fixed Work Partitioning

Given the definition of independent work packages (cf., definition 4.6), algorithm 13 describes a simple distribution scheme for a fixed set of work packages in Spark. In a first step, algorithm 13 partitions the work packages (line 2) as described in section 4.6.7.2 by running a pruned bottom-up pattern exploration starting at the empty pattern. Since the work packages obtained in such a way do not include the patterns explored in the preprocessing step, the candidate evaluation as described in algorithm 10 needs to be applied to these patterns. For this reason, a set of candidates

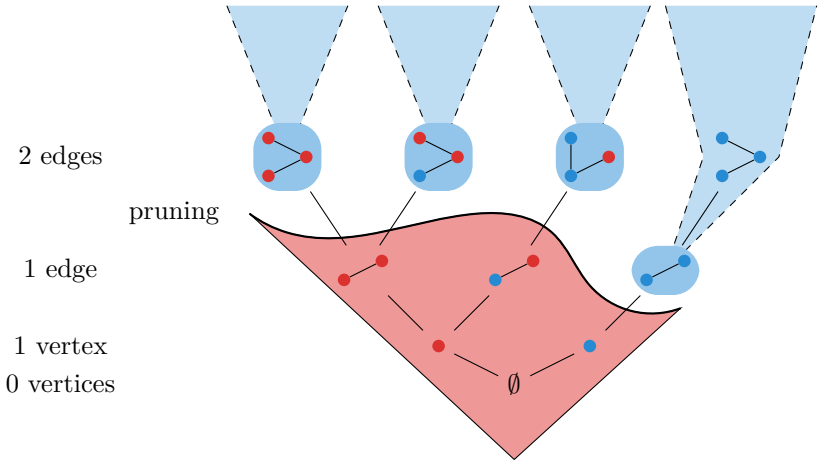


Figure 4.13: The gSpan recursion tree (non-canonical pattern representations are omitted for the sake of simplicity) partitioned into independent work packages. The red area indicates a bottom-up exploration of the pattern space with some pruning applied. The blue subtrees are independent work packages defined by the subtrees root (blue squircles).

Algorithm 13: Fixed Work Distribution for Phase 1 of Algorithm 7 in Spark

Input: graph dataset $\mathcal{G}$, sample fraction $s \in \mathbb{Q}^{\geq 0, \leq 1}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$, number or workers W

Output: candidate set $\mathfrak{C}$

```
1 sampleRdd ← RDD(𝒢)
      .sampleExact(2sW, true)
      .repartition(W);
2 (work, 𝔠) ← initialWork(sampleRdd, ◁, α);
3 workRdd ← RDD(work).repartition(W);
4 𝔠 ← 𝔠 ∪ sampleRdd
      .zipPartitions(workRdd, (s, w) → (({{s}}, {w})))
      .map((𝔖, pwork) → phase1Rooted(𝔖, pwork, ◁, α))
      .collect();
5 return 𝔠;
```

is returned alongside the work packages in line 2. In a second step, algorithm 13 runs a local instance of a subgraph pattern mining algorithm for problem 4.2 (MAX-CSPC-1) on each worker. For this, the procedure `phase1Parallel` of algorithm 11 was slightly modified (named `phase1Rooted`) to start the exploration at the root of each assigned work package. More precisely, `phase1Rooted` loops over all assigned work packages and replaces the empty DFS code in line 6 of algorithm 11 with the work packages root.

Lemma 4.11. *Algorithm 13 is correct w.r.t. corollary 4.7.*

In other words, algorithm 13 returns a pattern of maximum utility with a probability of at least $1 - \alpha$.

Proof. Let I be the set of local instances of algorithm 11, that are run by algorithm 13. This includes one instance for each RDD partition as well as the one instance for the initial work calculation. Each instance $i \in I$ is restricted to a subset $S_i \subseteq \mathcal{G}_{\sqsubseteq}$ of the pattern space and calculates a candidate set $\mathfrak{C}_i$ w.r.t. the problem $P_{\max,i} := \arg\max_{P \in S_i} f_{\mathcal{G},\triangleleft}(\{P\})$. The analysis in lemma 4.6 still applies to this restricted problem and $\mathfrak{C}_i$ will contain $P_{\max,i}$ with probability $1 - \alpha$. Since $\mathcal{G}_{\sqsubseteq} = \bigcup_{i \in I} S_i$ holds, $P_{\max}$ is contained in one of the subsets S_i for $i \in I$ of the subgraph pattern space. Let $S_{P_{\max}}$ be this subset. Then, $P_{\max}$ is contained in $S_{P_{\max}} \subseteq \mathfrak{C}_{P_{\max}}$ with probability $1 - \alpha$. The applicability of lemma 4.6 implies corollary 4.7. □

Algorithm 13 does have two major weaknesses compared to the shared-memory parallelization. First, the local reference pattern and rejection threshold does lead to a less efficient support-based pruning and larger candidate sets. Second, as already mentioned in the review of existing distributed FSM algorithms, an unbalanced work distribution can lead to a poor worker utilization and scaling behavior. These two weaknesses will be addressed in the following two sections (cf., sections 4.6.7.4 and 4.6.7.5).

4.6.7.4 Sharing the Rejection Threshold

In contrast to the shared-memory parallelization, the reference pattern and the rejection threshold $\hat{p}_{1l}$ are not shared among the workers. Given the analysis of lemma 4.11, an instance i is restricted to a subset $S_i \subseteq \mathcal{G}_{\sqsubseteq}$ of the pattern space. As a consequence, the local optimal pattern $P_{\max,i}$ might have a lower utility than the global optimal pattern $P_{\max}$. In this case, the values of the expected utility of the local reference pattern $R_{max,i}$ and the expected local rejection threshold $\hat{p}_{1l,i}$ are lower than the global values. Ultimately, this leads to a less efficient support-based pruning of the search space and larger candidate sets.

Observation 4.12. *Given two distinct instances $i, j \in I$ with independent, equally sized random samples $\mathfrak{S}_{1,i}$, $\mathfrak{S}_{2,i}$, $\mathfrak{S}_{1,j}$, and $\mathfrak{S}_{2,j}$, $\hat{p}_{1l,i}$ is a valid rejection threshold for instance j and vice versa.*

The comparison of a candidate pattern with a reference pattern of another instance aligns with the computation of the reference pattern in a non-distributed setting with the exception, that $\mathfrak{S}_{2,i}$ and $\mathfrak{S}_{2,j}$ are not identical. However, the candidate test only assumes, that the test statistic of the candidate pattern and the reference pattern are independent. In fact, the samples in the non-distributed setting are only re-used for multiple tests, since this lowers the computational complexity of the algorithm. It is thereby possible to share the rejection threshold among the workers and mitigate the weaknesses of local reference patterns. The Spark implementation is discussed in section 4.6.7.6.

4.6.7.5 Work Balancing

Algorithm 13 does not balance the work after the initial partitioning of the work packages. However, the work distribution of different work packages is usually skewed, which leads to a poor worker utilization for a fixed work package partitioning. There are several reasons for this skewness. For example, Dias et al. [Dia+19] mention irregular degree distributions in the single graph setting and Talukder and Zaki [TZ16] name different depths of the branches of the search tree. Since the subgraph isomorphism complexity does scale exponentially with the size of a pattern in the worst case, even small differences in the depths can result in a huge difference w.r.t. the enumeration performance. Additionally, the canonization of pattern representatives in the search tree is another source for a skewed distribution w.r.t. the number of patterns in each branch of MinDfsTree$_{\mathcal{G}}$. As discussed in paragraph 2.6.2.2.1 the canonization of gSpan is based on sorting vertices and edges w.r.t. their labels. Let P and P' be two patterns of the same rank with minimum DFS code code$_P$ and code$_{P'}$ and let code$_P$ < code$_{P'}$. Then, as a result of the prefix property (cf., lemma 2.4), the subtree of MinDfsTree$_{\mathcal{G}}$ rooted in code$_{P'}$ does not contain any pattern $Q \sqsupseteq P$. Thus, given the minimum DFS code ordering of patterns of a fixed rank, more and more patterns are excluded in the subtrees of MinDfsTree$_{\mathcal{G}}$ rooted in higher-ordered patterns. This can lead to a very unbalanced recursion tree under some circumstances. For example, in chemistry, carbon atoms are so frequent that they are the majority of atoms and are contained in nearly all molecules. If the carbon label is sorted first in this setting, this can lead to an accumulation of the majority of the frequent patterns in a single work package rooted at the pattern with a single carbon atom.

[Dia+19] Dias et al. "Fractal: A General-Purpose Graph Pattern Mining System". 2019

[TZ16] Talukder and Zaki "Parallel graph mining with dynamic load balancing". 2016

> **Observation 4.13.** *The patterns in* `priorityWorkQueue` *of algorithm 11 define independent work packages.*

In retrospect, the shared-memory parallelization already keeps independent work packages in the work queue. The discussion w.r.t. the initial work calculation for algorithm 13 with a bottom-up exploration (as depicted in fig. 4.13) already showed, how independent work packages can be obtained. Of course, this method still works, when the root of the bottom-up exploration is not the empty pattern, but the root of a work package. Since all parent patterns of the patterns in the work queue are already

evaluated, the work queue actually represents the front of a combined bottom-up exploration of the recursion tree.

In contrast to the fixed partitioned distributed algorithm 13, the work packages in algorithm 11 are not explored entirely, but intermediate work packages are directly fed back to the global work queue. On the one hand—as discussed before—this fine-grained work synchronization among the different workers is not desired in the distributed setting since it causes a huge synchronization overhead. On the other hand, the other extreme—of no work synchronization after an initial work partitioning—leads to a poor worker utilization. Thereby, it is beneficial to synchronize work only when needed, i.e., if workers are idle. In this situation, a part of the work packages in `priorityWorkQueue` of the active (non-idle) workers can be assigned to the idle ones in a work-stealing fashion.

4.6.7.6 An Optimized Distributed Algorithm for Phase 1

This section presents an optimized distributed algorithm (cf., algorithm 16) for phase 1 of algorithm 7, which is built on top of algorithm 13 and integrates a shared rejection threshold (cf., section 4.6.7.4) and work balancing (cf., section 4.6.7.5). Algorithm 16 is split into the subprocedures given in algorithms 14 and 15, which are adoptions of the shared-memory parallelized algorithms 10 and 11.

Both of the above optimizations are based on worker to worker communication. As discussed above, communication may be expensive in a distributed environment, especially if shared values need to be synchronized frequently in a blocking fashion. In fact, Spark does not have any mechanism to communicate between workers during the runtime of a spark job. Thus, using Spark alone would require to stop all the instances, collect their local state (i.e., the rejection threshold and the work queue) in the master, combine the local states to a global one, and resume the instances with w.r.t. the global state. While Spark can keep the local instances in the worker's memory without persisting the intermediate results, the synchronization over the master is still expensive. This is especially true since Spark does not provide an asynchronous collection of partial results from submitted jobs. Thus, it is not possible to end an instance as soon as a new rejection threshold is found and have the result immediately available in the master. Instead, the master has to wait until all other instances are finished to retrieve the updated rejections threshold. This also means that it is impossible to stop instances depending on events that occur in other instances. Consequentially, a pure Spark implementation would need to implement polling with a pre-defined polling interval. Such an interval is a performance-critical parameter that will undoubtedly result in unnecessary synchronizations and/or periods in which some local instances are idle or have out-of-date pruning bounds. In terms of the shared rejection threshold, this means a less efficient pruning power than possible. In terms of work balancing, this means a poor worker utilization. For this reason, the latter implementation will use the distributed Apache Ignite[10] database to share global, i.e., distributed, information. Ignite provides ACID[11]-conform transactions

[10] `https://ignite.apache.org/`

[11] ACID: atomicity, consistency, isolation, durability

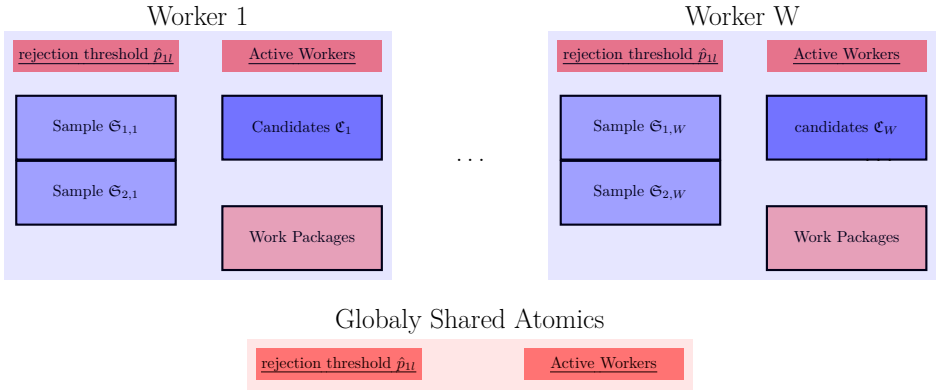


Figure 4.14: Layout of the main datastructures for the distributed computation of problem 4.2 (MAX-CSPC-1). Blue boxes indicate worker local datastructures. Red datastructures are subject to inter-worker synchronization or balancing. Distributed atomic variable names are depicted by underlined text.

as well as atomic updates of distributed variables, guaranteeing race condition free non-blocking updates.

A reason to not allow any communication between workers in Spark is, that Sparks fault tolerance is built on the immutability and reproducibility of intermediate results. This allows the spark job scheduler to re-compute results tracking the tasks lineage. After loosing a worker due to some failure, it can re-compute results based on still intact intermediate results (cf., section 2.2.4.2). Consequentially, sharing values between spark jobs can break this property if not used carefully. The same is true for randomized algorithms, where intermediate results are not deterministic. Algorithm 16 will preserve fault tolerance if intermediate candidate sets of each partial exploration instance is persisted during work balancing.

Figure 4.14 gives an overview of the layout of the main datastructures of algorithm 16. Each worker has two local independent samples from the dataset and a local candidate set. Additionally, each worker has a set of local work packages which are subject to work balancing. Finally, each worker has a local view of globally distributed atomic variables to share the pruning threshold and the total number of active workers for work balancing.

Given the fact that the rejection threshold is monotonically increasing, the workers can asynchronously update their local rejection threshold with the global value or update the global value, whenever the two differ from each other. The local exploration can run without a blocking rejection threshold synchronization. Whenever the rejection threshold increases, i.e., by a local update or by an external update from another worker, the candidates are filtered accordingly. Algorithm 14 is an adoption of

Algorithm 14: Shared-Memory Parallelized Candidate Evaluation with Distributed Rejection Threshold Sharing

Input: subgraph pattern $P \in \mathcal{G}_{\sqsubseteq}$ of a graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\lhd$, supporting graphs $\mathfrak{S}_{1_{\sqsupseteq P}}$ and $\mathfrak{S}_{2_{\sqsupseteq P}}$ of the random samples $\mathfrak{S}_1$ and $\mathfrak{S}_2$ drawn from $\mathcal{G}$, confidence level $\alpha \in [0, 0.5]$
Fixed Parameterization: minimum absolute number of Bernoulli examples $\mathsf{ex}_{\min} \in \mathbb{N}^{>0}$ (success and failure) for the candidate test
Shared-Memory Variables: $R_{\max}$, $\mathsf{utility}_{R_{\max}}$, $\mathfrak{C}$, $\hat{p}_{1l}$, $\hat{p}_{1l,\text{oldglobal}}$
Distributed Atomics: $\hat{p}_{1l,\text{global}}$
Subprocedure Of: algorithm 15

```
1  procedure candidateEvalDistrib(P, 𝔖1_⊒P, 𝔖2_⊒P, ◁, α)
2      utility_1 ← f_{𝔖1_⊒P, ◁}({P});
3      utility_2 ← f_{𝔖2_⊒P, ◁}({P});
4      if utility_2 > utility_{R_max}; then
5          synchronized
6              if utility_2 > utility_{R_max}; then
7                  R_max ← P;
8                  utility_{R_max} ← utility_2;
9                  utility_{R_max,valid} ← min{utility_2, |𝔖2| − ex_min − 1};
10                 p̂_1l ← rejectionThresh(utility_{R_max,valid}, |𝔖2|, α);
                                                    ▶ cf., lemma 4.8
11         if p̂_1l > p̂_1l,global then
12             async updateGlobalRejectionThresh(p̂_1l);
13     if p̂_1l,global > p̂_1l,oldglobal then
14         synchronized p̂_1l,oldglobal
15             if p̂_1l,global > p̂_1l,oldglobal then
16                 p̂_1l,oldglobal ← p̂_1l,global;
17                 synchronized 𝔠
18                     𝔠 ← {(C, utility_C) ∈ 𝔠 | utility_C ≥ p̂_1l,global};
19     if utility_1 ≥ p̂_1l,global or p̂_1l,global < ex_min or utility_1 ≥ |𝔖1| − ex_min then
20         synchronized 𝔠
21             𝔠 ← 𝔠 ∪ (P, utility_1);
```

algorithm 10 which implements rejection threshold sharing between different instances. The variable $\hat{p}_{1l}$ is now split into three different variables: (a) $\hat{p}_{1l}$, which remains the rejection threshold associated with the local reference pattern, (b) $\hat{p}_{1l,\text{global}}$, which is a local view of the globally shared rejection threshold, i.e., the maximum of all local rejection thresholds $\hat{p}_{1l}$, and (c) $\hat{p}_{1l,\text{oldglobal}}$ which is a local copy of $\hat{p}_{1l,\text{global}}$ to track changes and act accordingly. While $\hat{p}_{1l,\text{global}}$ is a view of the global Ignite atomic, reading the value of $\hat{p}_{1l,\text{global}}$ is a local non-blocking operation, that does not involve communication with the Ignite instance. Instead, global changes of the Ignite instance are asynchronously pushed to the workers, i.e., the update occurs in a separate thread. Thereby $\hat{p}_{1l,\text{global}}$ may increase its value at any time. This has the effect, that comparisons for conditional executions may be outdated when the value is used in some operation (cf., lines 12 and 18) or that composed conditionals (cf., line 19) may use mixed values. In line 12, the global rejection threshold is updated with a larger local value in the procedure `updateGlobalRejectionThresh`. This update needs to take care, that it never updates the global value based on a comparison with an outdated view. Ignite provides atomic updates, that do not only have the update value as parameter, but also the expected global value. Whenever the global value differs from the expected value, the update is not performed. In such a situation, it is possible to re-check the condition against the updated global value and repeat until the update succeeds or the local value is no longer larger than the global one. Since this checked update needs to block in each iteration until the result is available to the worker, the whole procedure is performed asynchronously in a separate thread (async keyword). The local view of the global value $\hat{p}_{1l,\text{global}}$ is directly increased to the local update value and is increased in a later step if the global value was changed to a higher value concurrently. In line 18, the conditional execution only serves performance reasons, i.e., the candidate list only needs to be filtered w.r.t. to outdated candidates, whenever the rejection threshold actually changes. Thus, filtering with a better (i.e., in between increased) threshold is still a correct operation and would have been performed in a future iteration otherwise. A similar effect can be observed for line 19. Using an updated rejection threshold for some of the comparisons may lead to a rejection of the pattern. Anyway, this pattern would have been filtered from the candidate list in the next iteration. The only situation where the asynchronous update may cause different results is when the global value is updated after the local worker has performed the last candidate list filtering in line 18. Thus, the candidates set may contain some outdated candidates below the rejection threshold. This case will be handled by introducing an extra filtering step after all workers have finished their work in algorithm 16 (line 13). In comparison with algorithm 10, the filtering of the candidate set is now based on changes to the global rejection threshold. As discussed for the update of the reference pattern in line 6, the shared-memory synchronization is only performed if an updated value is detected to avoid lock contention, which requires a second identical comparison to rule out concurrent modifications.

In contrast to the asynchronous update of the rejection threshold, work balancing is performed with dedicated Spark synchronization points. Whenever a worker becomes idle, it decrements the active worker's atomic. If the total number of active workers falls below a certain threshold, the Spark workers will halt their computation and

return the remaining work packages to the master. The master then applies a re-distribution strategy to assign the work packages to the workers and continues the computation. The reason to not share the work packages via ignite is fault tolerance. When the work packages and the candidate sets are persisted at synchronization points, Spark can re-schedule failed tasks on workers. Persisting intermediate results is necessary since the worker's output is non-deterministic if it is halted before the work packages are fully processed. This is a side effect of the non-deterministic computation speeds in combination with the share-memory synchronizations and the timed interruption of the spark tasks.

Algorithm 16 is an adoption of algorithm 13 which implements work balancing. It uses algorithm 15 (an adoption of algorithm 11) to partially explore work packages. Thus, only changes w.r.t. algorithm 15 are discussed for algorithm 15. Similar to `phase1Rooted`, algorithm 15 starts the exploration at the root of the work packages but also implements premature termination for work balancing. To start a local computation, algorithm 15 picks up a state to resume from, which is the local reference pattern $R_{\max}$, its utility $\text{utility}_{R_{\max}}$, and the candidate set $\mathfrak{C}$. Additionally, the assigned work packages are added to the work queue in lines 8 and 9 to initialize the algorithm. Since these work packages may come from another worker with different samples and a different gSpanRecTree the old parent transaction lists of the work packages root are no longer valid. Consequentially, the complete samples replace the parent transaction lists for work packages. To implement the premature termination for work balancing, the work loop in line 10 does exit whenever the number of active workers falls below a specified minimum. To avoid situations in which synchronization occurs too frequently, it is possible ot specify a minimum number of iterations before the work balancing criterion becomes effective. If the computation is terminated w.r.t. work balancing, the work loop may exit while there are still active processes that may lead to additions to the candidate set $\mathfrak{C}$. Thus, it is necessary to add another wait statement after the work loop in line 16. Before termination, `phase1Partial` does decrement the distributed variable `activeWorkers` line 17. Finally, algorithm 15 returns the remaining work alongside the internal state to continue the work in the next round of the partial pattern exploration.

Algorithm 16 actually distributes the work among the workers and is implemented using Spark. First, it samples[12] the `datasetRdd` in line 2. This RDD represents the distributed graph dataset, that must be provided by the calling algorithm. The sample is partitioned, such that each worker w has local access to a sample $\mathfrak{S}_w$ of two times the single sample size, which is then split into the samples $\mathfrak{S}_{1,w}$ and $\mathfrak{S}_{2,w}$ with size s in algorithm 15. Furthermore, the algorithm will maintain a fixed partitioning, such that the samples are never moved over the network to another worker. Later, each worker will execute one instance of the partial miner, i.e., algorithm 15. The RDD `resultRdd` will store the result of the partial miner. More precisely, this is the internal state of the miner to resume the work after work balancing and the unprocessed work packages which can be re-distributed among the workers. The `resultRdd` is initialized by the procedure `initialWork` in line 4. This procedure follows the same

[12]The boolean parameter of `sampleExact` causes a sample with replacement (cf., section 2.2.4.2).

Algorithm 15: Shared-Memory Parallelized Partial Search Space Exploration for Phase 1 of Algorithm 7 with Premature Termination for Work Balancing

Input: random sample $\mathfrak{S}$ of graph dataset $\mathcal{G}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$, work packages work, resume state state $= (R_{\max,\text{init}}, \text{utility}_{R_{\max,\text{init}}}, \mathfrak{C}_{\text{init}})$

Fixed Parameterization: pruning strategy $\texttt{pruning}_{\triangleleft}$, minimum number of iteration before premature termination $\text{iterations}_{\min}$, the minimum number of active workers $\text{activeWorkers}_{\min}$, maximum shared-memory parallelism $p_{\max}$

Output: resume state state, remaining work packages

Shared-Memory Variables: $\text{utility}_{R_{\max}}$, $\mathfrak{C}$, $\hat{p}_{1l}$, priorityWorkQueue, activeProcesses

Distributed Atomics: $\hat{p}_{1l,\text{global}}$, activeWorkers

Subprocedure Of: algorithm 16

1 **procedure** `phase1Partial`($\mathfrak{S}$, $\triangleleft$, α, work, state $= (R_{\max,\text{init}}, \text{utility}_{R_{\max,\text{init}}}, \mathfrak{C}_{\text{init}})$)
2 $(\mathfrak{S}_1, \mathfrak{S}_2) \leftarrow$ `split`($\mathfrak{S}$);
3 $R_{\max} \leftarrow R_{\max,\text{init}}$;
4 $\text{utility}_{R_{\max}} \leftarrow \text{utility}_{R_{\max,\text{init}}}$;
5 $\mathfrak{C} \leftarrow \mathfrak{C}_{\text{init}}$;
6 $\hat{p}_{1l} \leftarrow -\infty$;
7 activeProcesses $\leftarrow 0$;
8 **for** $\text{code}_{\text{wproot}} \in$ work **do**
9 priorityWorkQueue.`enqueue`($(\text{code}_{\text{wproot}}, \mathfrak{S}_1, \mathfrak{S}_2)$);
10 **while** priorityWorkQueue $\neq \emptyset$ **and** ($\text{activeWorkers}_{\min} <$ activeWorkers **or** $\text{iterations}_{\min} <$ iterations) **do**
11 **synchronized** priorityWorkQueue
12 $(\text{code}_P, \mathfrak{S}_{1_{\sqsupseteq P\downarrow}}, \mathfrak{S}_{2_{\sqsupseteq P\downarrow}}) \leftarrow$ priorityWorkQueue.`dequeue`();
► `Prioritize large patterns`
13 activeProcesses.`atomicIncrement`();
14 **async** `processPattern`($\text{code}_P, \mathfrak{S}_{1_{\sqsupseteq P\downarrow}}, \mathfrak{S}_{2_{\sqsupseteq P\downarrow}}, \triangleleft, \alpha$);
15 **wait until** activeProcesses $= 0$ **or** (activeProcesses $< p_{\max}$ **and** priorityWorkQueue $\neq ()$);
16 **wait until** activeProcesses $= 0$;
17 activeWorkers.`atomicDecrement`();
18 **return** $((R_{\max}, \text{utility}_{R_{\max}}, \mathfrak{C}),$ priorityWorkQueue);
19 **procedure** `processPattern`(code_P, $\mathfrak{S}_{1_{\sqsupseteq P\downarrow}}$, $\mathfrak{S}_{1_{\sqsupseteq P\downarrow}}$, $\triangleleft$, α)
20 $P \leftarrow [\text{code}_P]$;
21 $\mathfrak{S}_{1_{\sqsupseteq P}} \leftarrow \{\!\{G \in \mathfrak{S}_{1_{\sqsupseteq P\downarrow}} \mid G \sqsubseteq P\}\!\}$;
22 $\mathfrak{S}_{2_{\sqsupseteq P}} \leftarrow \{\!\{G \in \mathfrak{S}_{2_{\sqsupseteq P\downarrow}} \mid G \sqsubseteq P\}\!\}$;
23 **if** $\left|\mathfrak{S}_{1_{\sqsupseteq P}}\right| \geq \lfloor \hat{p}_{1l,\text{global}} + 1 \rfloor$ **and** `isMinimal`(code_P) **and not** $\texttt{pruning}_{\triangleleft}(P)$ **then**
24 **foreach** rightmost extension $\text{code}_{P\uparrow}$ of code_P **do**
25 **synchronized** priorityWorkQueue
26 priorityWorkQueue.`enqueue`($(\text{code}_{P\uparrow}, \mathfrak{S}_{1_{\sqsupseteq P}}, \mathfrak{S}_{2_{\sqsupseteq P}})$);
27 `candidateEvalDistrib`($P, \mathfrak{S}_{1_{\sqsupseteq P}}, \mathfrak{S}_{2_{\sqsupseteq P}}, \triangleleft, \alpha$);
► `cf., algorithm 14`
28 activeProcesses.`atomicDecrement`();

Algorithm 16: Optimized Distributed Algorithm for Phase 1 of Algorithm 7 in Spark

Input: graph dataset RDD datasetRdd (of some graph dataset $\mathcal{G}$), subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$, number or workers W, sample fraction $s \in \mathbb{Q}^{\geq 0, \leq 1}$

Fixed Parameterization: pruning strategy $\texttt{pruning}_{\triangleleft}$, minimum number of iteration before premature termination $\text{iterations}_{\min}$, the minimum number of active workers $\text{activeWorkers}_{\min}$

Output: candidate set $\mathfrak{C}$ and boolean validity of result

Distributed Atomics: $\hat{p}_{1l,\text{global}}$, activeWorkers

Subprocedure Of: algorithm 17

```
1  procedure phase1Distrib(datasetRdd, ◁, α, W, s)
2      p̂_{1l,global} ← −∞;
3      sampleRdd ← datasetRdd
           .sampleExact(2sW|G|, true)
           .repartition(W)
           .persist();
4      resultRdd ← initialWork(sampleRdd, ◁, α)
           .repartition(W)
           .persist();                          ▶ resultRdd of type (state, work)
5      work ← resultRdd.map((state, pwork) → pwork).collect();
6      while work ≠ ∅ do
7          activeWorkers ← W;
8          workRdd ← distributeWork(work, W);
9          prevResultRdd ← resultRdd;
10         resultRdd ← prevResultRdd.map((state, pwork) → state)
               .zipPartitions((sampleRdd, workRdd), zipFunc())
               .map((state, 𝔖, pwork) → phase1Partial(𝔖, ◁, α, pwork, state))
               .persist();
11         work ← resultRdd.map((state, pwork) → pwork).collect();
12         prevResultRdd.unpersist();
13     sync(p̂_{1l,global});          ▶ Make sure the local view is up to date
14     𝔠 ← resultRdd.map((state, pwork) → state.𝔠)
           .filter((C, utility_C) → utility_C ≥ p̂_{1l,global})
           .map((C, utility_C) → C)
           .collect();
15     if 𝔠 = ∅ then
16         𝔠 ← resultRdd.map((state, pwork) → state)
               .max(compareBy(state.utility_{R_max}))
               .map(state → state.R_max)
               .collect();
17     resultRdd.unpersist();
18     return 𝔠;
19 procedure zipFunc()
20     return
         (stateSeq, sampleSeq, workSeq) → (stateSeq[0], {{sampleSeq}}, {workSeq});
```

logic as the simple distribution scheme of algorithm 13. Thus, a single instance of algorithm 15 is executed on a single worker with a special pruning criterion to create initial work packages. The `resultRdd` will only contain a single result tuple although it is partitioned among the workers. Thus, all other workers with empty partition element will resume from an empty state in the following. The algorithm then collects the work in the master (line 5 and line 11 in later iterations) and re-distributes it with the procedure `distributeWork` in line 8, i.e., it creates an RDD `workRdd` with W partitions. With these datastructures at hand, the actual distributed pattern exploration and discovery is performed in line 10. For this, the miners' state of the previous round (or pre-loop initialization) is combined with the samples and the distributed work via `zipPartitions` and fed to an instance of algorithm 15 on each worker. It is important to understand, that the zip operation does not involve any movement of the state or sample across the network, since the combination occurs locally on each worker, i.e., partition element. This process is repeated until no further work packages are returned by the miners, i.e., the search space has been explored completely. As discussed above, the premature termination of partial instances may result in candidate lists with outdated candidates. Thus, a final filtering step is performed in line 13 before collecting the resulting candidates. In alignment with algorithm 8, the global maximal reference pattern is added to $\mathfrak{C}$ whenever $\mathfrak{C}$ is empty. Last, the candidate list is returned.

Intermediate results of algorithm 16 are persisted via the procedure `persist` and unpersisted via `unpersist` (cf., section 2.2.4.2). Otherwise, intermediate results would have to be re-computed if accessed multiple times. It is important to recall, that RDDs are lazily evaluated. Thus, data is only persisted after an action is performed. For example, line 10 does not cause any distributed computation. Instead, the computation is only performed after calling the `collect` action in line 11. Therefore, unpersisting an RDD must be handled with care, especially in the case of succeeding RDDs. More precisely, succeeding RDDs must have been evaluated and persisted before unpersisting the precursor.

4.6.8 A Distributed Algorithm for MAX-CSPC

Given the distributed algorithm 16 for phase 1 of algorithm 7, this section will describe the distributed algorithm 17 for problem 4.1 (MAX-CSPC). Since phase 2 is basically the distributed computation of sums, there is no dedicated section to describe it. Instead, this section will directly focus on the implementation, which is an adoption of algorithm 12 in the distributed setting.

In a fist step, algorithm 17 creates an RDD of the dataset. Reading the dataset in Spark is a distributed operation on a shared file system. Thus, $\mathcal{G}$ is just a lightweight reference to the complete dataset and the dataset does not have to be held in memory as a whole. Instead, Spark may split the dataset into small chunks that fit into the worker's memory and process the data in a streaming-like fashion. If the memory is

Algorithm 17: Distributed Algorithm for MAX-CSPC in Spark

Input: graph dataset $\mathcal{G}$, cardinality constraint $k \in \mathbb{N}^{\geq 1}$, subgraph pattern coverage relation $\triangleleft$, confidence level $\alpha \in [0, 0.5]$, number or workers W, sample fraction $s \in \mathbb{Q}^{\geq 0, \leq 1}$

Fixed Parameterization: pruning strategy $\texttt{pruning}_{\triangleleft}$, minimum number of iteration before premature termination $\mathsf{iterations}_{\min}$, the minimum number of active workers $\mathsf{activeWorkers}_{\min}$

Output: candidate set $\mathfrak{C}$ and boolean validity of result

Distributed Atomics: $\hat{p}_{1l,\text{global}}$, $\mathsf{activeWorkers}$

$S \leftarrow \emptyset$;
$\mathsf{utility}_{\mathsf{sum}} \leftarrow 0$;
$\mathsf{datasetRdd} \leftarrow \texttt{RDD}(\mathcal{G})\texttt{.persist()}$;
$\mathsf{coverRdd} \leftarrow \texttt{RDD}(\emptyset)$;
while $|S| < k$ **and** $\mathsf{datasetRdd}\texttt{.count()} \neq 0$ **do**
  $\mathsf{coverRdd}\texttt{.unpersist()}$;
  $\mathfrak{C} \leftarrow \texttt{indexedSet(phase1Distrib(}\mathsf{datasetRdd}, \triangleleft, \alpha, W, s\texttt{))}$;
    ▶ cf., algorithm 16
  $\texttt{broadcast(}\mathfrak{C}\texttt{)}$;
  $\mathsf{coverRdd} \leftarrow \mathsf{datasetRdd}\texttt{.map(}G \to (G, \texttt{coverVec(}G\texttt{))}\texttt{).persist()}$;
  $\mathsf{utilities} \leftarrow \mathsf{coverRdd}\texttt{.map(}(G, V) \to V\texttt{).reduce(}(V_1, V_2) \to V_1 + V2\texttt{)}$;
    ▶ V is vector; reduce uses vector algebra
  $\mathsf{datasetRdd}\texttt{.unpersist()}$;
  $i \leftarrow \arg\max_{i \in \mathbb{N}^{<|\mathfrak{C}|}} \mathsf{utilities}[i]$;
  $S \leftarrow S \cup \{\mathfrak{C}[i]\}$;
  $\mathsf{utility}_{\mathsf{sum}} \leftarrow \mathsf{utility}_{\mathsf{sum}} + \mathsf{utilities}[i]$;
  $\mathsf{datasetRdd} \leftarrow \mathsf{coverRdd}\texttt{.filter(}(G, V) \to V[i] = 0\texttt{)}$
    $\texttt{.map(}(G, V) \to G\texttt{)}$
    $\texttt{.persist()}$;
$\mathsf{datasetRdd}\texttt{.unpersist()}$;
$\mathsf{coverRdd}\texttt{.unpersist()}$;
return $(S, \mathsf{utility}_{\mathsf{sum}})$;
procedure $\texttt{coverVec}(G)$
  return $(\mathbf{1}_{\triangleleft}((\mathfrak{C}[0], G)), \mathbf{1}_{\triangleleft}((\mathfrak{C}[1], G)), \ldots, \mathbf{1}_{\triangleleft}((\mathfrak{C}[|\mathfrak{C}| - 1], G)))$;

not sufficient, intermediate results are persisted to disk[13]. The loop in line 5 then performs the iterative approximation algorithm as described in eq. (4.2) and Lines 7 to 13 solve problem 4.2 (MAX-CSPC-1) with probability $1-\alpha$. For this, the candidate set is computed by algorithm 16 in line 7. The set is indexed, such that a candidate pattern C_i can be identified by its index i and broadcasted to all the Spark workers in line 8. Broadcasting is a more efficient way to copy data to all workers, than providing the data as an argument to a function call, since it uses a tree-like distribution strategy and does not send the data from the master to all workers directly. Furthermore, broadcasted data is accessible by all tasks of the worker, avoiding redundancy in case of multiple tasks per worker. After broadcasting the candidates, a cover vector is computed for each dataset graph G in line 9, which contains a 1 at position i if $C_i \triangleleft G$. The vectors are summed up in line 10 to compute the overall utilities for each candidate. The index of the candidate with maximum utility $P_{\max_{\mathfrak{C}}}$ is computed in line 12, such that $P_{\max_{\mathfrak{C}}}$ can be added to S in line 13 and its utility can be added to the overall utility $\text{utiltiy}_{\text{sum}}$ in line 14. In a last step of the loop, the dataset is filtered to the uncovered graphs in line 15. Finally, the solution set S and its utility are returned in line 18

4.6.9 Analysis

4.6.9.1 Error Accumulation over Multiple Iterations

Since, the maximum error probability α (cf., corollary 4.7) is given for a single iteration of algorithms 12 and 17, the probability of multiple errors increases for larger solution sets or iterations (parameter k) respectively. To bound the overall error probability, it is, therefore, necessary to apply a multiple hypothesis testing correction w.r.t. k. A straightforward solution would be to apply the Bonferroni correction (cf., section 2.3.4.1) and use $\alpha' = \frac{\alpha}{k}$ as the corrected parameter.

In the worst case, it may happen, that the utility of an erroneous pattern returned by algorithms 12 and 17 is equal to 1. Since the random samples are drawn with replacement only a single dataset graph may be contained in the sample s times. Thus, in theory, this worst case can be observed for any dataset. Given this worst-case scenario, an overall utility of at least $(1 - \frac{1}{e} + \epsilon)\text{OPT}_{k'}$ can still be expected, where $k' := (1-\alpha)k$ is the number of expected correct results and $OPT_{k'}$ is the optimal utility w.r.t. a solution size of $\lfloor k' \rfloor$. Such a solution is always possible if erroneous patterns are simply ignored during analysis. In practice, however, it is extremely unlikely that the above-discussed worst case actually occurs. The nature of the binomial test makes it much more likely, that patterns with close to the optimal utility are chosen than patterns with relatively low utility (experimental evaluation of this statement is given in section 4.6.10.3).

Furthermore, errors are unlikely to accumulate as a sum of errors. Instead, it is likely that an error can be (partially) corrected afterwards. Let iteration i with $1 \leq i \leq k$

[13]Spark actually provides multiple persistence strategies (cf., section 2.2.4.2). Some of them have the mentioned properties and some do not. It is assumed that an appropriate strategy is chosen in the following.

return the first erroneous pattern. Let furthermore $S_{i,\text{exact}}$ be the exact non-erroneous solution to iteration i. Then the algorithm can be divided into two stages. First, the stage of solution elements 1 to i (inclusive) (denoted by $S_{1\ldots i}$) and second, the stage of solution elements $S_{i+1\ldots k}$ to be added later. Thus the second stage operates on the problem instance $\mathcal{G}_2 := \mathcal{G} \setminus \mathcal{G}_{\triangleright S_{1\ldots i}}$ with the remaining cardinality budget $k_2 := k - i$. During this second stage, it is always possible to select a solution $S_{\text{exact},i+1\ldots k}$, that would have been selected if the error in iteration i would not have occurred. There are two reasons a part of the error is most probably corrected later on: First, there are some dataset graphs $\mathcal{G}_{\text{missed}} := \mathcal{G}_{\triangleright S_{i,\text{exact}}} \setminus \mathcal{G}_{\triangleright S_i}$ that are uncovered by S_i and covered by $S_{i,\text{exact}}$. Vice versa, there may also exist some graphs $\mathcal{G}_{\text{added}} := \mathcal{G}_{\triangleright S_i} \setminus \mathcal{G}_{\triangleright S_{i,\text{exact}}}$ that are covered by S_i and uncovered by $S_{i,\text{exact}}$. Since $|\mathcal{G}_{\text{missed}}| > |\mathcal{G}_{\text{added}}|$, the overlap of covered graphs between $S_{\text{exact},i+1\ldots k}$ and $S_{1\ldots i}$ is expected to be smaller than the overlap between $S_{\text{exact},i+1\ldots k}$ and $S_{1\ldots i-1} \cup S_{i,\text{exact}}$. Thus, a part of the lost utility is most likely re-added during the second stage. Second, there may exist a better solution $S_{i+1\ldots k}$, than $S_{\text{exact},i+1\ldots k}$ and such a solution would also compensate for some of the lost utility.

Last, it should be noted, that erroneous solutions may not have a monotonically decreasing utility with each iteration. It may even happen, that patterns with similar utility and a low overlap of covered graphs simply flip the positions in the iteration order.

4.6.9.2 Computational Complexity

4.6.9.2.1 Non-Distributed Streaming Algorithm In this paragraph, algorithm 12 will be analyzed w.r.t. its streaming model properties and its work complexity.

[**HRR98**] Henzinger, Raghavan, and Rajagopalan "Computing on data streams". 1998

The streaming analysis is based on the streaming model of Henzinger, Raghavan, and Rajagopalan [HRR98]. It allows multiple passes over a finite stream of data objects. Subject of interest is the number of passes and the memory complexity s.t., the stream length. For each solution element, algorithm 12 requires three passes over the dataset: The first pass is required for the sampling in line 5, the second one is required to calculate the utility for each candidate pattern $C \in \mathfrak{C}$ in line 9, and the third pass is necessary to filter the dataset in line 15. Thus, the algorithm terminates after a maximum of $3k$ passes. The memory complexity of the algorithm is dominated by the random sampling. For this, it is required to know the number of graphs in the dataset, which results in a logarithmic asymptotic worst case complexity. In practice this number is negligible, especially in comparison to the memory requirements of the FSM algorithm and the candidate patterns. Nevertheless the asymptotic worst case memory complexities for corresponding parts do only depend on the sample size s, which is a constant.

In the following, the asymptotic worst-case work complexity of algorithm 12 is discussed. The work of algorithm 12 can be divided into two parts: Lines 8 to 12, which corresponds to phase 2 of algorithm 7, and the rest of the algorithm, which includes phase 1 (implemented in algorithm 11).

In the worst case, the candidate set $\mathfrak{C}$ may contain each enumerated pattern of the pattern space exploration, which may be the complete subgraph pattern space

of the sample, i.e., $\mathfrak{C} \subseteq \mathfrak{S}_{1_{\sqsubseteq}} \subseteq \mathcal{G}_{\sqsubseteq}$ or the output of the FSM algorithm with a support threshold of 1 on $\mathfrak{S}_1$. This worst-case bound can be actually reached for some subgraph pattern coverage relations. For example, a uniform dataset, which is a dataset of isomorphic graphs, would have the same support value for each pattern. In this case, no pattern could be rejected by the hypothesis test. For the relative size thresholded subgraph pattern coverage relation $\lhd_t$ smaller patterns would be discarded. Nevertheless, in the worst case, this is only a tiny fraction of the patterns, given the combinatorial explosion of larger patterns in complete or dense graphs. However, in practice, the pattern space is usually much more diverse and the candidate set is usually much smaller than the complete subgraph pattern space of $\mathfrak{S}_1$, i.e., $\mathfrak{C} \ll \mathfrak{S}_{1_{\sqsubseteq}}$ (cf., the experimental evaluation section 4.6.10). To evaluate the candidate set, each of the patterns has to be compared with each dataset graph, involving the subgraph isomorphism test of the subgraph pattern coverage relation $\lhd$. Let $w(P \lhd G)$ be the work that is required to evaluate the relation $P \lhd G$) and $w_{\max}(\lhd)$ the maximal work required for any $P \in \mathfrak{S}_{1_{\sqsubseteq}}$ and $G \in \mathcal{G}$, then the overall worst-case work complexity for phase 2 is in $\mathcal{O}(\sum_{P \in \mathfrak{S}_{1_{\sqsubseteq}}, G \in \mathcal{G}} w(P \lhd G))$ or $\mathcal{O}(\left|\mathfrak{S}_{1_{\sqsubseteq}}\right| \, |\mathcal{G}| \, w_{\max}(\lhd))$.

The work of the rest of algorithm 12, except lines 5 and 15, is asymptotically identical to the running time of algorithm 6 (cf., section 4.5.7.1 for the corresponding analysis) in relation to the sample $\mathfrak{S}$ (instead of complete dataset $\mathcal{G}$). Besides the sampling in line 5 and the differences in algorithm 11, the main loop of algorithm 12 aligns with algorithm 6. It will be shown in the following, that algorithm 11 does only have a constant overhead w.r.t. to the corresponding lines 5 to 14 in algorithm 6. For this, the relevant differences between algorithm 6 and algorithm 9 will be discussed first and the parallelization of algorithm 9 given by algorithm 11 will be discussed in a second step.

From a complexity point of view, algorithm 9 and the corresponding parts in algorithm 6 differ in three main aspects: (a) The support-based pattern space pruning is bound by the rejection threshold of the candidate test and not by the utility value. (b) Algorithm 9 maintains a second transaction list $\mathfrak{S}_{2_{\sqsupseteq P}}$ for each pattern. (c) The candidate set $\mathfrak{C}$ replaces the single maximal pattern $P_{\max}$ and $\mathfrak{C}$ is filtered whenever a new reference pattern is found. The analysis of algorithm 6 in section 4.5.7.1 relates the asymptotic upper bound to the running time of the FSM algorithm with a support threshold of 1 donated by $\texttt{FSM}_{\mathcal{G}_i,1}$. While the rejection threshold (cf., item a) implies a less strict pruning power in comparison with the utility-based pruning, it is obviously respecting the bound $\texttt{FSM}_{\mathcal{G}_i,1}$. The union of the two transaction lists $\mathfrak{S}_{1_{\sqsupseteq P}}$ and $\mathfrak{S}_{2_{\sqsupseteq P}}$ (cf., item b) correspond to a single transaction list $\mathfrak{S}_{\sqsupseteq P}$. Since the running time of algorithm 12 is related to $\mathfrak{S}$, the transaction list $\mathfrak{S}_{\sqsupseteq P}$ is actually part of the related frequent subgraph mining instance with a minimum support of 1 and the analysis of algorithm 6 is still valid. Adding a pattern to the candidate set $\mathfrak{C}$ (cf., item c) is a constant time operation. However, the filtering of the candidates in line 15 requires a linear scan of the candidates if implemented naively. Using a bucket sorted candidate set instead of an unsorted set or list solves this issue. In this case, insertions are still a constant time operation and the filtering only requires the deletion of buckets with lower values. The amortized costs are linear

in the number of additions to $\mathfrak{C}$ and the size of $\mathfrak{S}_1$, which is an upper bound for the utility. Thus, the management of the candidate set only adds a constant overhead to the overall work complexity.

The parallelization of algorithm 9, i.e., algorithm 11 and the related subprocedure algorithm 10, does not change many aspects w.r.t. to work complexity. Besides some synchronization blocks, the main difference is to maintain a sorted priority queue for the work packages. Since the priority values are limited by maximum pattern size, it is again possible to resort to a bucket sorted data structure, resulting in constant time `enqueue` and `dequeue` operations. Thread synchronization can be considered a constant work operation for modern hardware and a limited number of processors.

In summary, the complexity of the algorithm is divided into two parts related to phase 1 and phase 2. The asymptotic worst-case work complexity is the same as for the exact algorithm algorithm 6. The complexity of phase 2 is heavily influenced by the actual candidate set cardinality. The work for the remaining lines 5 and 15 of algorithm 12 is linear in $|\mathcal{G}|$ and can be asymptotically subsumed by the work assigned to phase 2.

4.6.9.2.2 Distributed Algorithm Since Spark does not provide asymptotic worst-case running times for all RDD operations, algorithm 17 makes use of a variety of different RDD operations, and network effects are hard to estimate during analysis, no exact worst-case analysis of the work required by algorithm 17 will be given here. Nevertheless, some key aspects of performance are discussed below.

First, it should be noted, that phase 2 is a simple counting problem and that the key phase of interest w.r.t. to distribution performance is phase 1. During this phase algorithm 16 divides the gSpan recursion tree gSpanRecTree into independent work packages (cf., definition 4.6), i.e., non-overlapping units of work. These work packages are processed by algorithm 15 which is the partial variant of algorithm 11. Thus, the accumulated work of all instances of algorithm 15 is expected to be very close to the work of the pure shared memory variant. Minor deviations of the necessary work are the result of the following factors: (a) The overhead necessary to update the distributed atomics $\hat{p}_{1l,\text{global}}$ and `activeWorkers`. These operations are lightweight and limited in their frequency. The rejection threshold can be updated at most one time for each pattern and at most as often as the optimal utility value of a reference pattern (i.e., $\texttt{utility}_{R_{\max}}$). The number of active workers is only updated at most once per instance. In case of update conflicts, the update is repeated, but in each repeat, the number of conflicting updates is reduced by at least one pending concurrent update. (b) The re-computation of transaction lists for distributed work packages root patterns (transaction lists are initialized with the full sample in line 9 of algorithm 15). However, in the worst-case analysis of the non-distributed algorithm (cf., paragraph 4.6.9.2.1), these transaction lists did contain all graphs from the sample. Additionally, in practice, not many synchronization rounds are necessary (see experiments below). Thus, a large amount of patterns is usually processed before the termination of the partial miner, resulting in a negligible practical impact of this aspect. (c) The different exploration order, caused by a more parallelized exploration

of the search space and the distribution strategy of the work packages. The former aspect is also observable for the shared-memory parallelization. The direction of this running time effect is unclear, but a more random exploration order might actually help to find good reference patterns with good pruning bounds faster at the beginning. Thereby, it can even have a positive running time effect in cases where the sequential algorithm fully explores low utility subspaces of the search space at the beginning of the algorithm. (d) The delayed update of the rejection threshold. The delay causes some miners to run with less than optimal pruning power for a short period of time. However, a practically observable impact is unlikely given the usually quite low network latency compared to the overall running time of the algorithm.

A larger impact on the performance of the algorithm can be expected due to work package synchronization itself. This synchronization results in multiple rounds of distributed computation for each solution element of algorithm 16. While the candidate lists and samples are kept in the worker's memory and are not transferred over the network in between these rounds, synchronization needs some time in order to await the termination of all miners, collect the miners' work packages and redistribute them. As discussed during the presentation of the algorithm, too frequent synchronizations thereby might hurt the performance of the algorithm while too few synchronizations might lead to a low utilization of available resources. The parameters $\texttt{activeWorkers}_{\min}$ and $\texttt{iterations}_{\min}$ were introduced to algorithm 16 to avoid situations in which some miners, which run out of work very fast, cause very frequent synchronization. Since the recursion tree gSpanRecTree is limited in depth by the pattern size the cardinality of higher-level poset ranks should usually be high enough for a good parallelization. Nevertheless, the DFS-like exploration order may result in a situation where some miners only get very small work packages assigned. In theory, this might even result in a linear number of synchronization rounds w.r.t. to the number of processed patterns if more than $W - \texttt{activeWorkers}_{\min}$ workers get constant-sized work packages assigned in each round. However, given a random distribution strategy of work packages, this event is unlikely.

4.6.10 Experimental Evaluation

The goal of this chapter is the evaluation of the sampling-based approach w.r.t. performance and quality. A focus is put on the influence of the sample cardinality on the candidate set cardinality and the pruning power as well as the effectiveness of the approach w.r.t. to different algorithm parametrization using the non-distributed streaming algorithm 12. The distributed algorithm 17 is evaluated regarding the practical scaling behavior in dependence on distribution parameters and problem instances. It is furthermore demonstrated that the approach scales to hard and very large-scale settings. Finally, the representative mining approach is applied to the StruClus algorithm. It is demonstrate, that the objective of problem 4.1 (MAX-CSPC) can improve the clustering quality of this state of the art clustering algorithm.

Table 4.2: Descriptive statistics of the evaluation datasets.

		$\lvert V\rvert$				$\lvert E\rvert$					
Dataset $\mathcal{G}$	$\lvert\mathcal{G}\rvert$	Min.	Max.	Avg.	Med.	Min.	Max.	Avg.	Med.	$\lvert\mathcal{L}_V\rvert$	$\lvert\mathcal{L}_E\rvert$
PubChem Compounds	103 274 330	2	891	25.90	24	1	890	27.66	25	114	5
ChemDB	7 100 106	2	435	28.66	28	1	496	31.12	31	89	5
ChEMBL	1 678 393	2	876	29.79	27	1	894	32.18	30	34	5
CH*I*PMUNK Complete	18 292 556	3	52	28, 25	29	2	60	31.02	32	66	6
CH*I*PMUNK Heterocycle CoMol Subset	4 158 909	4	39	27.56	28	4	46	29.71	30	62	6
Protein Interaction	2 242 972	4	126	5.17	4	3	125	4.17	3	717	1

4.6.10.1 Computational Environment

The Hard- and software environment is identical to the environment for the non-distributed evaluation (cf., section 4.5.9.1). In addition, Spark is installed in version 2.4.3 using Scala 2.12 and Hadoop 2.7. It is configured to use the local node storage for persistence. Since the LiDO3 cluster does not have a Spark compatible cluster manager installed, Spark's standalone cluster manager was configured to launch from a SLURM job.

4.6.10.2 Datasets

In addition to the previously used datasets (cf., section 4.5.9.2) two additional datasets, CH*I*PMUNK Complete and PubChem, are used to demonstrate the scalability of the approach. Table 4.2 shows the descriptive statistics of the complete list of datasets used in the following evaluation.

The CH*I*PMUNK Complete dataset is a superset of the already known CH*I*PMUNK Heterocycle CoMol dataset. It is the drug-like 500 g mol^{-1} variant of the CH*I*PMUNK library including all subsets of the library as presented in [Hum+18]. Figure 4.15 shows the size and label distributions of the dataset.

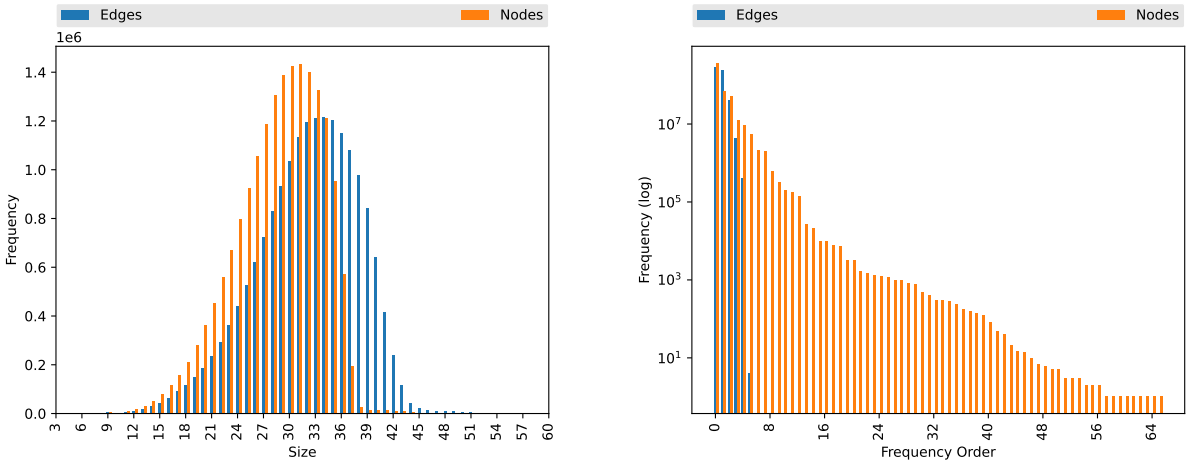
[Hum+18] Humbeck et al., "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". 2018

The PubChem compound library [Kim+21] is an open chemistry database of the National Institute of Health (NHI). It is the largest publicly available collection of real molecules and aggregates publicly available data about the molecules from many different sources. PubChem mostly contains small molecules. However, it also contains larger molecules such as nucleotides, carbohydrates, lipids, peptides, and chemically-modified macromolecules. For the experimental evaluation, a snapshot of the compound library was downloaded on the 10th of July 2020. Figure 4.16 shows the size and label distributions of the dataset.

[Kim+21] Kim et al., "PubChem in 2021: new data content and improved web interfaces". 2021

4.6.10.3 Influence of the Sample Size and Error Probability

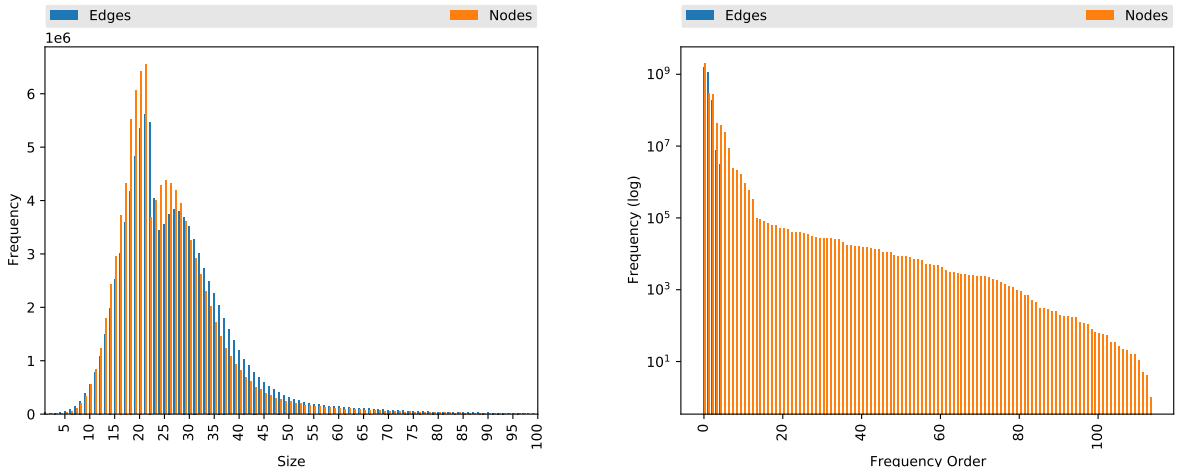
It was discussed during the evaluation of the exact algorithm, that the dataset size has a linear influence on the running time of the algorithm if the search space is identical. Besides the limits of the minimum sample size inflicted by the candidate test to generate valid results, one might be tempted to choose very small sample



(a) Size distribution of the CH*I*PMUNK Complete dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the CH*I*PMUNK Complete dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

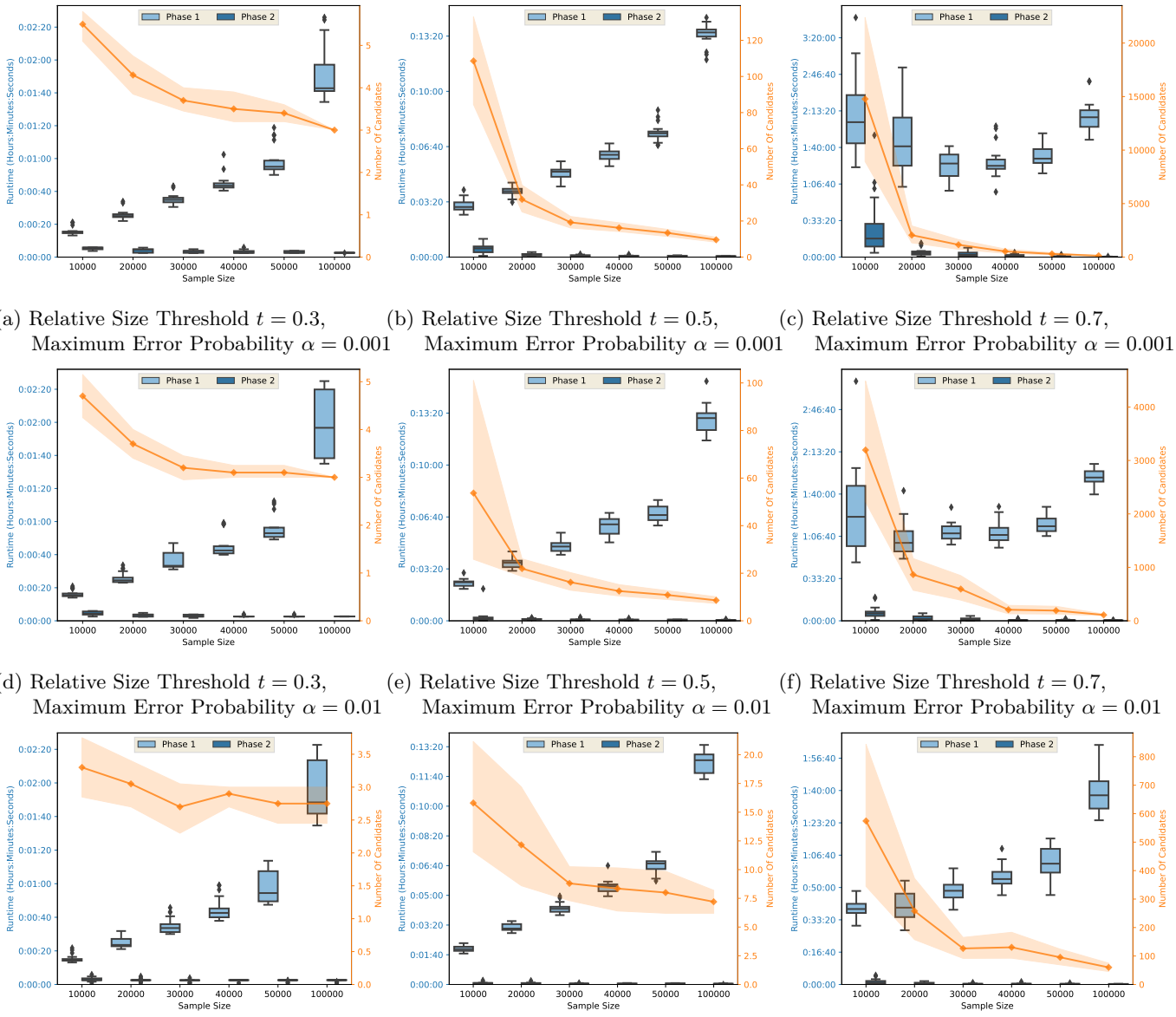
Figure 4.15: Frequency distributions of the CH*I*PMUNK Complete dataset



(a) Size distribution of the PubChem dataset. The plot is limited to a maximum size of 100. Larger sizes have frequencies below the displayed values.

(b) Label distribution of the PubChem dataset. Labels are sorted by their frequency. The most frequent labels appear on the left.

Figure 4.16: Frequency distributions of the PubChem dataset



(a) Relative Size Threshold $t = 0.3$, Maximum Error Probability $\alpha = 0.001$

(b) Relative Size Threshold $t = 0.5$, Maximum Error Probability $\alpha = 0.001$

(c) Relative Size Threshold $t = 0.7$, Maximum Error Probability $\alpha = 0.001$

(d) Relative Size Threshold $t = 0.3$, Maximum Error Probability $\alpha = 0.01$

(e) Relative Size Threshold $t = 0.5$, Maximum Error Probability $\alpha = 0.01$

(f) Relative Size Threshold $t = 0.7$, Maximum Error Probability $\alpha = 0.01$

(g) Relative Size Threshold $t = 0.3$, Maximum Error Probability $\alpha = 0.1$

(h) Relative Size Threshold $t = 0.5$, Maximum Error Probability $\alpha = 0.1$

(i) Relative Size Threshold $t = 0.7$, Maximum Error Probability $\alpha = 0.1$

Figure 4.17: Running Time w.r.t. different sample sizes s for algorithm 12 on the ChemDB dataset. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The mean cardinality of the candidate set $\mathfrak{C}$ is given in orange with standard deviation given in lighter color. The dataset is filtered to graphs with an edge count of 30 or less.

sizes. However, the sample size also influences the power of the candidate test. The larger the absolute utilities are the closer is the rejection threshold to the utility of the reference pattern in relative terms. Thus, larger sample sizes most probably increase the pruning power of the search space and thereby have a positive effect on the running time. It should be noted though, that very large samples may not tighten the search space pruning significantly, since the rejection threshold is bound by the utility of the reference pattern. Besides the search space pruning implications, a higher rejection threshold leads to smaller candidate sets, which in turn reduces the number of comparisons in phase 2. As shown w.r.t. the pruning power there is a natural limit to this effect since the candidate set will always contain at least a single pattern. For these reasons, it is unclear which sample size leads to the best overall running time from a theoretical point of view.

The maximum error probability α has a similar effect on the running time. Lower error probabilities result in a lower test power and consequentially in a less tight search space pruning and larger candidate set cardinalities.

The experiments in this section will therefore focus on the running time effect of different sample sizes and error probabilities. They are limited to a single iteration of the non-distributed streaming algorithm, i.e., algorithm 12, and the ChemDB dataset. Figure 4.17 shows the wall clock running times of phase 1 and phase 2 (as divided in the analysis of paragraph 4.6.9.2.1) and the cardinality of the candidate set w.r.t. the sample sizes $s \in \{10\,000, 20\,000, 30\,000, 40\,000, 50\,000, 100\,000\}$, the maximum error probabilities $\alpha \in \{0.001, 0.01, 0.1\}$, and the relative size thresholds $t \in \{0.3, 0.5, 0.7\}$.

In all experiments, the running times of phase 2 are significantly lower than the ones of phase 1. While a clear trend towards lower cardinalities of the candidate sets in case of higher sample sizes is observable throughout all settings, the effect of the low cardinalities and the resulting lower running times of phase 2 is almost negligible w.r.t. the overall running times. Only in some very hard settings, i.e., a low error probability α and a high relative size coverage threshold t, the running time of phase 2 has a limited effect on the overall running time.

Except for these hard cases, a general increase in running time w.r.t. larger sample sizes can be seen for phase 1. The exception w.r.t. the hard cases comes as no surprise since the test power decreases for lower error probabilities as discussed above. Furthermore, higher values of the relative size coverage threshold lead to lower absolute utilities and thereby lower test powers as well. Finally, the lesser pruning power might cause the exponential worst-case complexity of the subgraph isomorphism and pattern canonization tests to become more relevant in comparison to the linear sample size effect.

To conclude, with low absolute utilities and low error probabilities it might be beneficial to choose a sample size above 10 000 performance-wise. Additionally, the running time of phase 2 linearly depends on the dataset size. Thus, given very large datasets, which are magnitudes larger than ChemDB, phase 2 might become a more crucial performance factor. Otherwise, the smaller sample sizes seem to perform better. It is expected though, that the best sample size might vary for different datasets and later iterations in case of larger solution set cardinalities. These aspects will be evaluated in the following section 4.6.10.4.

Table 4.3: Real errors and approximation ratios of worst results w.r.t. the experiments presented in fig. 4.17 grouped by the relative size coverage threshold t and the probabilistic error bound α.

α	t	Correct Results (Error Rate)	Worst Approximation Ratio
0.001	0.3	120/ 120 (0.00%)	1
0.001	0.5	120/ 120 (0.00%)	1
0.001	0.7	120/ 120 (0.00%)	1
0.01	0.3	120/ 120 (0.00%)	1
0.01	0.5	120/ 120 (0.00%)	1
0.01	0.7	120/ 120 (0.00%)	1
0.1	0.3	115/ 120 (4.17%)	0.995
0.1	0.5	120/ 120 (0.00%)	1
0.1	0.7	113/ 120 (5.83%)	0.976

4.6.10.3.1 Real Errors and Approximation Ratios Given the theoretically bound error probability as shown in lemma 4.6 it is an open question how tight this error bound is in practice. Additionally, the algorithm might give close to optimal results if an error occurs. In fact, it is much more likely that the algorithm outputs a close to the optimal solution than a bad solution as a result of the binomial test nature.

Table 4.3 shows the errors for the experiments conducted above (cf., fig. 4.17). The results are grouped by the error probabilities α and the relative size coverage thresholds t, resulting in 120 repeats in each category and 360 repeats for distinct values of α. For $\alpha \in \{0.001, 0.01\}$ no errors are present in the given repeats. This is expected for $\alpha = 0.001$, since the number of experiments is relatively low compared to the error probabilities. However, at least a single error would be expected for the other settings under a tight error bound assumption. If the bound would be tight, i.e., the expected real error would be equal to the α, at least one error would be expected for 360 repeats (ignoring t) with a probability of ≈ 0.30 for $\alpha = 0.001$, ≈ 0.97 for $\alpha = 0.01$, and ≈ 1.00 for $\alpha = 0.1$. Overall, 12 errors are observed for $\alpha = 0.1$, which is about one-third of the maximal expected error. No trend is observable w.r.t. to t and the tightness of the error bound.

Concerning the approximation ratios of the erroneous results, the worst result was observed for the setting $t = 0.7$ and $\alpha = 0.1$ with an approximation ratio of 0.976 and an absolute utility of 8 186 instead of the optimal value 8 391. Thus, even in the case of errors, high-quality results are observed.

4.6.10.4 Performance and Key Statistics over Multiple Iterations of the Streaming Algorithm

In alignment with the experiments conducted for the exact algorithm in section 4.5.9.4, this section will evaluate the performance of the streaming algorithm 12 over different iterations. Figures 4.18 to 4.21 show the results for the ChemDB, ChEMBL, CH*I*PMUNK Heterocycle CoMol, and Protein Interaction datasets filtered to 30 edges. The maximum error probability per iteration is set to 0.01 for all experiments. In addition to the known statistics "running time", "utility", "uncovered graphs", and "pattern edge count" from section 4.5.9.4, the running time will be split into phase 1 and phase 2. Furthermore, the cardinality of the candidate set $\mathfrak{C}$ is added, such that the running time of phase 2 can be judged in further detail. Since the cardinality of $\mathfrak{C}$ varies over different repeats with i.i.d. drawn random samples, the standard deviation is given as transparent colored area around the line plot. The same is true for other line plots if solutions diverge because of errors or ambiguities w.r.t. to the optimal pattern (i.e., multiple patterns with equal utility; cf., corollary 4.7). Experiments are performed with sample sizes $s \in \{10\,000, 30\,000\}$ to judge if sample sizes above 10 000 (cf., previous evaluation of the sample size in section 4.6.10.3) are beneficial in later iterations.

The overall running times of algorithm 12 are always faster than the running time of algorithm 6. The speedups are in the range of one or two orders of magnitude given $s = 10\,000$ with very few exceptions. For the exact algorithms, the decreasing dataset size led to lower running times in latter iterations in some cases. In some other cases, this effect was overshadowed by other running time factors, such as the absolute utility and the aforementioned effect on the pruning capabilities. In the case of algorithm 12, this effect is no longer present during phase 1, since each iteration will have equally sized samples. Furthermore, the experiments presented in section 4.5.9.4 showed, that the utilities are usually dropping faster than the number of uncovered graphs in the datasets in the first iterations. Thus, the absolute utilities are expected to drop for constant-sized random samples in later iterations. For this reason, a rise in the running times can be seen for phase 1 in some cases for later iterations of the algorithm. However, this rise is usually moderate. In some cases with $t = 0.7$, the running times and candidate set cardinalities even drop for later iterations. Of course, there exists some performance-wise crossing point, where the number of uncovered graphs gets close to the sample size. In this case, it would be beneficial to switch over to the exact algorithm in later iterations. An adaption of the exact algorithm to the distributed setting (evaluation see below) is straightforward if the whole dataset fits in each worker's memory. The candidate set cardinality has a high variance w.r.t. different iteration in some cases. For example, iteration 7 of the CH*I*PMUNK Heterocycle CoMol dataset in the setting $t = 0.5$ shows a very large sample in contrast to the other iterations. Thus, there exist inherently hard instances, where many patterns with close to optimal utility exist. In some other examples a high variance w.r.t. the repeats are observed, which indicates a high number of patterns with a utility on the border of the rejection threshold.

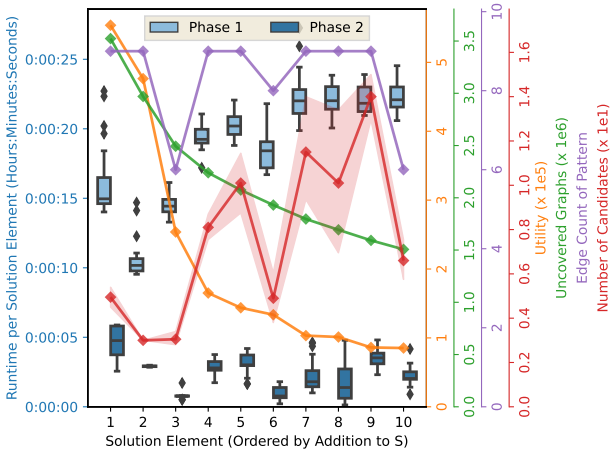
A sample size of $s = 30\,000$ usually leads to an increase in the overall running time. While the running times of phase 2 are often reduced significantly, the running time increase in phase 1 usually outweighs these benefits. The Protein Interaction dataset is an exception to this observation. It shows the unique characteristic, that the running time of phase 2 is usually higher than the one of phase 1 for a sample size of 10 000. This behavior could be explained by the low graph sizes. As a consequence, the pattern space itself has only a few ranks and is most likely smaller than that of the other datasets.

4.6.10.4.1 Error Accumulation over Multiple Iterations As discussed in the analysis (cf., section 4.6.9.1) an error accumulation most probably is not a problem in real case scenarios where close to optimal solutions are an option. During all the experiments, only a single setting (ChEMBL dataset, $s = 10000$, $t = 0.7$) deviates from the non-probabilistic results. Deviations are present in 5 of 20 results (repeats) at the end of the 10th iteration with 2 deviations being the results of an error attributed to the sampling approach and the rest are ambiguities in the selection of the optimal pattern. Such ambiguities express themselves in non-isomorphic patterns in comparison with the exact result in the iteration before a utility deviation was observed. The approximation ratio of the worst complete result ($k = 10$) is 0.98, while the worst approximation ratio in a single iteration was 0.89. While the number of erroneous results is too low to deduce statistically sound generalizations, the observed approximation ratios are an example of the discussed non-amplifying behavior of errors.

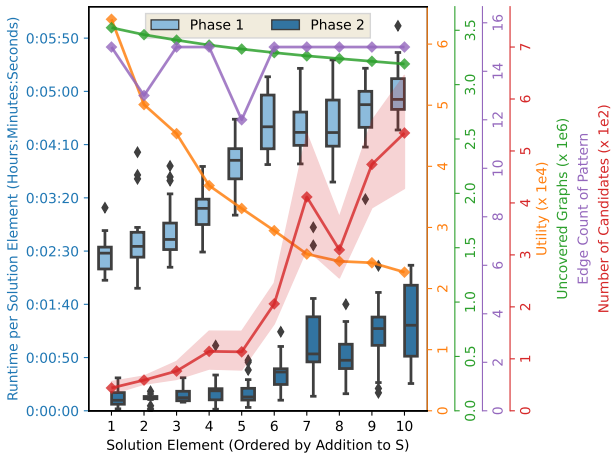
4.6.10.5 Distributed Scaling

Since the distribution does not introduce any new errors in relation to pure shared memory parallelization (cf., algorithm 12), the major question w.r.t. to the distributed algorithm (cf., algorithm 17) is its scalability with the numbers of miners and its overhead w.r.t. to the pure shared memory parallelized algorithm 12. To lower the total amount of resources needed on the LiDO3 cluster, distributed experiments are only repeated 10 times per parameter combination and are limited to a single solution element, i.e., $k = 1$.

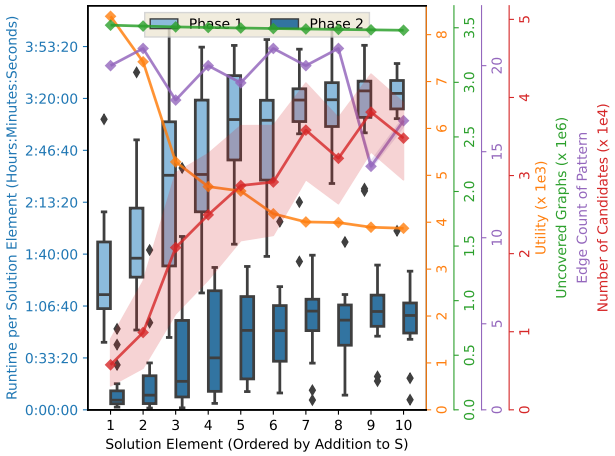
4.6.10.5.1 Distributed Parametrization This paragraph looks at the influence of scaling parameters on the scaling behavior algorithm 17. Figure 4.22 shows the scaling behavior w.r.t. $\texttt{activeWorkers}_{\min} \in \{75\%, 87.5\%, 100\%\}$. In other words, a fraction of $\frac{1}{4}$, $\frac{1}{8}$, or 0 workers are allowed to idle in phase 1 without causing synchronization of work packages. The ChemDB dataset is utilized and parameters are set to $t = 0.7$, $\alpha = 0.01$, and $s = 10000$. Thus, a problem instance was used, that needed about one hour wall-clock time in the non-distributed setting. To provide comparability with the non-distributed experiments, the dataset was again filtered to a maximum edge count of 30. Since, the other parameter that influences the scaling behavior $\texttt{iterations}_{\min}$ did not had a significant effect on the running time during evaluation, a detailed analysis will be omitted here and the parameter is fixed to $\texttt{iterations}_{\min} = 1000$ in all distributed experiments.



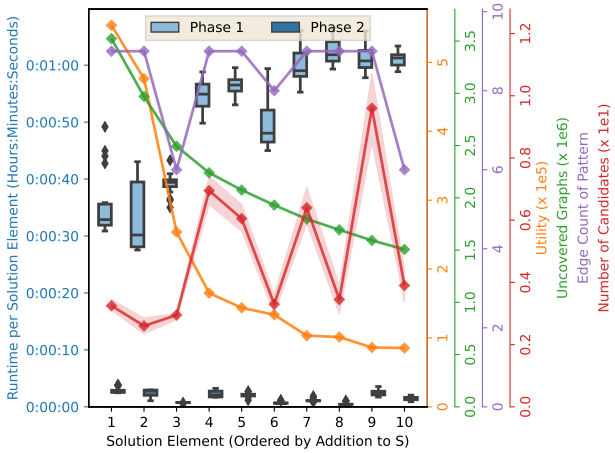
(a) Relative Size Threshold $t = 0.3$, Sample Size $s = 10\,000$



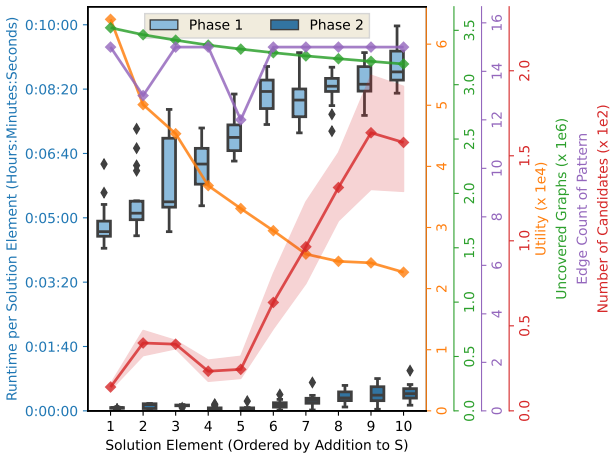
(b) Relative Size Threshold $t = 0.5$, Sample Size $s = 10\,000$



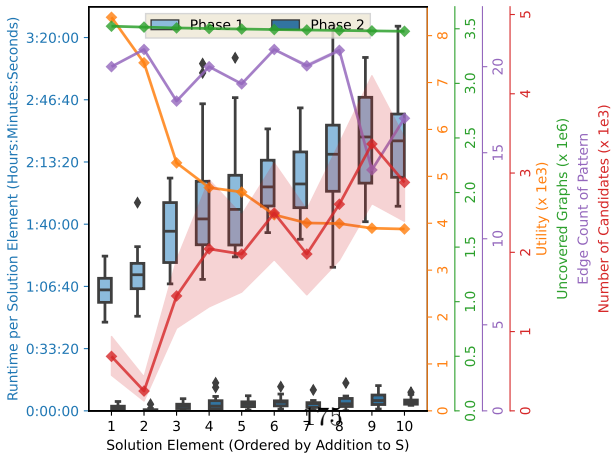
(c) Relative Size Threshold $t = 0.7$, Sample Size $s = 10\,000$



(d) Relative Size Threshold $t = 0.3$, Sample Size $s = 30\,000$

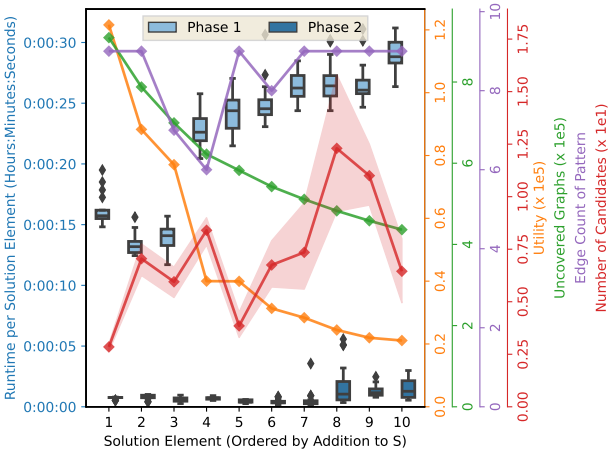


(e) Relative Size Threshold $t = 0.5$, Sample Size $s = 30\,000$

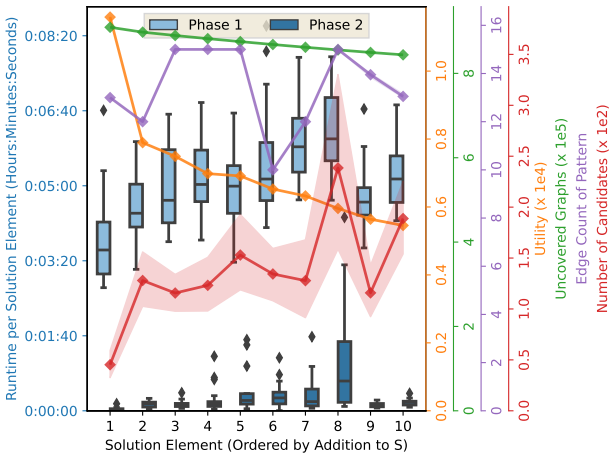


(f) Relative Size Threshold $t = 0.7$, Sample Size $s = 30\,000$

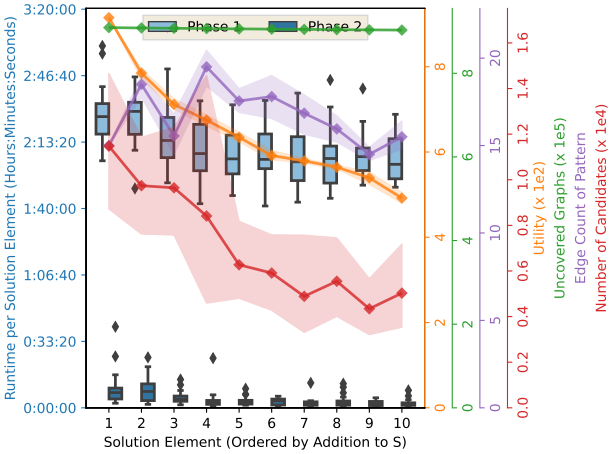
Figure 4.18: Running Time, utility, uncovered graphs, edge count of $P_{\max}$, and mean cardinality of the candidate set $\mathfrak{C}$ for each iteration (i.e., addition to S) of algorithm 12 on the ChemDB dataset. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The maximum error probability is set to $\alpha = 0.01$. The dataset is filtered to graphs with an edge count of 30 or less.



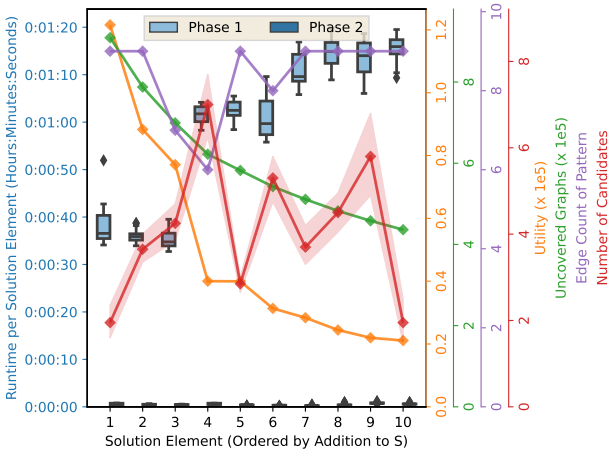
(a) Relative Size Threshold $t = 0.3$, Sample Size $s = 10\,000$



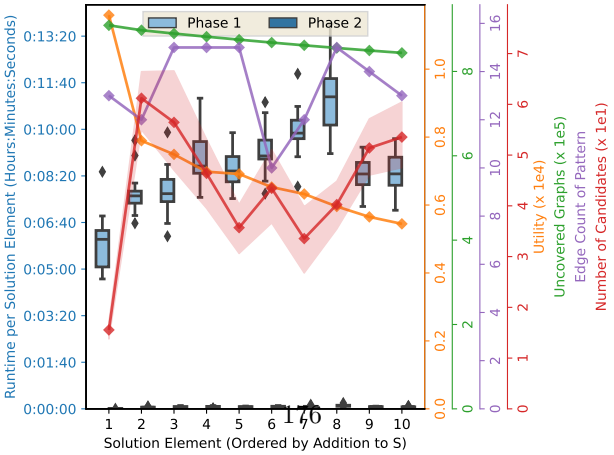
(b) Relative Size Threshold $t = 0.5$, Sample Size $s = 10\,000$



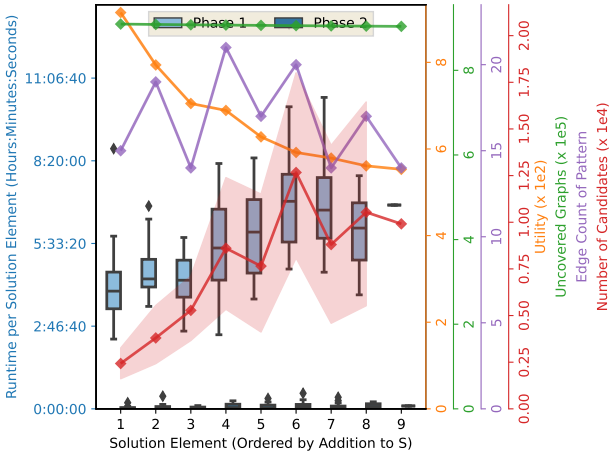
(c) Relative Size Threshold $t = 0.7$, Sample Size $s = 10\,000$



(d) Relative Size Threshold $t = 0.3$, Sample Size $s = 30\,000$

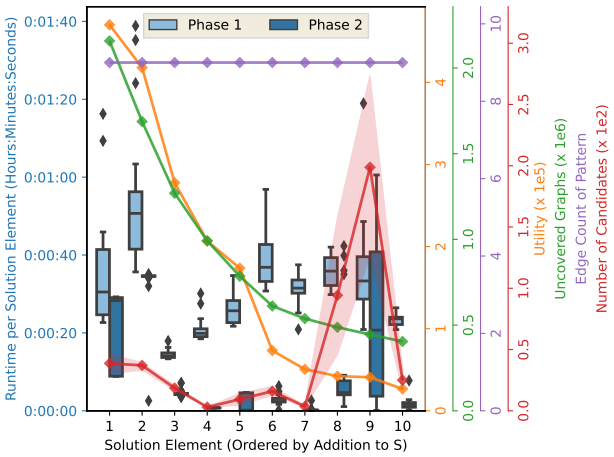


(e) Relative Size Threshold $t = 0.5$, Sample Size $s = 30\,000$

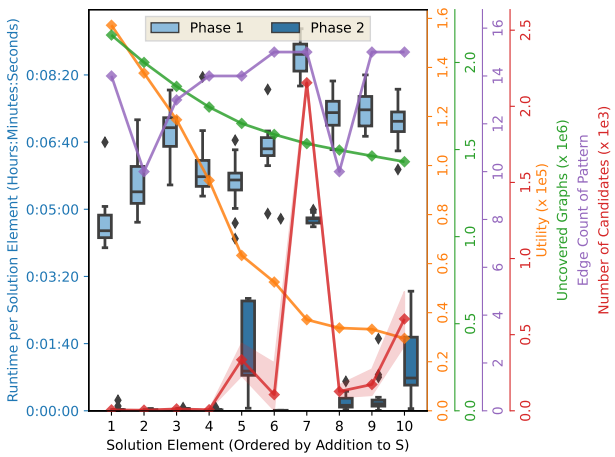


(f) Relative Size Threshold $t = 0.7$, Sample Size $s = 30\,000$

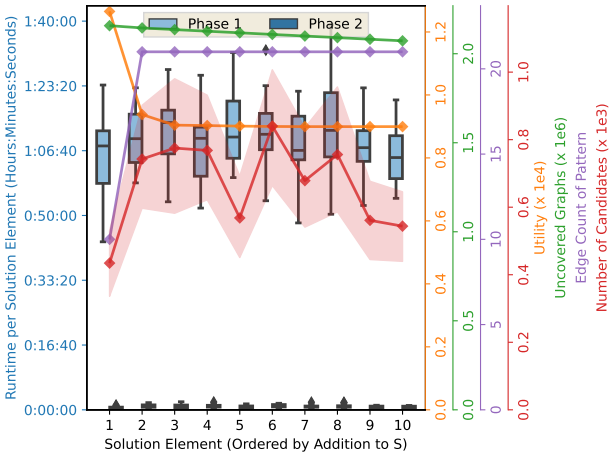
Figure 4.19: Running Time, utility, uncovered graphs, edge count of $P_{\max}$, and mean cardinality of the candidate set $\mathfrak{C}$ for each iteration (i.e., addition to S) of algorithm 12 on the ChEMBL dataset. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The maximum error probability is set to $\alpha = 0.01$. The dataset is filtered to graphs with an edge count of 30 or less.



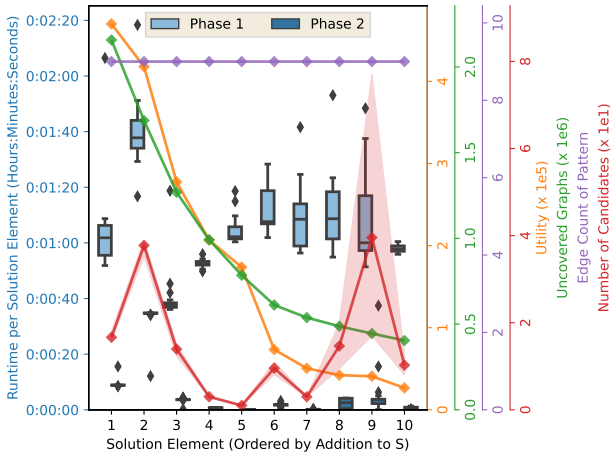
(a) Relative Size Threshold $t = 0.3$, Sample Size $s = 10\,000$



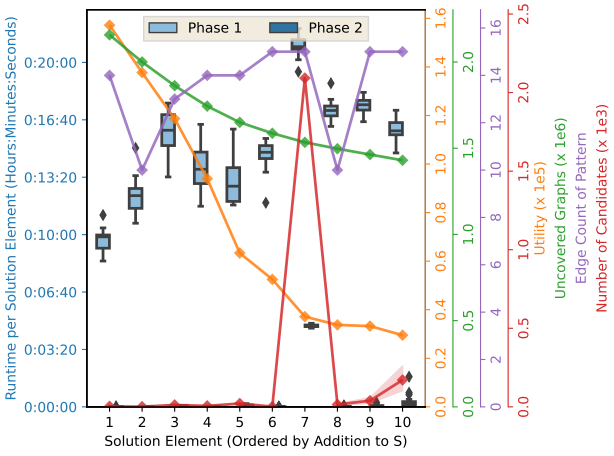
(b) Relative Size Threshold $t = 0.5$, Sample Size $s = 10\,000$



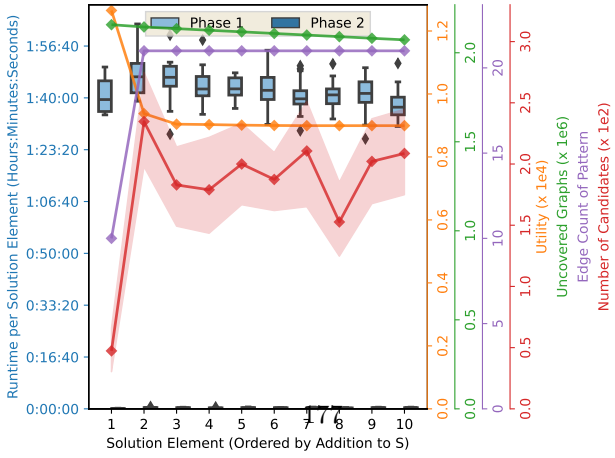
(c) Relative Size Threshold $t = 0.7$, Sample Size $s = 10\,000$



(d) Relative Size Threshold $t = 0.3$, Sample Size $s = 30\,000$

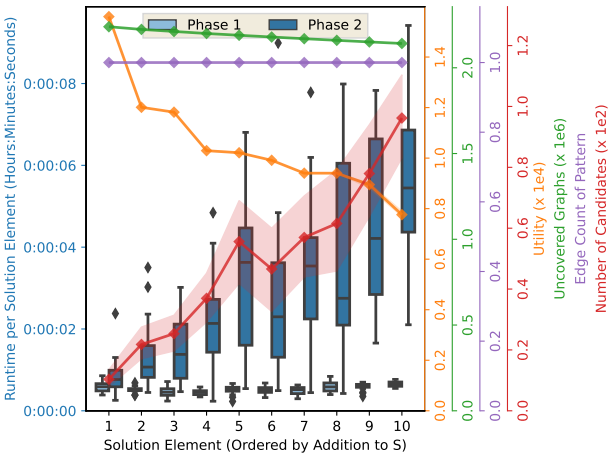


(e) Relative Size Threshold $t = 0.5$, Sample Size $s = 30\,000$

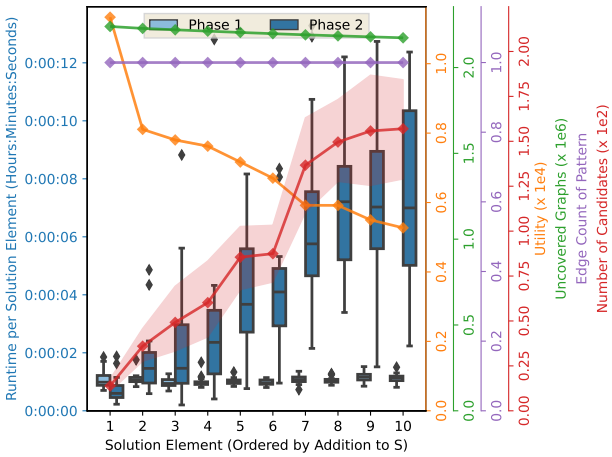


(f) Relative Size Threshold $t = 0.7$, Sample Size $s = 30\,000$

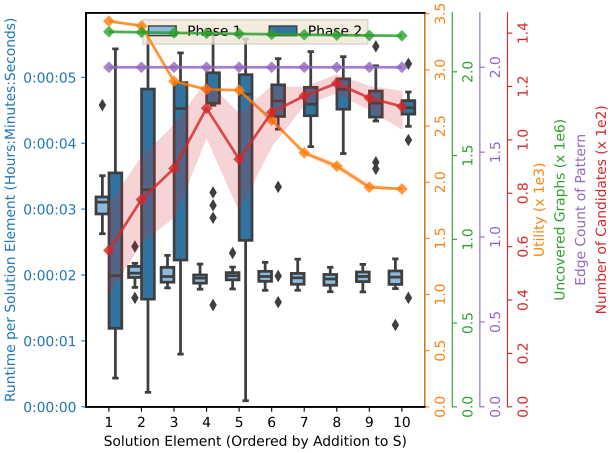
Figure 4.20: Running Time, utility, uncovered graphs, edge count of $P_{\max}$, and mean cardinality of the candidate set $\mathfrak{C}$ for each iteration (i.e., addition to S) of algorithm 12 on the CH*I*PMUNK Heterocycle CoMol dataset. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The maximum error probability is set to $\alpha = 0.01$. The dataset is filtered to graphs with an edge count of 30 or less.



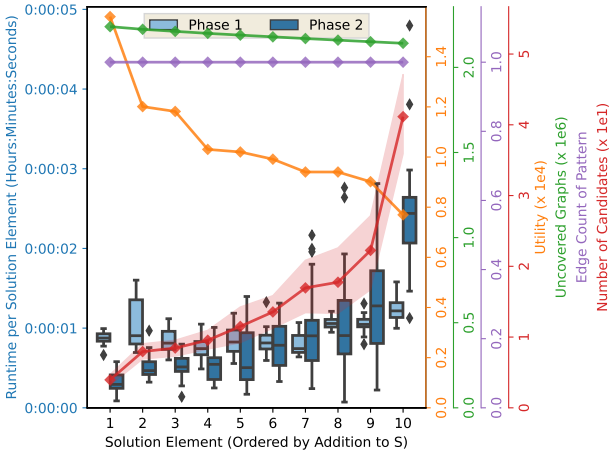
(a) Relative Size Threshold $t = 0.3$, Sample Size $s = 10\,000$



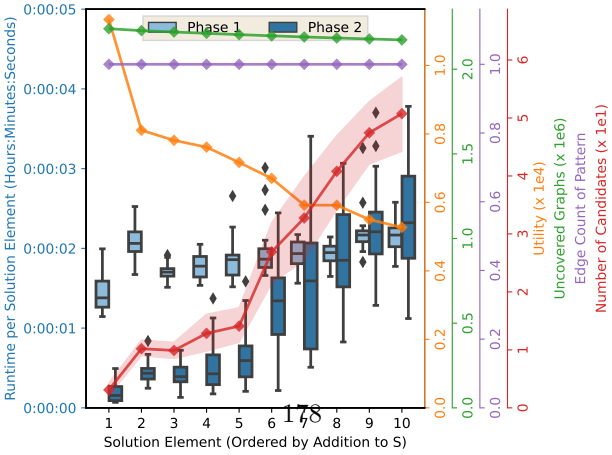
(b) Relative Size Threshold $t = 0.5$, Sample Size $s = 10\,000$



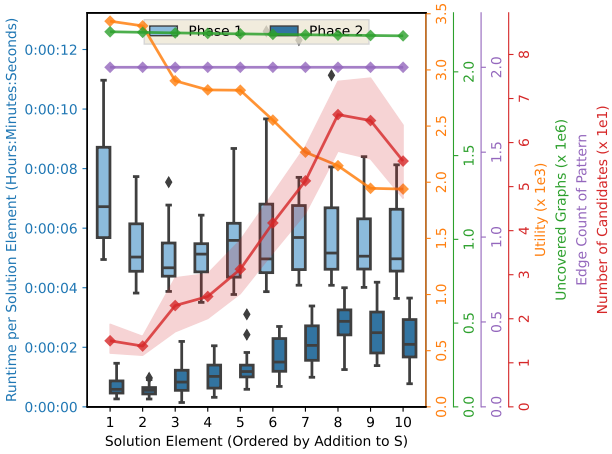
(c) Relative Size Threshold $t = 0.7$, Sample Size $s = 10\,000$



(d) Relative Size Threshold $t = 0.3$, Sample Size $s = 30\,000$



(e) Relative Size Threshold $t = 0.5$, Sample Size $s = 30\,000$



(f) Relative Size Threshold $t = 0.7$, Sample Size $s = 30\,000$

Figure 4.21: Running Time, utility, uncovered graphs, edge count of $P_{\max}$, and mean cardinality of the candidate set $\mathfrak{C}$ for each iteration (i.e., addition to S) of algorithm 12 on the Protein Interaction dataset. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The maximum error probability is set to $\alpha = 0.01$. The dataset is filtered to graphs with an edge count of 30 or less.

The experimental results show a good scaling behavior with a scaling factor of ≈ 12.0 for 16 workers and ≈ 7.3 for 8 workers on average given $\texttt{activeWorkers}_{\min} = 75\%$. The total number of distributed rounds in these settings is close to 20 on average for 16 workers and lower for fewer workers.

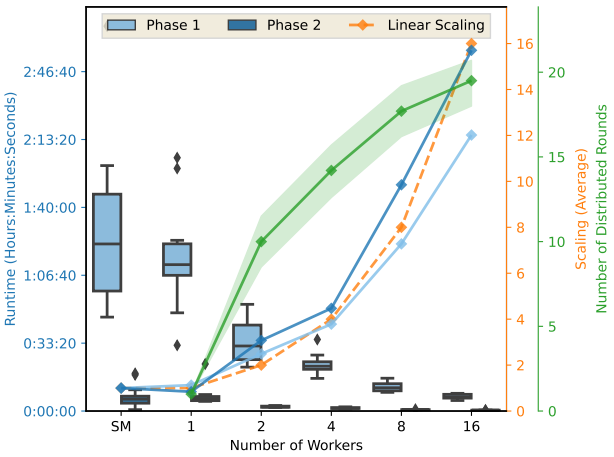
The parameter $\texttt{activeWorkers}_{\min}$ has a huge impact on the scalability for larger amounts of workers. In the settings $\texttt{activeWorkers}_{\min} \in \{87.5\%, 100\%\}$, the running time using 16 workers is even lower than for 8 workers. This is interesting to see as the number of iterations does not increase by a large amount, especially in the setting $\texttt{activeWorkers}_{\min} = 87.5\%$. Thus, other factors such as an increased amount of work packages most probably played a role in the scaling behavior. Furthermore, the settings with two workers show a more than linear scaling on average. The differences are not huge enough, such that a random effect could completely be ruled out. However, other effects such as a larger amount of randomness regarding the search space exploration order (cf., analysis in paragraph 4.6.9.2.2) most probably also played a role.

During phase 2 the scaling behavior is without flaws. Often a more than linear scaling can be observed which might have to do with a different shared-memory parallelization on the worker level. The shared-memory streaming algorithm uses Java thread pools for parallelization and the distributed variant solely relies on Spark. Spark uses batched parallelization strategies that might better utilize the caches of the compute nodes and may need less overhead for task creation and management.

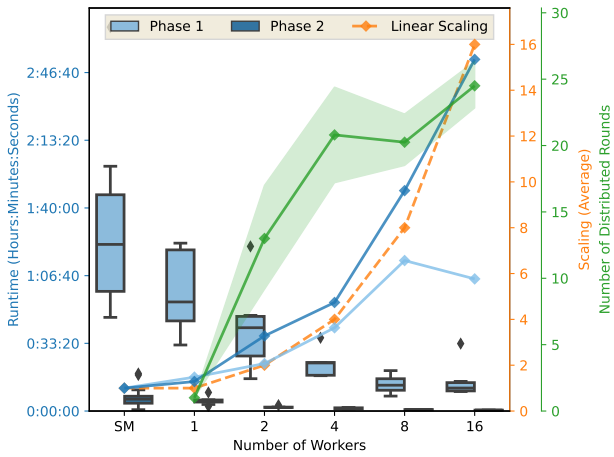
To measure the overall speedup w.r.t. to parallelization, algorithm 12 was also executed on a single-core to compute the shared-memory parallelization speedup. The experiment was applied to all datasets filtered to a maximum of 30 edges with $k = 1$ and $s = 10000$. All speedups were very close to the linear speedup (always above 90%). Given the above speedup of 12 for 16 workers, the overall speedup due to parallelization and distribution on 16 workers with 10 cores was above 100.

4.6.10.5.2 Distributed Scaling for Other Datasets and Relative Size Coverage Thresholds The scaling evaluation in paragraph 4.6.10.5.1 focused on a single dataset and $t = 0.7$. The following experiments focus on different datasets and relative size coverage thresholds with fixed scaling parameters to $\texttt{activeWorkers}_{\min} = 75\%$ and $\texttt{iterations}_{\min} = 1000$. The maximum error probability was again set to $\alpha = 0.01$. Table 4.4 shows the total running times and speedups w.r.t. to the shared-memory parallelized exact algorithm 6, the sampling and streaming algorithm 12, and the distributed algorithm 17. To get the speedups w.r.t. the pure sequential implementation of algorithm 6, it is again (cf., paragraph 4.6.10.5.1) possible to multiply the speedups with the interval $[9, 10]$.

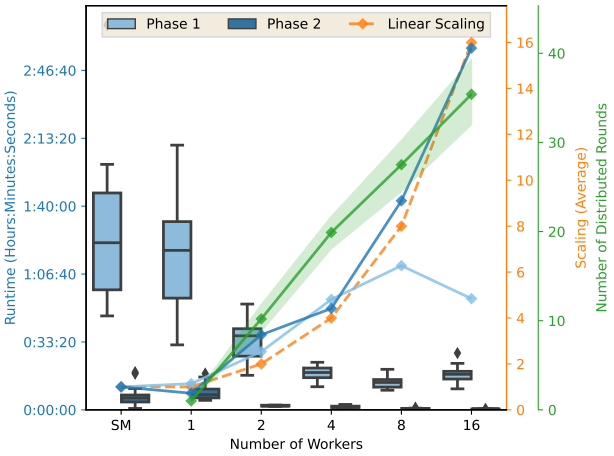
Overall the evaluation shows that the scaling behavior of the distributed algorithm 17 strongly depends on the total amount of work. In the extreme case, the running times of the Protein Interaction datasets were already lower than 3 seconds using algorithm 12. In such cases, the running times of the distributed algorithm even regressed. This comes as no surprise since there exists a constant overhead to initialize and coordinate spark workers. Thus, a key question w.r.t. to the effectiveness of distribution is whether the



(a) $\texttt{activeWorkers}_{\min} = 75\%$



(b) $\texttt{activeWorkers}_{\min} = 87.5\%$



(c) $\texttt{activeWorkers}_{\min} = 100\%$

Figure 4.22: Running Time scaling of a single iteration of algorithm 17 w.r.t. the number of miners in comparison with the pure shared memory parallelization (SM, cf., algorithm 12). Experiments are conducted on the ChemDB dataset with a relative size threshold $t = 0.7$, a maximum error probability $\alpha = 0.01$, and a sample size $s = 10\,000$. Running Time is differentiated w.r.t. phase 1 and phase 2 of algorithm 7. The dataset is filtered to graphs with an edge count of 30 or less.

Table 4.4: Running Times and speedups w.r.t. the shared-memory parallelized exact algorithm 5, shared-memory streaming algorithm 12, and the distributed algorithm 17. Running Times are evaluated for a single solution element ($k = 1$), different datasets and different settings of the relative size coverage threshold t. The sample size for algorithms 12 and 17 was set to $s = 10\,000$. Algorithm 17 was evaluated with $\texttt{activeWorkers}_{\min} = 75\%$ and $W = 16$. Datasets are filtered to a maximum edge count of 30. Running Times are given in hours:minutes:seconds.

		Running Time			Speedup		
Dataset	t	Algorithm 6	Algorithm 12	Algorithm 17	$6 \to 12$	$12 \to 17$	$6 \to 17$
ChEMBL	$0,30$	00:08:39	00:00:16	00:00:37	32.16	0.43	13.77
ChEMBL	$0,50$	01:05:33	00:03:47	00:00:59	17.29	3.8	65.75
ChEMBL	$0,70$	13:31:42	02:27:41	00:08:36	5.5	17.17	94.38
ChemDB	$0,30$	00:45:11	00:00:16	00:00:27	166.75	0.58	96.95
ChemDB	$0,50$	03:57:54	00:02:26	00:00:59	97.49	2.47	240.39
ChemDB	$0,70$	18:40:42	01:28:11	00:07:10	12.71	12.28	156.11
CHIPMUNK Heterocycle CoMol	$0,30$	00:28:23	00:00:35	00:00:44	47.97	0.81	38.65
CHIPMUNK Heterocycle CoMol	$0,50$	04:20:32	00:04:38	00:01:17	56.19	3.57	200.8
CHIPMUNK Heterocycle CoMol	$0,70$	1 days 10:47:23	01:05:40	00:09:46	31.78	6.72	213.55
Protein Interaction	$0,30$	00:00:21	00:00:00	00:00:35	37.4	0.02	0.61
Protein Interaction	$0,50$	00:00:36	00:00:01	00:00:39	33.39	0.03	0.93
Protein Interaction	$0,70$	00:01:43	00:00:03	00:01:27	33.07	0.04	1.19

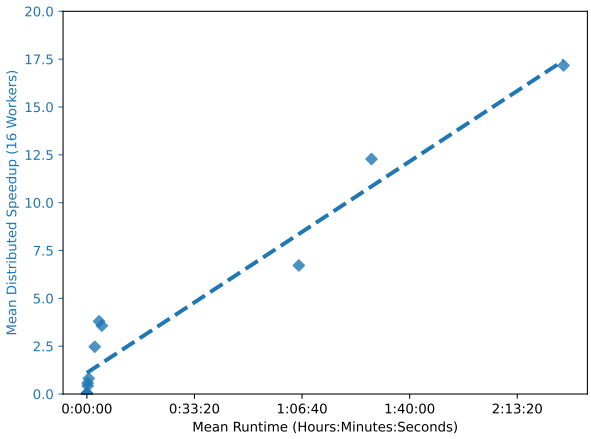


Figure 4.23: Distributed speedup of algorithm 17 w.r.t. the running time of algorithm 12. Dashed line shows the linear regression. Pearson Correlation Coefficient is $r = 0.97$.

amount of work is large enough to justify the distribution-related overheads. Instances, which require a few minutes to compute using algorithm 12, were accelerated, but show relatively low speedups ranging from about 2.5 to 4. However, for instances that need several hours using algorithm 12 the scaling behavior is much better with speedups ranging from approximately 6.7 for the CH*I*PMUNK Heterocycle CoMol dataset to 17.2 on the ChEMBL dataset. Overall, there is a strong correlation (Pearson correlation coefficient is $r = 0.97$) between the running time of algorithm 12 and the distributed speedup on the given instances. Figure 4.23 show a regression plot of these two data points. Of course, a linear increase in speedup cannot be expected beyond the number of workers. Nevertheless, it can be concluded, that the distribution scales well on hard instances with a large amount of work. It should be remembered, that the problem instances during the previous evaluation are chosen, such that they are still solvable in a meaningful amount of time using the exact algorithm 6 in order to compare the performance of the different algorithms against each other with a sufficient amount of repeats and parameter combinations. Thus, much harder instances are common in the practical application of the algorithm, when dataset filtering is not an option.

4.6.10.6 Hard and Big Instances

The goal of this section is to evaluate the limits of the distributed algorithm w.r.t. to hard and big instances. The algorithm was parameterized the same way as in the previous paragraph 4.6.10.5.2. A selection of hard instances is presented in table 4.5. The ChemDB dataset was filtered to a maximal edge count of 40, 50, and 60. The relative size threshold was set to $t \in \{0.5, 0.7\}$. The threshold t had the largest

Table 4.5: Running Times w.r.t. the distributed algorithm 17 for a selection of hard instances. Running Times are evaluated for a single solution element ($k = 1$). The sample size for algorithms 12 and 17 was set to $s = 10\,000$. Algorithm 17 was evaluated with $\texttt{activeWorkers}_{\min} = 75\%$ and $W = 16$. Running Times are given in hours:minutes:seconds.

Dataset	t	Running Time Phase 1	Running Time Phase 2
ChemDB ($\lvert E\rvert \leq 40$)	$0,50$	00:03:10 ($\sigma =$ 00:00:22)	00:00:45 ($\sigma =$ 00:00:02)
ChemDB ($\lvert E\rvert \leq 50$)	$0,50$	00:17:29 ($\sigma =$ 00:05:01)	00:07:05 ($\sigma =$ 00:05:54)
ChemDB ($\lvert E\rvert \leq 60$)	$0,50$	00:33:44 ($\sigma =$ 00:09:38)	00:12:59 ($\sigma =$ 00:11:11)
ChemDB ($\lvert E\rvert \leq 40$)	$0,70$	01:07:03 ($\sigma =$ 00:13:25)	00:01:04 ($\sigma =$ 00:00:07)
ChemDB ($\lvert E\rvert \leq 50$)	$0,70$	11:32:16 ($\sigma =$ 01:50:39)	00:06:17 ($\sigma =$ 00:05:29)
ChemDB ($\lvert E\rvert \leq 60$)	$0,70$	17:36:45 ($\sigma =$ 04:59:32)	00:07:09 ($\sigma =$ 00:07:00)
CH*I*PMUNK Heterocycle CoMol ($\lvert E\rvert \leq \infty$)	$0,50$	00:02:15 ($\sigma =$ 00:00:13)	00:00:29 ($\sigma =$ 00:00:00)
CH*I*PMUNK Heterocycle CoMol ($\lvert E\rvert \leq \infty$)	$0,70$	02:01:37 ($\sigma =$ 00:30:13)	00:00:41 ($\sigma =$ 00:00:08)

influence on the running times. It is now possible to calculate the representatives of medium values of t for some reasonable maximum edge counts in the context of drug discovery. For example, the drug-like CH*I*PMUNK Heterocycle CoMol dataset was processed in less than three minutes for a relative size threshold of $t = 0.5$.

In addition to the hard instances, the unfiltered CH*I*PMUNK Complete dataset, as well as the PubChem compound library filtered to a maximal edge count of 50 (resulting in a dataset of 98 941 008 graphs) were processed by algorithm 17. The results are shown in table 4.6. It stands out, that the PubChem instance with $t = 0.5$ and close to 100 million graphs was processed in less than 10 minutes. Also, the complete CH*I*PMUNK library with almost 19 million graphs was processed in less than one and a half minutes for $t = 0.3$. Both results demonstrate the scalability of the approach to very large big data instances.

4.6.11 Application to StruClus

To demonstrate the usefulness of the representative mining approach, it is utilized as a pre-clustering routine for the StruClus algorithm (cf., chapter 3).

The pre-clustering of StruClus serves as an initial lightweight clustering of the dataset that should at best align with some structural commonalities. Thus, the algorithm can build upon these commonalities in the later iterative refinement. However, the existing approach (cf., section 3.2.5) is limited in two ways. First, it does not guarantee the existence of a common substructure in a cluster, since the assignment to each element of the α-orthogonal maximal frequent pattern set $\mathfrak{M}$ is performed using a similarity. Even if such a similarity would be based on common substructures, low similarity assignments could still be based on different

Table 4.6: Running Times w.r.t. the distributed algorithm 17 for a selection of big instances. Running Times are evaluated for a single solution element ($k = 1$). The sample size for algorithms 12 and 17 was set to $s = 10\,000$. Algorithm 17 was evaluated with $\texttt{activeWorkers}_{\min} = 75\%$ and $W = 16$. Running Times are given in hours:minutes:seconds.

Dataset	t	Running Time Phase 1	Running Time Phase 2
CHIPMUNK Complete ($\lvert E\rvert \leq \infty$)	$0,30$	00:00:56 (σ = 00:00:16)	00:02:22 (σ = 00:00:20)
CHIPMUNK Complete ($\lvert E\rvert \leq \infty$)	$0,50$	00:03:04 (σ = 00:00:31)	00:02:24 (σ = 00:00:19)
CHIPMUNK Complete ($\lvert E\rvert \leq \infty$)	$0,70$	00:53:35 (σ = 00:03:08)	00:03:44 (σ = 00:00:28)
PubChem Compounds ($\lvert E\rvert \leq 50$)	$0,30$	00:00:40 (σ = 00:00:01)	00:07:52 (σ = 00:00:01)
PubChem Compounds ($\lvert E\rvert \leq 50$)	$0,50$	00:01:27 (σ = 00:00:09)	00:07:53 (σ = 00:00:02)
PubChem Compounds ($\lvert E\rvert \leq 50$)	$0,70$	00:53:18 (σ = 00:14:12)	00:20:59 (σ = 00:07:48)

substructures. Second, $\mathfrak{M}$ does not necessarily contain a similar pattern for each dataset graph. While the patterns themselves are dissimilar, they might stem from a subset of the dataset, leaving the rest of the dataset unrepresented. Such a situation can occur if the unrepresented graphs do not contain maximal frequent subgraphs for the chosen frequency threshold.

The representative mining, as defined in problem 4.1 (MAX-CSPC), can solve these issues. When generating a representative set, each pattern P of the solution set S represents a certain portion $\mathcal{G}_{\triangleright P}$ of the dataset and P is a common substructure in $\mathcal{G}_{\triangleright P}$. Furthermore, the objective is to maximize the number of covered dataset graphs. Thus, the issue of unrepresented graphs is avoided.

The adoption of problem 4.1 (MAX-CSPC) as pre-clustering step for StruClus looks as follows. The greedy algorithm 6 is used to iteratively calculate a subgraph pattern $P_{\max}$ with maximum utility. Then, $\mathcal{G}'_{\triangleright P_{\max}}$ in line 17 is used to create a cluster with the common subgraph pattern $P_{\max}$ and the procedure is repeated for the remaining graphs $\mathcal{G}' \setminus \mathcal{G}'_{\triangleright P_{\max}}$. Instead of using a fixed cardinality threshold k in line 4, the algorithm is modified to stop when $\texttt{utility}_{\max}$ drops below a minimum utility threshold $\texttt{utility}_{\min}$. This minimum threshold guarantees, that noise graphs are not added to very small or singleton clusters.

The evaluation is performed on the synthetic dataset with cardinality 10 000 (cf., section 3.2.9.2). The minimum utility threshold was set to $\texttt{utility}_{\min} = 50$. The initial relative size threshold for the subgraph coverage relation $\lhd_t$ was set to $t = 0.2$. In the cluster splitting step of StruClus (cf., section 3.2.4.1) the pre-clustering is reused. In this step, the relative size threshold was set to match the average homogeneity of the existing clusters. This guarantees, that newly created clusters are not immediately split in the following iteration. The test environment is identical to the setting in the StruClus evaluation (cf., section 3.2.9.1).

Table 4.7: Comparison of average running times and quality values for StruClus w.r.t. the classical and coverage-based pre-clustering routines. Running Times are given in hours.

$\mathcal{G}$	Classical Pre-Clustering			Coverage Pre-Clustering		
	Running Time	NVI	FM	Running Time	NVI	FM
Synthetic 10 000	**0.19**	0.95	0.87	18.55	**0.99**	**0.95**

Table 4.7 shows the results of the experiment. It is noticeable, that the coverage based pre-clustering is significantly slower than the classical variant as defined in section 3.2.5 with an absolute running time of over 18 hours compared to just 11.4 minutes. The other two datasets used during the evaluation of StruClus (Heterocyclic and AnchorQuery) had running times above 24 hours and are therefore left out in this evaluation. While the sampling algorithm 17 can improve overall running time for large-scale datasets, the tested instance has already only the size of the default sample Cardinality in the previous experiments. Thus, while scalability to large-scale datasets is still expected, the coverage-based pre-clustering adds a large constant to the running time.

From the quality point of view, the coverage-based pre-clustering makes a significant difference. With a score of 0.99 for the NVI measure and a score of 0.95 for the FM measure, StruClus does almost perfectly reproduce the ground truth. This is remarkable since the results of StruClus with classical pre-clustering already outperformed other competitors with a fair amount of difference (cf., section 3.2.9.8).

4.7 Summary

A novel coverage-based subgraph pattern representative objective (cf., problem 4.1 (MAX-CSPC)) has been proposed. In contrast to existing representative subgraph pattern mining algorithms, which focus on the diversification of (maximal) frequent subgraph patterns, the objective is orientated toward the representation of the dataset graphs directly. The flexible definition of the subgraph pattern coverage relation allows the integration of addition constraints w.r.t. the application field. For example, the (aromatic) ring structures in molecules are significantly different from chains w.r.t. to chemical properties, since the corresponding molecular parts are either relatively stable or flexible. Such an insight is easy to integrate into the coverage relation, e.g., by adding a constraint that rings must be preserved in a representative subgraph pattern.

It was shown, that problem 4.1 (MAX-CSPC) is a special case of the maximum coverage problem and that the problem is NP-hard. To the best of my knowledge, this is the first scalable representative subgraph pattern mining algorithm with proven approximation ratios for the representative objective. A novel sampling approach leads to a constant-time algorithm to generate candidate sets, which contain an

optimal pattern with a stochastically bound error. The experimental evaluation shows, that even in the case of false results, the results are of high quality and close to the optimum. The sampling approach enables a streaming and a distributed algorithm for the computation of the objective. The steaming implementation is able to accelerate the baseline algorithm by one to two orders of magnitude for moderately sized datasets with an increasing margin for larger datasets. The distributed variant utilizes the sampling approach to be the first distributed subgraph pattern mining algorithm that distributes the pattern space exploration and the dataset graphs at the same time. The distributed algorithm is able to speed up the computation by a factor of two magnitudes for 16 workers with 10 cores for instances with a sufficient amount of work. The overall scalability was demonstrated by the processing of approximately 100 million graphs of the PubChem dataset in just a few minutes per iteration. While the representative objective is useful in more general terms (e.g., in visual analytics of molecular datasets), it was demonstrated, that the representative mining objective can significantly increase the quality of the StruClus algorithm.

Chapter 5

Conclusions

It has been shown that structural analysis of molecular datasets for drug discovery can be extended to very large-scale datasets without sacrificing quality. Sampling strategies for datasets play an important role to overcome the high computational costs of graph-theoretical concepts in this context. The avoidance of intermediate vectorial or distance-based representations of graph datasets can lead to more consistent and interpretable data mining results. Parallelization and distribution of algorithms can play an important role to make the running times applicable for visual analytics processes. The Scaffold Hunter application was presented as an example of a typical drug-discovery workflow in the early stages. To infer existing knowledge about molecular graphs to de-novo datasets, structural summaries and unsupervised classification are identified as key tools to work with large-scale datasets. As such, the presented scalable approaches enable chemists to advance into previously undiscovered regions of the chemical space, paving the way to novel drug discoveries.

Bibliography

[21] *Most Cited Computer Science Articles*. 2021. URL: https://citeseerx.ist.psu.edu/stats/articles (visited on 01/21/2021) (cit. on p. 38).

[Agg+07] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. J. Zaki. "Xproj: a framework for projected structural clustering of xml documents". In: *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*. Ed. by P. Berkhin, R. Caruana, and X. Wu. ACM Press, Aug. 2007, pp. 46–55. DOI: 10.1145/1281192.1281201 (cit. on pp. 69, 74, 105).

[Agg+99] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. "Fast Algorithms for Projected Clustering". In: *Proceedings of the international conference on Management of data*. Ed. by A. Delis, C. Faloutsos, and S. Ghandeharizadeh. ACM SIGMOD 1999. ACM Press, June 1999, pp. 61–72. DOI: 10.1145/304182.304188 (cit. on pp. 62, 89).

[Agr+98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In: *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. Ed. by L. M. Haas and A. Tiwary. ACM Press, 1998, pp. 94–105. DOI: 10.1145/276304.276314 (cit. on pp. 57, 62).

[AH14] C. C. Aggarwal and J. Han. *Frequent Pattern Mining*. Springer Publishing Company, Incorporated, 2014 (cit. on p. 33).

[Ams+15] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K.-i. Kawarabayashi, and M. Nett. "Estimating Local Intrinsic Dimensionality". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 29–38. DOI: 10.1145/2783258.2783405 (cit. on p. 62).

[And73] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973 (cit. on pp. 56 sq.).

[Ank+99] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. "OPTICS: Ordering Points To Identify the Clustering Structure". In: *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*. Ed. by A. Delis, C. Faloutsos, and S. Ghandeharizadeh. ACM Press, 1999, pp. 49–60. DOI: 10.1145/304182.304187 (cit. on pp. 57, 59).

[AS94] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. Morgan Kaufmann Publishers Inc., 1994, pp. 487–499 (cit. on pp. 38 sq.).

[Ass+07] I. Assent, R. Krieger, E. Müller, and T. Seidl. "DUSC: Dimensionality Unbiased Subspace Clustering". In: *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*. IEEE Computer Society, 2007, pp. 409–414. DOI: 10.1109/ICDM.2007.49 (cit. on p. 62).

[AW10] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. 1st. Springer Publishing Company, Incorporated, 2010 (cit. on p. 33).

[BA83] H. Bunke and G. Allermann. "Inexact graph matching for structural pattern recognition". In: *Pattern Recognit. Lett.* 1.4 (1983), pp. 245–253. DOI: 10.1016/0167-8655(83)90033-8 (cit. on p. 64).

[Bad+14] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. "Streaming Submodular Maximization: Massive Data Summarization on the Fly". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, 2014, pp. 671–680. DOI: 10.1145/2623330.2623637 (cit. on pp. 26 sq., 30 sq., 131).

[BB02] C. Borgelt and M. R. Berthold. "Mining Molecular Fragments: Finding Relevant Substructures of Molecules". In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 2002, pp. 51–58. DOI: 10.1109/ICDM.2002.1183885 (cit. on p. 47).

[Ben+13] A. P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krüger, Y. Light, L. Mak, S. McGlinchey, M. Nowotka, G. Papadatos, R. Santos, and J. P. Overington. "The ChEMBL bioactivity database: an update". In: *Nucleic Acids Research* 42.D1 (Nov. 2013), pp. D1083–D1090. DOI: 10.1093/nar/gkt1031 (cit. on pp. 101, 118).

[Bey+99] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. "When Is "Nearest Neighbor" Meaningful?" In: *Proceedings of the 7th International Conference on Database Theory*. ICDT '99. Springer-Verlag, 1999, pp. 217–235 (cit. on p. 61).

[BH14] M. A. Bhuiyan and M. A. Hasan. "FSM-H: Frequent Subgraph Mining Algorithm in Hadoop". In: *2014 IEEE International Congress on Big Data*. June 2014, pp. 9–16. DOI: 10.1109/BigData.Congress.2014.12 (cit. on p. 48).

[BH15] M. A. Bhuiyan and M. A. Hasan. "An Iterative MapReduce Based Frequent Subgraph Mining Algorithm". In: *IEEE Transactions on Knowledge and Data Engineering* 27.3 (Mar. 2015), pp. 608–620. DOI: 10.1109/TKDE.2014.2345408 (cit. on pp. 48, 149).

[BL10] Y. Bilu and N. Linial. "Are Stable Instances Easy?" In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*. Ed. by A. C. Yao. Tsinghua University Press, 2010, pp. 332–341 (cit. on p. 60).

[Bor07] C. Borgelt. "Canonical Forms for Frequent Graph Mining". In: *Advances in Data Analysis: Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006*. Ed. by R. Decker and H. .-. Lenz. Springer Berlin Heidelberg, 2007, pp. 337–349. DOI: 10.1007/978-3-540-70981-7_38 (cit. on pp. 50 sqq., 117, 144).

[Bro09] N. Brown. "Chemoinformatics - an introduction for computer scientists". In: *ACM Comput. Surv.* 41.2 (2009), 8:1–8:38. DOI: 10.1145/1459352.1459353 (cit. on p. 3).

[BS98] H. Bunke and K. Shearer. "A graph distance metric based on the maximal common subgraph". In: *Pattern Recognition Letters* 19.3-4 (1998), pp. 255–259. DOI: 10.1016/S0167-8655(97)00179-7 (cit. on pp. 5, 64, 85).

[Bun97] H. Bunke. "On a relation between graph edit distance and maximum common subgraph". In: *Pattern Recognition Letters* 18.8 (1997), pp. 689–694. DOI: 10.1016/S0167-8655(97)00060-3 (cit. on p. 64).

[CFZ99] C. H. Cheng, A. W. Fu, and Y. Zhang. "Entropy-based Subspace Clustering for Mining Numerical Data". In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*. Ed. by U. M. Fayyad, S. Chaudhuri, and D. Madigan. ACM, 1999, pp. 84–93. DOI: 10.1145/312129.312199 (cit. on p. 63).

[Cha+08] V. Chaoji, M. A. Hasan, S. Salem, J. Besson, and M. J. Zaki. "ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns". In: *Statistical Analysis and Data Mining* 1.2 (2008), pp. 67–84. DOI: 10.1002/sam.10004 (cit. on p. 86).

[Che+07] J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi. "ChemDB update—full-text search and virtual chemical space". In: *Bioinformatics* 23.17 (June 2007), pp. 2348–2351. DOI: 10.1093/bioinformatics/btm341 (cit. on p. 118).

[CN01] E. Chávez and G. Navarro. "A Probabilistic Spell for the Curse of Dimensionality". In: *Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments*. Ed. by A. L. Buchsbaum and J. Snoeyink. Vol. 2153. Lecture Notes in Computer Science. ALENEX 2001. Springer, Jan. 2001, pp. 147–160. DOI: 10.1007/3-540-44808-X_12 (cit. on p. 62).

[Cor+09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009 (cit. on pp. 14, 43 sq.).

[Cov74] T. M. Cover. "The Best Two Independent Measurements Are Not the Two Best". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-4.1 (1974), pp. 116–117. DOI: 10.1109/TSMC.1974.5408535 (cit. on p. 107).

[Dav91] R. N. Davé. "Characterization and detection of noise in clustering". In: *Pattern Recognit. Lett.* 12.11 (1991), pp. 657–664. DOI: 10.1016/0167-8655(91)90002-4 (cit. on p. 60).

[DE84] W. Day and H. Edelsbrunner. "Efficient algorithms for agglomerative hierarchical clustering methods". In: *Journal of Classification* 1.1 (1984), pp. 7–24. DOI: 10.1007/BF01890115 (cit. on pp. 56 sq.).

[DG04] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*. Ed. by E. A. Brewer and P. Chen. USENIX Association, 2004, pp. 137–150 (cit. on p. 17).

[DG08] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. DOI: 10.1145/1327452.1327492 (cit. on pp. 17 sq.).

[Dia+19] V. Dias, C. H. C. Teixeira, D. Guedes, W. Meira, and S. Parthasarathy. "Fractal: A General-Purpose Graph Pattern Mining System". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 1357–1374. DOI: 10.1145/3299869.3319875 (cit. on pp. 50, 153).

[DKM17] A. Droschinsky, N. M. Kriege, and P. Mutzel. "Finding Largest Common Substructures of Molecules in Quadratic Time". In: *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings*. Ed. by B. Steffen, C. Baier, M. van den Brand, J. Eder, M. Hinchey, and T. Margaria. Vol. 10139. Lecture Notes in Computer Science. Springer, 2017, pp. 309–321. DOI: 10.1007/978-3-319-51963-0_24 (cit. on p. 5).

[DKM18] A. Droschinsky, N. M. Kriege, and P. Mutzel. "Largest Weight Common Subtree Embeddings with Distance Penalties". In: *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*. Ed. by I. Potapov, P. G. Spirakis, and J. Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 54:1–54:15. DOI: 10.4230/LIPIcs.MFCS.2018.54 (cit. on pp. 5, 64).

[DTK98] L. Dehaspe, H. Toivonen, and R. D. King. "Finding Frequent Substructures in Chemical Compounds". In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. KDD'98. New York, NY: AAAI Press, 1998, pp. 30–36 (cit. on p. 46).

[Dun73] J. C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046 (cit. on p. 56).

[Dur+02] J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse. "Reoptimization of MDL Keys for Use in Drug Discovery". In: *J. Chem. Inf. Comput. Sci.* 42.5 (2002), pp. 1273–1280. DOI: 10.1021/ci010132r (cit. on p. 63).

[ER12] P. Ertl and B. Rohde. "The Molecule Cloud - compact visualization of large collections of molecules". In: *J. Cheminformatics* 4 (2012), p. 12. DOI: 10.1186/1758-2946-4-12 (cit. on p. 9).

[Est+96] M. Ester, H. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. Ed. by E. Simoudis, J. Han, and U. M. Fayyad. AAAI Press, 1996, pp. 226–231 (cit. on pp. 57, 59 sq.).

[Fei98] U. Feige. "A Threshold of ln n for Approximating Set Cover". In: *Journal of the ACM* 45.4 (July 1998), pp. 634–652. DOI: 10.1145/285055.285059 (cit. on pp. 27, 29).

[Fer+09] M. Ferrer, E. Valveny, F. Serratosa, I. Bardají, and H. Bunke. "Graph-Based k-Means Clustering: A Comparison of the Set Median versus the Generalized Median Graph". In: *Computer Analysis of Images and Patterns*. Ed. by N. P. Xiaoyi Jiang. Vol. 5702. Lecture Notes in Computer Science. CAIP 2009. Springer, Sept. 2009, pp. 342–350. DOI: 10.1007/978-3-642-03767-2_42 (cit. on pp. 37, 69).

[FM83] E. B. Fowlkes and C. L. Mallows. "A Method for Comparing Two Hierarchical Clusterings". In: *Journal of the American Statistical Association* 78.383 (1983), pp. 553–569 (cit. on p. 65).

[FO71] K. Fukunaga and D. R. Olsen. "An Algorithm for Finding Intrinsic Dimensionality of Data". In: *IEEE Trans. Computers* 20.2 (1971), pp. 176–183. DOI: 10.1109/T-C.1971.223208 (cit. on p. 62).

[FPS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. "From data mining to knowledge discovery in databases". In: *AI magazine* 17.3 (1996), p. 37 (cit. on p. 33).

[FV01] M.-L. Fernández and G. Valiente. "A graph distance metric combining maximum common subgraph and minimum common supergraph". In: *Pattern Recognition Letters* 22.6–7 (2001), pp. 753–758. DOI: http://dx.doi.org/10.1016/S0167-8655(01)00017-4 (cit. on pp. 5, 64).

[GC20] W. Gao and C. W. Coley. "The Synthesizability of Molecules Proposed by Generative Models". In: *Journal of Chemical Information and Modeling* 60.12 (2020), pp. 5714–5723. DOI: 10.1021/acs.jcim.0c00174 (cit. on p. 3).

[Gir02] M. A. Girolami. "Mercer kernel-based clustering in feature space". In: *IEEE Trans. Neural Networks* 13.3 (2002), pp. 780–784. DOI: 10.1109/TNN.2002.1000150 (cit. on pp. 58 sq., 89).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (cit. on pp. 32, 52).

[GRB06] P. Gedeck, B. Rohde, and C. Bartels. "QSAR - How Good Is It in Practice? Comparison of Descriptor Sets on an Unbiased Cross Section of Corporate Data Sets". In: *J. Chem. Inf. Model.* 46.5 (2006), pp. 1924–1936. DOI: 10.1021/ci050413p (cit. on p. 63).

[Har55] F. Harary. "The number of linear, directed, rooted, and connected graphs". In: *Trans. Amer. Math. Soc.* 78 (1955), pp. 445–463 (cit. on pp. 35, 37).

[Has+07] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. "ORIGAMI: Mining Representative Orthogonal Graph Patterns". In: *Proceedings of the 7th International Conference on Data Mining*. ICDM 2007. IEEE. IEEE Computer Society, Oct. 2007, pp. 153–162. DOI: 10.1109/ICDM.2007.45 (cit. on pp. 54, 75, 106 sqq.).

[HBD07] T. Horváth, B. Bringmann, and L. De Raedt. "Frequent Hypergraph Mining". In: *Inductive Logic Programming*. Ed. by S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad. Springer Berlin Heidelberg, 2007, pp. 244–259 (cit. on p. 52).

[HRR98] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. "Computing on data streams". In: *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*. Ed. by J. M. Abello and J. S. Vitter. Vol. 50. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1998, pp. 107–118. DOI: 10.1090/dimacs/050/05 (cit. on p. 164).

[HRW10] T. Horváth, J. Ramon, and S. Wrobel. "Frequent Subgraph Mining in Outerplanar Graphs". In: *Data Min. Knowl. Discov.* 21.3 (Nov. 2010), pp. 472–508. DOI: 10.1007/s10618-009-0162-1 (cit. on p. 52).

[HS06] H. He and A. K. Singh. "GraphRank: Statistical Modeling and Mining of Significant Subgraphs in the Feature Space". In: *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*. IEEE Computer Society, 2006, pp. 885–890. DOI: 10.1109/ICDM.2006.79 (cit. on p. 106).

[Hua+04] J. Huan, W. Wang, J. Prins, and J. Yang. "SPIN: Mining Maximal Frequent Subgraphs from Graph Databases". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: Association for Computing Machinery, 2004, pp. 581–586. DOI: 10.1145/1014052.1014123 (cit. on p. 48).

[Hum+18] L. Humbeck, S. Weigang, T. Schäfer, P. Mutzel, and O. Koch. "CHIPMUNK: A Virtual Synthesizable Small-Molecule Library for Medicinal Chemistry, Exploitable for Protein–Protein Interaction Modulators". In: *ChemMedChem* 13.6 (2018), pp. 532–539. DOI: 10.1002/cmdc.201700689 (cit. on pp. 3, 11, 90, 101, 118, 168).

[HWP03] J. Huan, W. Wang, and J. Prins. "Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism". In: *Proceedings of the Third IEEE International Conference on Data Mining*. ICDM '03. IEEE Computer Society, 2003, p. 549 (cit. on p. 47).

[HZ13] X. Hui and L. Zhongmon. "Clustering Validation Measures". In: *Data clustering: algorithms and applications*. Ed. by C. C. Aggarwal and C. K. Reddy. CRC Press, 2013. Chap. 23, pp. 571–605 (cit. on p. 65).

[Irw+12] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. "ZINC: A Free Tool to Discover Chemistry for Biology". In: *Journal of Chemical Information and Modeling* 52.7 (2012), pp. 1757–1768. DOI: 10.1021/ci3001277 (cit. on p. 101).

[IWM00] A. Inokuchi, T. Washio, and H. Motoda. "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data". In: *Principles of Data Mining and Knowledge Discovery*. Ed. by D. A. Zighed, J. Komorowski, and J. Żytkow. Springer Berlin Heidelberg, 2000, pp. 13–23 (cit. on pp. 38, 46).

[JCZ13] C. Jiang, F. Coenen, and M. Zito. "A survey of frequent subgraph mining algorithms". In: *The Knowledge Engineering Review* 28.1 (2013), pp. 75–105. DOI: 10.1017/S0269888912000331 (cit. on pp. 35, 53).

[JK05] K. Jahn and S. Kramer. "Optimizing gSpan for molecular datasets". In: *Proceedings of the third international workshop on mining graphs, trees and sequences (MGTS-2005)*. Citeseer. 2005, pp. 77–89 (cit. on p. 47).

[JP73] R. A. Jarvis and E. A. Patrick. "Clustering Using a Similarity Measure Based on Shared Near Neighbors". In: *IEEE Trans. Computers* 22.11 (1973), pp. 1025–1034. DOI: 10.1109/T-C.1973.223640 (cit. on p. 57).

[JTL10] S. Jouili, S. Tabbone, and V. Lacroix. "Median Graph Shift: A New Clustering Algorithm for Graph Domain". In: *20th International Conference on Pattern Recognition*. Aug. 2010, pp. 950–953. DOI: 10.1109/ICPR.2010.238 (cit. on p. 69).

[Kim+20] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. E. Bolton. "PubChem in 2021: new data content and improved web interfaces". In: *Nucleic Acids Research* 49.D1 (Nov. 2020), pp. D1388–D1395. DOI: 10.1093/nar/gkaa971 (cit. on p. 3).

[Kim+21] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. Bolton. "PubChem in 2021: new data content and improved web interfaces". In: *Nucleic Acids Res.* 49.Database-Issue (2021), pp. D1388–D1395. DOI: 10.1093/nar/gkaa971 (cit. on p. 168).

[KK01] M. Kuramochi and G. Karypis. "Frequent subgraph discovery". In: *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, pp. 313–320. DOI: 10.1109/ICDM.2001.989534 (cit. on pp. 41, 46).

[KKK04] K. Kailing, H. Kriegel, and P. Kröger. "Density-Connected Subspace Clustering for High-Dimensional Data". In: *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*. Ed. by M. W. Berry, U. Dayal, C. Kamath, and D. B. Skillicorn. SIAM, 2004, pp. 246–256. DOI: 10.1137/1.9781611972740.23 (cit. on pp. 62 sq.).

[KKM11] K. Klein, N. Kriege, and P. Mutzel. "CT-Index: Fingerprint-based Graph Indexing Combining Cycles and Trees". In: *Proceedings of the 27th International Conference on Data Engineering*. ICDE 2011. IEEE. Apr. 2011, pp. 1115–1126. DOI: 10.1109/ICDE.2011.5767909 (cit. on pp. 63, 88, 117).

[Kle+13] K. Klein, O. Koch, N. Kriege, P. Mutzel, and T. Schäfer. "Visual Analysis of Biological Activity Data with Scaffold Hunter". In: *Molecular Informatics* 32.11-12 (2013), pp. 964–975. DOI: 10.1002/minf.201300087 (cit. on p. 11).

[KM12] N. M. Kriege and P. Mutzel. "Subgraph Matching Kernels for Attributed Graphs". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012 (cit. on p. 89).

[KMS14a] N. Kriege, P. Mutzel, and T. Schäfer. "Practical SAHN Clustering for Very Large Data Sets and Expensive Distance Metrics". In: *Journal of Graph Algorithms and Applications* 18.4 (2014), pp. 577–602. DOI: 10.7155/jgaa.00338 (cit. on pp. 7, 11, 67, 97).

[KMS14b] N. M. Kriege, P. Mutzel, and T. Schäfer. "SAHN Clustering in Arbitrary Metric Spaces Using Heuristic Nearest Neighbor Search". In: *Algorithms and Computation - 8th International Workshop, WALCOM 2014, Chennai, India, February 13-15, 2014, Proceedings*. Ed. by S. P. Pal and K. Sadakane. Vol. 8344. Lecture Notes in Computer Science. Springer, 2014, pp. 90–101. DOI: 10.1007/978-3-319-04657-0_11 (cit. on p. 11).

[Kra+05] A. Kraskov, H. Stögbauer, R. G. Andrzejak, and P. Grassberger. "Hierarchical clustering using mutual information". In: *Europhysics Letters (EPL)* 70.2 (Apr. 2005), pp. 278–284. DOI: 10.1209/epl/i2004-10483-y (cit. on p. 65).

[Kri15] N. M. Kriege. "Comparing Graphs: Algorithms & Applications". PhD thesis. TU Dortmund, 2015 (cit. on p. 4).

[Lin89] A. Lingas. "Subgraph Isomorphism for Biconnected Outerplanar Graphs in Cubic Time". In: *Theor. Comput. Sci.* 63.3 (Mar. 1989), pp. 295–302. DOI: 10.1016/0304-3975(89)90011-X (cit. on p. 52).

[Llo82] S. P. Lloyd. "Least squares quantization in PCM". In: *IEEE Trans. Inf. Theory* 28.2 (1982), pp. 129–136. DOI: 10.1109/TIT.1982.1056489 (cit. on pp. 55 sqq., 59).

[Lup59] O. B. Lupanov. "Asymptotic estimates of the number of graphs with n edges". In: *Dokl. Akad. Nauk* 126.3 (1959), pp. 298–500 (cit. on pp. 35, 37).

[Lup62] O. B. Lupanov. "An asymptotic estimate of the number of graphs and networks with n edges," in: *Trans: Prob. Cyber* 4 (1962), pp. 1137–1155 (cit. on p. 35).

[LW15] R. Li and W. Wang. "REAFUM: Representative Approximate Frequent Subgraph Mining". In: *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*. Ed. by S. Venkatasubramanian and J. Ye. SIAM, 2015, pp. 757–765. DOI: 10.1137/1.9781611974010.85 (cit. on p. 106).

[LWH03] B. Luo, R. C. Wilson, and E. R. Hancock. "Spectral embedding of graphs". In: *Pattern Recognit.* 36.10 (2003), pp. 2213–2230. DOI: 10.1016/S0031-3203(03)00084-0 (cit. on p. 63).

[LXG14] W. Lin, X. Xiao, and G. Ghinita. "Large-scale frequent subgraph mining in MapReduce". In: *2014 IEEE 30th International Conference on Data Engineering*. Mar. 2014, pp. 844–855. DOI: 10.1109/ICDE.2014.6816705 (cit. on pp. 49, 77, 149).

[Mag+14] G. Maggiora, M. Vogt, D. Stumpfe, and J. Bajorath. "Molecular Similarity in Medicinal Chemistry". In: *Journal of Medicinal Chemistry* 57.8 (2014), pp. 3186–3204. DOI: 10.1021/jm401411z (cit. on p. 3).

[MBB04a] T. Meinl, C. Borgelt, and M. Berthold. "Discriminative Closed Fragment Mining and Perfect Extensions in MoFa". In: *STAIRS 2004: Proceedings of the Second Starting AI Researchers ' Symposium*. Ed. by E. Onaindia. Frontiers in artificial intelligence and applications 109. IOS Press, 2004, pp. 3–14 (cit. on p. 47).

[MBB04b] T. Meinl, C. Borgelt, and M. R. Berthold. "Mining Fragments with Fuzzy Chains in Molecular Databases". In: *Proc. of the Workshop W7 on Mining Graphs, Trees and Sequences*. Ed. by J. Kok and T. Washio. 2004, pp. 49–60 (cit. on p. 47).

[Mei07] M. Meilă. "Comparing clusterings—an information based distance". In: *Journal of multivariate analysis* 98.5 (2007), pp. 873–895 (cit. on p. 65).

[Men+11] X.-Y. Meng, H.-X. Zhang, M. Mezei, and M. Cui. "Molecular Docking: A Powerful Approach for Structure-Based Drug Discovery". In: *Current Computer-Aided Drug Design* 7.2 (2011), pp. 146–157. DOI: 10.2174/157340911795677602 (cit. on p. 2).

[Mir+13] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. "Distributed Submodular Maximization: Identifying Representative Elements in Massive Data". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 2049–2057 (cit. on pp. 30, 131).

[Mör07] F. Mörchen. "Unsupervised Pattern Mining from Symbolic Temporal Data". In: *SIGKDD Explor. Newsl.* 9.1 (June 2007), pp. 41–55. DOI: 10.1145/1294301.1294302 (cit. on p. 106).

[MS11] G. M. Maggiora and V. Shanmugasundaram. "Molecular Similarity Measures". In: *Chemoinformatics and Computational Chemical Biology*. Ed. by J. Bajorath. Humana Press, 2011, pp. 39–100. DOI: 10.1007/978-1-60761-839-3_2 (cit. on p. 4).

[Mül+09] E. Müller, S. Günnemann, I. Assent, and T. Seidl. "Evaluating Clustering in Subspace Projections of High Dimensional Data". In: *Proc. 35th International Conference on Very Large Data Bases*. Vol. 2. PVLDB Journal 1. VLDB 2009. VLDB Endowment, 2009, pp. 1270–1281 (cit. on p. 97).

[MZ15] V. Mirrokni and M. Zadimoghaddam. "Randomized Composable Coresets for Distributed Submodular Maximization". In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*. STOC '15. Portland, Oregon, USA: ACM, 2015, pp. 153–162. DOI: 10.1145/2746539.2746624 (cit. on pp. 30, 131).

[NEB10] X. V. Nguyen, J. Epps, and J. Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *Journal of Machine Learning Research* 11 (2010), pp. 2837–2854 (cit. on p. 66).

[NK04] S. Nijssen and J. Kok. "Frequent graph mining and its application to molecular databases". In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. Vol. 5. 2004, 4571–4577 vol.5. DOI: 10.1109/ICSMC.2004.1401252 (cit. on p. 48).

[NL91] B. Nitzberg and V. M. Lo. "Distributed Shared Memory: A Survey of Issues and Algorithms". In: *Computer* 24.8 (1991), pp. 52–60. DOI: 10.1109/2.84877 (cit. on p. 17).

[NM17] S. Neumann and P. Miettinen. "Reductions for Frequency-Based Data Mining Problems". In: *2017 IEEE International Conference on Data Mining (ICDM)*. Nov. 2017, pp. 997–1002. DOI: 10.1109/ICDM.2017.128 (cit. on p. 52).

[NR16] D. Natarajan and S. Ranu. "A Scalable and Generic Framework to Mine Top-k Representative Subgraph Patterns". In: *16th International Conference on Data Mining*. Dec. 2016, pp. 370–379. DOI: 10.1109/ICDM.2016.0048 (cit. on pp. 106 sq.).

[NR18] D. Natarajan and S. Ranu. "Resling: a scalable and generic framework to mine top-k representative subgraph patterns". In: *Knowledge and Information Systems* 54.1 (Jan. 2018), pp. 123–149 (cit. on pp. 106 sq.).

[NWF78] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical Programming* 14.1 (Dec. 1978), pp. 265–294. DOI: 10.1007/BF01588971 (cit. on pp. i, 10, 27–30, 112, 117, 142).

[Pan+15] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang. "Finding the best not the most: regularized loss minimization subgraph selection for graph classification". In: *Pattern Recognition* 48.11 (2015), pp. 3783–3796. DOI: 10.1016/j.patcog.2015.05.019 (cit. on p. 54).

[PBM16] M. Piernik, D. Brzezinski, and T. Morzy. "Clustering XML documents by patterns". In: *Knowl. Inf. Syst.* 46.1 (2016), pp. 185–212. DOI: 10.1007/s10115-015-0820-0 (cit. on p. 69).

[PJR17] A. Petermann, M. Junghanns, and E. Rahm. "DIMSpan: Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems". In: *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. BDCAT '17. Austin, Texas, USA: ACM, 2017, pp. 237–246. DOI: 10.1145/3148055.3148064 (cit. on pp. 49, 149).

[PLD05] H. Peng, F. Long, and C. H. Q. Ding. "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.8 (2005), pp. 1226–1238. DOI: 10.1109/TPAMI.2005.159 (cit. on p. 107).

[PM06] A. Patrikainen and M. Meila. "Comparing Subspace Clusterings". In: *IEEE Trans. Knowl. Data Eng.* 18.7 (2006), pp. 902–916. DOI: 10.1109/TKDE.2006.106 (cit. on p. 97).

[RH10] D. Rogers and M. Hahn. "Extended-Connectivity Fingerprints". In: *Journal of Chemical Information and Modeling* 50.5 (2010). PMID: 20426451, pp. 742–754. DOI: 10.1021/ci100050t (cit. on pp. 4, 63).

[RHS14] S. Ranu, M. Hoang, and A. Singh. "Answering Top-k Representative Queries on Graph Databases". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. Snowbird, Utah, USA: ACM, 2014, pp. 1163–1174. DOI: 10.1145/2588555.2610524 (cit. on pp. 106 sq.).

[Rud+12] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond. "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". In: *Journal of Chemical Information and Modeling* 52.11 (2012), pp. 2864–2875. DOI: 10.1021/ci300415d (cit. on p. 3).

[Sav98] J. E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998 (cit. on p. 14).

[Sch+07] A. Schuffenhauer, P. Ertl, S. Roggo, S. Wetzel, M. A. Koch, and H. Waldmann. "The Scaffold Tree - Visualization of the Scaffold Universe by Hierarchical Scaffold Classification". In: *Journal of Chemical Information and Modeling* 47.1 (2007), pp. 47–58. DOI: 10.1021/ci600338x (cit. on p. 7).

[Sch+17] T. Schäfer, N. M. Kriege, L. Humbeck, K. Klein, O. Koch, and P. Mutzel. "Scaffold Hunter: a comprehensive visual analytics framework for drug discovery". In: *J. Cheminformatics* 9.1 (2017), 28:1–28:18. DOI: 10.1186/s13321-017-0213-3 (cit. on pp. 5, 11).

[See+10] M. Seeland, T. Girschick, F. Buchwald, and S. Kramer. "Online Structural Graph Clustering Using Frequent Subgraph Mining". In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III*. 2010, pp. 213–228. DOI: 10.1007/978-3-642-15939-8_14 (cit. on pp. 56, 69).

[See+11] M. Seeland, S. A. Berger, A. Stamatakis, and S. Kramer. "Parallel Structural Graph Clustering". In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*. ECML PKDD 2011. Sept. 2011, pp. 256–272. DOI: 10.1007/978-3-642-23808-6_17 (cit. on p. 97).

[SF83] A. Sanfeliu and K. S. Fu. "A Distance measure between attributed relational graphs for pattern recognition." In: *IEEE Transactions on Systems, Man, and Cybernetics* 13.3 (1983), pp. 353–362 (cit. on pp. 4, 64).

[Sha95] J. P. Shaffer. "Multiple hypothesis testing". In: *Annual review of psychology* 46.1 (1995), pp. 561–584 (cit. on p. 24).

[She+09] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. "Efficient graphlet kernels for large graph comparison". In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*. Ed. by D. A. V. Dyk and M. Welling. Vol. 5. JMLR Proceedings. AISTATS 2009. JMLR.org, Apr. 2009, pp. 488–495 (cit. on pp. 63, 98).

[She+11] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. "Weisfeiler-Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12 (2011), pp. 2539–2561 (cit. on p. 63).

[SI21] N. J. A. Sloane and T. O. F. Inc. *The on-line encyclopedia of integer sequences*. 2021 (cit. on p. 36).

[SKK14] M. Seeland, A. Karwath, and S. Kramer. "Structural clustering of millions of molecular graphs". In: *Symposium on Applied Computing*. SAC 2014. ACM, Mar. 2014, pp. 121–128. DOI: 10.1145/2554850.2555063 (cit. on pp. 69, 89, 97).

[SM17] T. Schäfer and P. Mutzel. "StruClus: Scalable Structural Graph Set Clustering with Representative Sampling". In: *Advanced Data Mining and Applications*. Ed. by G. Cong, W.-C. Peng, W. E. Zhang, C. Li, and A. Sun. Springer International Publishing, 2017, pp. 343–359 (cit. on pp. 11, 69).

[Stö+19] B. K. Stöcker, T. Schäfer, P. Mutzel, J. Köster, N. M. Kriege, and S. Rahmann. "Protein Complex Similarity Based on Weisfeiler-Lehman Labeling". In: *Similarity Search and Applications - 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2-4, 2019, Proceedings*. Ed. by G. Amato, C. Gennaro, V. Oria, and M. Radovanovic. Vol. 11807. Lecture Notes in Computer Science. Springer, 2019, pp. 308–322. DOI: 10.1007/978-3-030-32047-8_27 (cit. on pp. 11, 63, 118, 120).

[Stu+15] W. Sturm, T. Schäfer, T. Schreck, A. Holzinger, and T. Ullrich. "Extending the Scaffold Hunter Visualization Toolkit with Interactive Heatmaps". In: *Computer Graphics and Visual Computing (CGVC)*. Ed. by R. Borgo and C. Turkay. The Eurographics Association, 2015. DOI: 10.2312/cgvc.20151247 (cit. on pp. 7, 11).

[Sug+15] M. Sugiyama, F. Llinares-López, N. Kasenburg, and K. M. Borgwardt. "Significant Subgraph Mining with Multiple Testing Correction". In: *Proceedings of the 2015 SIAM International Conference on Data Mining*. Apr. 2015, pp. 37–45 (cit. on pp. 54, 63).

[Sys82] M. M. Sysło. "The subgraph isomorphism problem for outerplanar graphs". In: *Theoretical Computer Science* 17.1 (1982), pp. 91–97. DOI: https://doi.org/10.1016/0304-3975(82)90133-5 (cit. on p. 32).

[TdT16] A. Terada, D. duVerle, and K. Tsuda. "Significant Pattern Mining with Confounding Variables". In: *Advances in Knowledge Discovery and Data Mining: 20th Pacific-Asia Conference*. Ed. by J. Bailey, L. Khan, T. Washio, G. Dobbie, J. Z. Huang, and R. Wang. Springer International Publishing, 2016, pp. 277–289. DOI: 10.1007/978-3-319-31753-3_23 (cit. on pp. 54, 106).

[Tho+10] M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. J. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt. "Discriminative frequent subgraph mining with optimality guarantees". In: *Statistical Analysis and Data Mining* 3.5 (2010). CORK, pp. 302–318. DOI: 10.1002/sam.10084 (cit. on pp. 54, 63, 107).

[TK06] K. Tsuda and T. Kudo. “Clustering graphs by weighted substructure mining”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 953–960 (cit. on pp. 54, 69).

[TK08] K. Tsuda and K. Kurihara. “Graph Mining with Variational Dirichlet Process Mixture Models”. In: *Proceedings of the International Conference on Data Mining*. SDM 2008. SIAM. SIAM, Apr. 2008, pp. 432–442. DOI: 10.1137/1.9781611972788.39 (cit. on pp. 54, 69, 97).

[TK14] D. Taeger and S. Kuhnt. *Statistical Hypothesis Testing with SAS and R*. 1st. Wiley Publishing, 2014 (cit. on pp. 25, 136).

[TK70] J. Turner and W. H. Kautz. “A Survey of Progress in Graph Theory in the Soviet Union”. In: *SIAM Review* 12 (1970), pp. 1–68 (cit. on p. 35).

[TS22] E. Thordsen and E. Schubert. “ABID: Angle Based Intrinsic Dimensionality — Theory and analysis”. In: *Information Systems* (2022), p. 101989. DOI: https://doi.org/10.1016/j.is.2022.101989 (cit. on p. 62).

[TZ16] N. Talukder and M. J. Zaki. “Parallel graph mining with dynamic load balancing”. In: *2016 IEEE International Conference on Big Data (Big Data)*. Dec. 2016, pp. 3352–3359. DOI: 10.1109/BigData.2016.7840995 (cit. on pp. 49, 149, 153).

[Ull76] J. R. Ullmann. “An Algorithm for Subgraph Isomorphism”. In: *J. ACM* 23.1 (Jan. 1976), pp. 31–42. DOI: 10.1145/321921.321925 (cit. on p. 53).

[Wal+01] W. D. Wallis, P. Shoubridge, M. Kraetzl, and D. Ray. “Graph distances using graph union”. In: *Pattern Recognition Letters* 22.6/7 (2001), pp. 701–704 (cit. on pp. 5, 64).

[Wan+16] X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. “RPM: Representative Pattern Mining for Efficient Time Series Classification”. In: *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16*. Ed. by E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, and K. Stefanidis. OpenProceedings.org, 2016, pp. 185–196. DOI: 10.5441/002/edbt.2016.19 (cit. on p. 106).

[Wei68] A. A. L. Weisfeiler. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno - Technicheskaja Informatsia, Seria* 2 (1968) (cit. on p. 4).

[Wel20] P. Welke. “Efficient Frequent Subgraph Mining in Transactional Databases”. In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 2020, pp. 307–314. DOI: 10.1109/DSAA49011.2020.00044 (cit. on pp. 46, 52 sq.).

[WHW19] P. Welke, T. Horváth, and S. Wrobel. “Probabilistic and exact frequent subtree mining in graphs beyond forests”. In: *Mach. Learn.* 108.7 (2019), pp. 1137–1164. DOI: 10.1007/s10994-019-05779-1 (cit. on p. 53).

[Wil+17] E. L. Willighagen, J. W. Mayfield, J. Alvarsson, A. Berg, L. Carlsson, N. Jeliazkova, S. Kuhn, T. Pluskal, M. Rojas-Chertó, O. Spjuth, G. M. Torrance, C. T. A. Evelo, R. Guha, and C. Steinbeck. "The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching". In: *J. Cheminformatics* 9.1 (2017), 33:1–33:19. DOI: 10.1186/s13321-017-0220-4 (cit. on p. 119).

[Wör+05] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston". In: *Knowledge Discovery in Databases: PKDD 2005*. Ed. by A. M. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama. Springer Berlin Heidelberg, 2005, pp. 392–403 (cit. on pp. 44, 50, 53).

[YH02] X. Yan and J. Han. "gSpan: Graph-Based Substructure Pattern Mining". In: *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. 2002, pp. 721–724. DOI: 10.1109/ICDM.2002.1184038 (cit. on pp. 38, 47, 50 sq.).

[YH03] X. Yan and J. Han. "CloseGraph: Mining Closed Frequent Graph Patterns". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: Association for Computing Machinery, 2003, pp. 286–295. DOI: 10.1145/956750.956784 (cit. on p. 47).

[Zah+12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, Apr. 2012, pp. 15–28 (cit. on pp. 18 sqq.).

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*. Ed. by H. V. Jagadish and I. S. Mumick. ACM Press, 1996, pp. 103–114. DOI: 10.1145/233269.233324 (cit. on pp. 55, 97).

[ZSK12] A. Zimek, E. Schubert, and H.-P. Kriegel. "A survey on unsupervised outlier detection in high-dimensional numerical data". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5.5 (2012), pp. 363–387. DOI: https://doi.org/10.1002/sam.11161 (cit. on p. 61).

[ZYL09] S. Zhang, J. Yang, and S. Li. "RING: An Integrated Method for Frequent Representative Subgraph Mining". In: *Ninth IEEE International Conference on Data Mining*. Dec. 2009, pp. 1082–1087. DOI: 10.1109/ICDM.2009.96 (cit. on pp. 106 sq.).