

StruClus: Structural Clustering of Large-Scale Graph Databases

Till Schäfer

TU Dortmund University

Dept. of Computer Science
44227 Dortmund, Germany

Email: till.schaefer@cs.tu-dortmund.de

Petra Mutzel

TU Dortmund University

Dept. of Computer Science
44227 Dortmund, Germany

Email: petra.mutzel@cs.tu-dortmund.de

Abstract—We present a structural clustering algorithm for large-scale datasets of small labeled graphs, utilizing a frequent subgraph sampling strategy. A set of representatives provides an intuitive description of each cluster, supports the clustering process, and helps to interpret the clustering results. The projection-based nature of the clustering approach allows us to bypass dimensionality and feature extraction problems that arise in the context of graph datasets reduced to pairwise distances or feature vectors. While achieving high quality and (human) interpretable clusterings, the runtime of the algorithm only grows linearly with the number of graphs. Furthermore, the approach is easy to parallelize and therefore suitable for very large datasets. Our extensive experimental evaluation on synthetic and real world datasets demonstrates the superiority of our approach over existing structural and subspace clustering algorithms, both, from a runtime and quality point of view.

I. INTRODUCTION

Molecules, protein interaction networks, XML documents, social media interactions, and image segments all have in common that they can be modeled by labeled graphs. The ability to represent topological and semantic information causes graphs to be among the most versatile data structures in computer science. In the age of Big Data, huge amounts of graph data are collected and the demand to analyze them increases with its collection. We focus on the special case of clustering large sets of small labeled graphs. Our main motivation stems from the need to cluster large-scale molecular databases for drug discovery, such as PubChem¹, ChEMBL², ChemDB³ or synthetically constructed de-novo databases [1], which contain up to a billion molecules. However, the presented approach is not limited to this use case.

Clustering techniques aim to find homogeneous subsets in a set of objects. Classical approaches do not interpret the objects directly, but abstract them by utilizing some intermediate representation, such as feature vectors or pairwise distances. While the abstraction over pairwise distances is beneficial in terms of generality, it can be disadvantageous in the case of *intrinsic high dimensional* datasets [2, 3]. In this case the *concentration effect* may cause the pairwise distances to loose their relative contrast; i.e. the distances converge towards a

common value [4]. The concentration effect is closely related to a bad *clusterability* [5].

Sets of graphs are usually clustered by transforming the graphs to feature vectors or by using graph theoretic similarity measures.

Typical feature extraction methods for graphs are: counting *graphlets*, that is, small subgraphs [6, 7], counting *walks* [8], and using *eigenvectors* of adjacency matrices (spectral graph theory) [9]. The enumeration of all subgraphs is considered intractable even for graphs of moderate size, because there exist up to exponentially many subgraphs wrt the graph size. Many efficient clustering algorithms have been proposed for vector space. Therefore, the transformation to feature vectors might look beneficial in the first place. However, the above mentioned feature extraction methods tend to produce a large amount of distinct and often unrelated features. This results in datasets with a high intrinsic dimensionality [10, 11]. Additionally, the extracted features only approximate the graph structure, which implies that feature vectors cannot be transformed back into a graph. Hence, the interpretability of clustering algorithms which perform vector modifications (e.g. calculating centroids) is limited.

Besides the utilization of various feature extraction techniques, it is possible to compare graphs directly using graph theoretic distances such as (*maximum*) *common subgraph* derived distances [12–14] or the *graph edit distance* [15]. The computation of the previously mentioned graph theoretic distances is NP-hard and as shown in [11] their application results in datasets with a high intrinsic dimensionality as well. High quality clustering methods for arbitrary (metric) and high dimensional datasets furthermore require a superlinear number of exact distance computations. These factors render graph theoretic distance measures in combination with generic clustering algorithms infeasible for large-scale datasets.

Subspace and projected clustering methods tackle high dimensional datasets by identifying subspaces in which well separated clusters exist. However, generic subspace algorithms come with a high runtime burden and are often limited to an euclidean vector space.

Our *structural projection* clustering algorithm approaches the dimensionality problems by explicitly selecting cluster representatives in form of common subgraphs. Consider a

¹<https://pubchem.ncbi.nlm.nih.gov/>

²<https://www.ebi.ac.uk/chembl/>

³<http://cdb.ics.uci.edu/>

feature mapping to binary feature vectors that contain one feature for each subgraph that is found in the complete dataset. For each graph, its feature vector has binary entries encoding the presence of the associated substructure graph. Selecting a common subgraph S as cluster representative is another way of selecting a subspace in these feature vectors consisting of all features associated with subgraphs of S .

Our main contributions in this paper are: We present a novel structural projection clustering algorithm for datasets of small labeled graphs which scales linearly with the dataset size. A set of representatives provides an intuitive description of each cluster, supports the clustering process, and helps to interpret the clustering results. Up to our knowledge, this is the first approach actively selecting representative sets for each cluster based on a new ranking function. The candidates for the representatives are constructed using frequent subgraph sampling. In order to speed up the computation, we suggest a new error bounded sampling strategy for support counting in the context of frequent subgraph mining. Our experimental evaluation shows that our new approach outperforms competitors in its runtime and quality.

The paper is structured as follows: Section II provides an overview of related clustering algorithms. Basic definitions are given in Section III. Section IV presents the main algorithm and a runtime analysis. Our experimental evaluation in which we compare our new algorithm to SCAP [16], PROCLUS [17] and Kernel K-Means [18] is presented in Section V.

II. RELATED WORK

Several clustering algorithms for graph and molecule data have been proposed in the last years. Tsuda and Kudo [10] presented an EM algorithm using a binomial mixture model over very high dimensional binary vectors, indicating the presence of all frequent substructures. Two years later, Tsuda and Kurihara [19] presented a similar graph clustering algorithm using a Dirichlet Process mixture model and pruning the set of frequent substructures to achieve smaller feature vectors. A K-Median like graph clustering algorithm has been presented by Ferrer et al. [20], that maps each graph into the euclidean space utilizing the edit distance to some pivot elements. A median graph is then selected based on the distance to the euclidean median. Furthermore, a parallel greedy overlapping clustering algorithm has been presented by Seeland et al. [21]. It adds a graph to a cluster whenever a common substructure of a user-defined minimum size exists. However, none of the previously mentioned algorithms are suitable for large datasets as a result of their high computational complexity. XProj [22] uses a projection-based approach by selecting all (enumerated) frequent substructures of a fixed size as cluster representatives. The approach scales well with the dataset size but is limited to trees. A generalization to graphs would result in a huge performance degradation. Furthermore, there exist some hybrid approaches, that pre-cluster the dataset by using a vector-based representation and refine the results using structural clustering algorithms. The most relevant with respect

to large-scale datasets is the SCAP algorithm proposed by Seeland et al. [16].

Of course many subspace algorithms are also applicable after using a feature extraction method. Giving a comprehensive overview on subspace techniques is out of the scope of this article. However, there are two algorithms that are of special interest for this work: In [23] it is shown that frequent pattern mining can be used for feature selection in vector space. This relates to XProj and our approach in the way, that the selection of a graph representative—with the help of frequent substructure mining—is another way of selecting a subspace in the feature space of substructures. In the later evaluation we will compare ourself to the PROCLUS [17] algorithm. It is a fast projected clustering algorithm with noise detection, that selects features by minimizing variance. The algorithm has been studied intensively and performed well in various subspace clustering comparisons [24, 25].

III. PRELIMINARIES

An *undirected labeled graph* $G = (V, E, l)$ consists of a finite set of *vertices* $V(G) = V$, a finite set of *edges* $E(G) = E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$ and a labeling function $l : V \uplus E \rightarrow L$, where L is a finite set of *labels*. $|G|$ is used as a short term for $|V(G)| + |E(G)|$. A *path* of length n is a sequence of vertices (v_0, \dots, v_n) such that $\{v_i, v_{i+1}\} \in E$ and $v_i \neq v_j$ for $i \neq j$. Let G and H be two undirected labeled graphs. A (*label preserving*) *subgraph isomorphism* from G to H is an injection $\psi : V(G) \rightarrow V(H)$, where $\forall v \in V(G) : l(v) = l(\psi(v))$ and $\forall u, v \in V(G) : \{u, v\} \in E(G) \Rightarrow \{\psi(u), \psi(v)\} \in E(H) \wedge l(\{u, v\}) = l(\{\psi(u), \psi(v)\})$. Iff there exists a subgraph isomorphism from G to H we say G is *supported* by H , G is a *subgraph* of H , H is a *supergraph* of G or write $G \subseteq H$. If there exists a subgraph isomorphism from G to H and from H to G , the two graphs are isomorphic. A *common subgraph* of G and H is a graph S , that is subgraph isomorphic to G and H . Furthermore, the *support* $\text{sup}(G, \mathcal{G})$ of a graph G over a set of graphs \mathcal{G} is the fraction of graphs in \mathcal{G} , that support G . G is said to be *frequent*, iff its support is larger or equal than a *minimum support threshold* minSup . A frequent subgraph G is *maximal*, iff there exists no frequent supergraph of G . For a set of graphs \mathcal{G} , we write $\mathcal{F}(\mathcal{G})$ for the set of all frequent subgraphs and $\mathcal{M}(\mathcal{G})$ for the set of all maximal frequent subgraphs. A *clustering* of a graph dataset \mathcal{X} is a partition $\mathcal{C} = \{C_1, \dots, C_n\}$ of \mathcal{X} . Each *cluster* $C \in \mathcal{C}$ consists of a set of graphs and is linked to a *set of cluster representatives* $\mathcal{R}(C) = \{R_1, \dots, R_k\}$ which are itself undirected labeled graphs. Please note, that we consider each graph in our dataset to be a distinct object. As a result, it is possible to have isomorphic graphs in a single set.

IV. THE *StruClus* ALGORITHM

A high level description of the *StruClus* algorithm is given in Algorithm 1. Initially, it partitions the dataset using a lightweight pre-clustering algorithm. Afterwards, the clustering is refined using an optimization loop similar to the K-Means algorithm. In order to fit the number of clusters to the

dataset structure (i.e., to achieve a good cluster separation and homogeneity, see Sections IV-B and IV-D), we use a cluster splitting and merging strategy in each iteration.

Algorithm 1 *StruClus* Algorithm

```

1: apply pre-clustering {Section IV-E}
2: while not convergent {Section IV-F} do
3:   split clusters {Section IV-D}
4:   merge clusters {Section IV-D}
5:   update representatives {Section IV-B}
6:   assign graphs to closest cluster {Section IV-C}
7: end while

```

An important ingredient of our algorithm is the set of representatives $\mathcal{R}(C)$ for each cluster $C \in \mathcal{C}$. Representatives serve as an intuitive description of the cluster and define the substructures over which intra cluster similarity is measured. The set $\mathcal{R}(C)$ is chosen such that for every graph $G \in C$ there exists at least one representative $R \in \mathcal{R}(C)$ which is subgraph isomorphic (i.e., supported by G). With the exception of a single *noise* cluster the following invariant holds after each iteration:

$$\forall C \in \mathcal{C} : \forall G \in C : \exists R \in \mathcal{R}(C) : R \subseteq G. \quad (1)$$

The representative set $\mathcal{R}(C)$ of a cluster $C \in \mathcal{C}$ is constructed using maximal frequent subgraphs of C (see Section IV-A). Having a representative set instead of a single representative has the advantage, that graphs composed of multiple common substructures can be represented. In order to be meaningful and human interpretable, the cardinality of $\mathcal{R}(C)$ is limited by a user defined value \mathcal{R}_{\max} . Figure 1 shows two example clusters and their representatives of a real world molecular dataset generated with *StruClus*.

A. Stochastic Representative Mining

We construct the representative set $\mathcal{R}(C)$ of a cluster $C \in \mathcal{C}$ using maximal frequent subgraphs of C . Since the set $\mathcal{M}(C)$ may have exponential size wrt the maximal graph size in C , we restrict ourselves to a subset of candidate representatives $\mathcal{S}(C) \subseteq \mathcal{M}(C)$ using a randomized maximal frequent connected subgraph sampling technique from ORIGAMI [26] combined with a new stochastic sampling strategy for support counting. In a second step, the final representative set $\mathcal{R}(C) \subseteq \mathcal{S}(C)$ is selected using a ranking function (see Section IV-B1).

ORIGAMI constructs a maximal frequent connected subgraph $S \in \mathcal{S}(\mathcal{G})$ over a set of graphs \mathcal{G} by extending a random frequent vertex with frequent paths of length one leading to a graph S' . In a first step, all frequent vertices $N(\mathcal{G})$ and all frequent paths of length one $P(\mathcal{G})$ are enumerated with a single scan of \mathcal{G} . Then, for each extension, a random vertex of S' is chosen and a random, label preserving path $p \in P(\mathcal{G})$ is connected to it in a forward (creating a new vertex) or backward (connecting two existing vertices) fashion. After each extension, the support $\text{sup}(S', \mathcal{G})$ is evaluated by solving a subgraph isomorphism test for all graphs in \mathcal{G} . If $\text{sup}(S', \mathcal{G}) \geq \text{minSup}$, the extension is permanently added to

S' or otherwise removed. If no further extension is possible without violating the minimum support threshold, a maximal frequent subgraph S has been found. This process is justified by the *monotonicity property* of subgraphs \mathcal{G}_\subseteq of graphs in \mathcal{G} :

$$\forall G, H \in \mathcal{G}_\subseteq : G \subseteq H \Rightarrow \text{sup}(G, \mathcal{G}) \geq \text{sup}(H, \mathcal{G}) \quad (2)$$

While ORIGAMI greatly improves performance in comparison with enumeration algorithms, the $|\mathcal{G}|$ subgraph isomorphism tests for each extension remain a major performance bottleneck for *StruClus*. For this reason, we have added a stochastic sampling strategy for support counting.

Initially, we draw a random sample $\mathcal{H} \subseteq \mathcal{G}$. Then $\hat{\theta} = \text{sup}(S', \mathcal{H})$ is an estimator for the parameter θ of a binomial distribution $B(\cdot, \theta)$, where $\theta = \text{sup}(S', \mathcal{G})$ is the true probability of the underlying Bernoulli distribution. We are interested if the true value of $\text{sup}(S', \mathcal{G})$ is smaller than the minimum support threshold. Without loss of generality, let us focus on the case $\hat{\theta} < \text{minSup}$ in the following. We can take advantage of a binomial test under the null hypothesis, that $\theta \geq \text{minSup}$ and thereby determine the probability of an error, if we assume $\text{sup}(S', \mathcal{G}) < \text{minSup}$. With a predefined significance level α we can decide if the sample gives us enough confidence to justify our assumption. If we cannot discard our null hypothesis, we continue by doubling the sample size $|\mathcal{H}|$ and repeat the process. In the extreme case we will therefore calculate the exact value of $\text{sup}(S', \mathcal{G})$.

The statistical test is repeated for each extension and each sample size doubling. As a consequence, a multiple hypothesis testing correction is necessary to bound the real error for S to be a maximal frequent substructure of \mathcal{G} .

Proposition IV-A.1. *Let \mathcal{G} be a set of undirected labeled graphs, $|\mathcal{H}_{\min}|$ the minimal sample size, $P(\mathcal{G})$ the set of all frequent paths of length one, minSup a minimum support threshold, and V_{\max} the $(1 - \text{minSup})$ -quantile of the sorted (increasing order) graph sizes of each graph in \mathcal{G} . Then the maximal number of binomial tests to construct a maximal frequent substructure over \mathcal{G} is bounded by:*

$$\left\lceil \log \frac{|\mathcal{G}|}{|\mathcal{H}_{\min}|} \right\rceil (V_{\max}^2 + V_{\max}) |P(\mathcal{G})|$$

Proof. The sample size is doubled $\left\lceil \log \frac{|\mathcal{G}|}{|\mathcal{H}_{\min}|} \right\rceil$ times if the test never reaches the desired significance level. The size of some $S \in \mathcal{M}(\mathcal{G})$ is bounded by the size of each supporting graph. In the worst case S is supported by the $(|\mathcal{G}| \text{minSup})$ -largest graphs of \mathcal{G} . The graph size of the smallest supporting graph is then equal to the $(1 - \text{minSup})$ -quantile of the sorted graph sizes in increasing order. The number of backward extensions is bounded by the number of vertex pairs times the number of applicable extensions $p \in P(\mathcal{G})$. Additionally, we need to check $|P(\mathcal{G})|$ forward extensions for each vertex to conclude that S is maximal. \square

Finally, with the value of Proposition IV-A.1 we are able to apply a Bonferroni correction to our significance level. We can afford a relative high error, because the selection of the

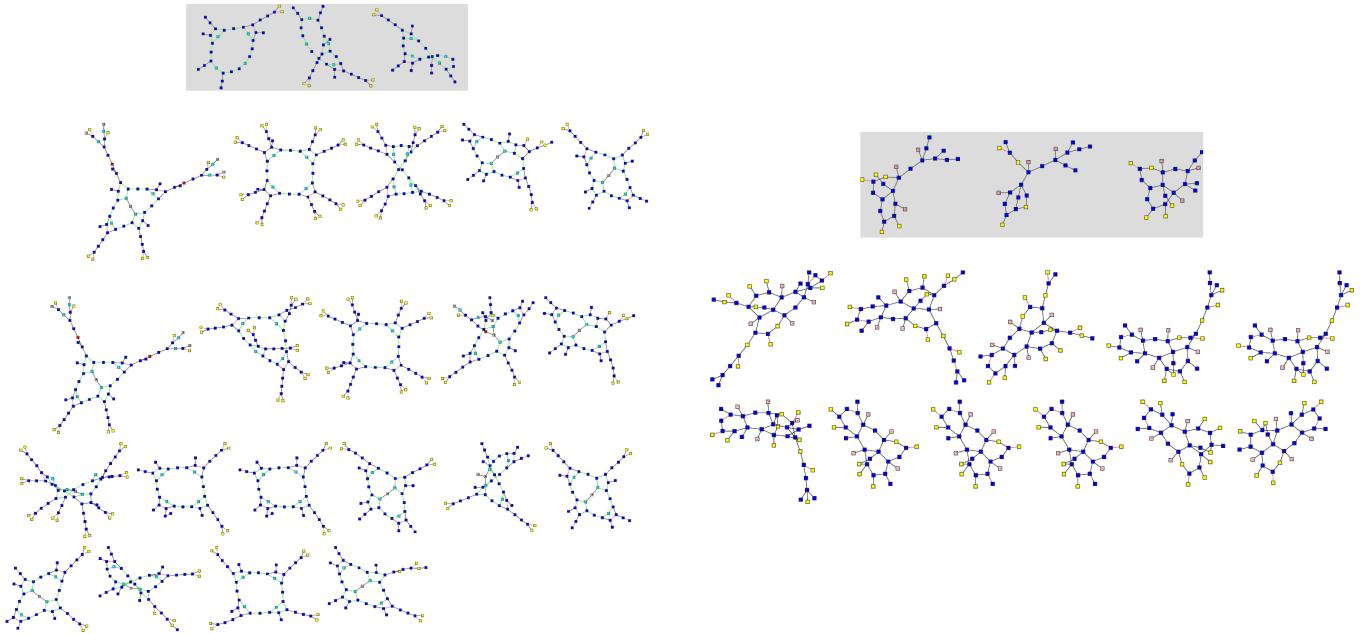


Fig. 1. Two real world clusters generated with *StruClus*. The grey boxes show the cluster representatives.

final representatives will filter out bad candidates. However, the used significance level has an influence on the runtime. On the one hand a high error leads to many bad candidates and we need to increase the number of candidates to mine. On the other hand a low error will lead to larger sample sizes to reject the null hypothesis. A maximal error of 50% has turned out to be a good choice during our experimental evaluation.

B. Update Representatives

1) Representative Selection: In its role as a cluster description, a good representative $R \in \mathcal{R}(C)$ explains a large portion of its cluster. Accordingly, it should (a) be supported by a large fraction of C and (b) cover a large fraction of vertices and edges of each graph $G \in C$ supporting R . We define the coverage of a graph G by a representative R as $\text{cov}(R, G) := \frac{|R|}{|G|}$. The two criteria are closely related to the cluster homogeneity. A uniform cluster, that is, a cluster that contains only isomorphic graphs, can achieve optimal values for both criteria. Vice versa, the monotonicity property (2) implies non-optimal values for inhomogeneous clusters for at least one of the two criteria. As homogeneous clusters are desired, we use a product of the two criteria for our ranking function.

In order to discriminate clusters from each other, a cluster representative should be cluster specific as well. Thus, its support in the rest of the dataset should be low. For this reason, we use the following ranking function for a dataset \mathcal{X} , cluster C and representative $R \in \mathcal{R}(C)$:

$$C_R := \{G \in C \mid R \subseteq G\}$$

$$\text{rank}(R) := \frac{|C_R| |R|}{\sum_{G \in C_R} |G|} (\text{sup}(R, C) - \text{sup}(R, \mathcal{X})) \quad (3)$$

Finally, we select the \mathcal{R}_{\max} highest ranked sampled subgraphs from $\mathcal{S}(C)$ as cluster representatives $\mathcal{R}(C)$.

2) Balancing Cluster Homogeneity: Besides the representative selection, the choice of the minimum support threshold for representative mining has an influence on the cluster homogeneity. The fraction of unsupported graphs for a cluster C after updating the cluster representatives is bounded by $(1 - \text{minSup})$. The bound clearly relates to criteria (a) from the representative ranking (see Section IV-B1). However, unsupported graphs are removed from the cluster and will be assigned to a different cluster at the end of the current iteration (see Section IV-C). Due to the monotonicity property (2), this process of sorting out graphs by choosing a minimum support threshold below 1, will increase the size of the representatives and therefore our coverage value, i.e criteria (b). A decrease of the minimum support threshold will lead to an increase in the size of the representatives. Subsequently, this process will also reduce the cluster cardinality. Therefore, increasing the homogeneity to an optimal value will result in a clustering with uniform or singleton clusters, which is clearly not the desired behavior.

To get around this, we will aim towards a similar homogeneity for all clusters and choose the minimum support threshold cluster specific. However, choosing a fixed homogeneity level a priori is not an easy task, as an appropriate value depends on the dataset. Therefore we will calculate an average coverage score over all clusters and use this as a baseline adjustment. For the ease of computation, we choose a slightly simplified

coverage approximation:

$$\text{aCov}(C) = \frac{\frac{1}{|\mathcal{R}(C)|} \sum_{R \in \mathcal{R}(C)} |R|}{\frac{1}{|C|} \sum_{G \in C} |G|} \quad (4)$$

$$\text{relCov}(C, \mathcal{C}) = \frac{\text{aCov}(C)}{\frac{1}{|\mathcal{C}|} \sum_{C' \in \mathcal{C}} \text{aCov}(C')} \quad (5)$$

Finally, we can define a linear mapping from the relative coverage relCov to a cluster specific minimum support threshold with the help of two predefined tuples (ls, lr) and (hs, hr) , where $ls < hs$ and $lr < hr$. The parameter ls (hs) denotes the lowest (highest) support value and lr (hr) the relative coverage value mapped to the lowest (highest) minimum support:

$$\begin{aligned} f(C, \mathcal{C}) &:= \text{relCov}(C, \mathcal{C}) \frac{hs - ls}{hr - lr} + (ls - lr) \frac{hs - ls}{hr - lr} \\ \text{minSup}(C, \mathcal{C}) &:= \begin{cases} ls & \text{if } \text{relCov}(C, \mathcal{C}) < lr \\ hs & \text{if } \text{relCov}(C, \mathcal{C}) > hr \\ f(C, \mathcal{C}) & \text{otherwise.} \end{cases} \end{aligned} \quad (6)$$

To result in a minimum support threshold of 1 for all clusters (i.e. stopping the process of sorting out graphs if the clustering is balanced), we will set the values of the parameters hs very close or equal to 1 and $hr < 1$.

C. Cluster Assignment

Each graph G in the dataset is assigned to its *most similar cluster* in the assignment phase. As a measure for similarity, we are summing up the squared sizes of the representatives of a cluster, which are subgraph isomorphic to G . This choice of similarity is once more justified by the representative ranking criteria. We square the representative sizes, to prefer a high coverage over a high number of representatives to be subgraph isomorphic to the assigned graph.

As mentioned in Section IV-B2 it is possible that a graph $G \in C$ will no more be supported by any representative $R \in \mathcal{R}(C)$ of its cluster C after updating the representatives. In this situation we will create a single noise cluster, where all graphs (of all clusters) that are not supported by any representative are collected. As the minimum support threshold is bounded by the fixed value ls (see Section IV-B2) and the number of representatives is limited by \mathcal{R}_{\max} , it is not guaranteed that we can find an appropriate set of representatives for this most likely largely inhomogeneous noise cluster. It is therefore excluded from the invariant (1).

The problem of finding all subgraph isomorphic graphs in a graph database is also known as the *subgraph search problem* and was extensively studied in the past. We apply the fingerprint pre-filtering technique CT-Index [27], which has emerged from this research topic, to speed up the assignment phase. CT-Index enumerates trees and circles up to a specified size for a given graph and hashes the presence of these subgraphs into a binary fingerprint of fixed length. If the fingerprint of a graph G has a bit set that is unset in the fingerprint of a graph H we can conclude that no subgraph isomorphism from G to H exists, because G contains a subgraph that is not present in H . We calculate a fingerprint for each graph and

representative and only perform a subgraph isomorphism test in our assignment phase if the fingerprint comparison cannot rule out the presence of a subgraph isomorphism.

D. Cluster Splitting and Merging

Without the operation of cluster splitting, the above mentioned process of creating noise clusters would create at most one extra cluster in each iteration of our main loop. A large difference in the initial and final number of clusters therefore would lead to a slow convergence of the *StruClus* algorithm towards its final result. As mentioned before, it is also possible that no representative is found at all for the noise cluster, and therefore the process of sorting out graphs from the noise cluster to increase its homogeneity is stopped completely in the worst case. A similar situation can occur for regular clusters. For example, if a cluster is composed of uniform sets $T \in \mathcal{T}$ of graphs, we will require a minimum support threshold less than or equal to $1 - \frac{1}{\min_{T \in \mathcal{T}} \{|T|\}}$ to sort the smallest possible number of graphs out. For this reason, a cluster splitting step is necessary (see Algorithm 1). In this step, all clusters that have a relative coverage value below an a priori specified threshold relCov_{\min} will be merged into a single set of graphs and the pre-clustering algorithm is applied on them. The resulting clusters are added back to the clustering.

On the contrary to cluster splitting, which focuses on cluster homogeneity, cluster merging ensures a minimum separation between clusters. Separation can be measured on different levels. Many classical measures define separation as the minimum distance between two cluster elements. However, this type of definition is not suitable for projected clustering algorithms, because the comparison does not take the cluster specific subspace into account. As mentioned in the introduction, the cluster representatives in *StruClus* define the subspace of the cluster. Additionally, they serve as a description of the graphs inside the cluster itself. A high coverage value leads to an accurate cluster description. Thus, we will define *separation* between two clusters C and C' solely over the representatives sets $\mathcal{R}(C)$ and $\mathcal{R}(C')$. This definition is also beneficial from a runtime perspective, as separation calculation is independent of the cluster size. Without cluster merging it is possible that clusters with very similar representatives do exist. Although the pre-clustering will ensure that the initial clusters will have dissimilar representatives (see Section IV-E) it may happen that two clusters converge towards each other or that newly formed clusters are similar to an already existing one. Therefore, we will merge two clusters whenever their representatives are too similar.

To compare two single representatives R, R' we calculate the size of their maximum common subgraph (MCS) and use its relative size as similarity:

$$\text{sim}(R, R') := \frac{|\text{mcs}(R, R')|}{\max\{|R|, |R'|\}} \quad (7)$$

The maximum of the representatives sizes is chosen as denominator to support different clusters with subgraph isomorphic

representatives, which differ largely in size. Finally, we will merge two clusters C and C' if the following condition holds:

$$|\{(R, R') \in \mathcal{R}(C) \times \mathcal{R}(C') \mid \text{sim}(R, R') \geq \text{sim}_{\min}\}| \geq \text{sim}_{\text{num}} \quad (8)$$

where sim_{num} is a minimum number of representative pairs which have a similarity greater than or equal to sim_{\min} . Note that the calculated MCS between two representatives $R \in \mathcal{R}(C), R' \in \mathcal{R}(C')$ is supported by all the graphs $G \in C \cup C'$, that support either R or R' , because the subgraph isomorphism relation is transitive. The coverage for these graphs in the merged cluster is furthermore bounded by $\max\{\text{cov}(G, R), \text{cov}(G, R')\} \text{sim}(R, R')$ if we reuse the MCS as representative. For this reason, we recommend to set sim_{num} close to the number of representatives per cluster to support a large fraction of graphs in the merged cluster. The parameter sim_{\min} is furthermore an intuitive knob to adjust the granularity of the clustering.

Finally, the representatives for the merged clusters are updated. We do not use the calculated MCSs as representatives, because better representatives may exist and the decision problem “Does an MCS larger than some threshold exist?” is computationally less demanding than calculating the MCS itself.

E. Pre-Clustering

The pre-clustering serves as an initial partitioning of the dataset. A random partitioning of all graphs would be problematic as representatives may not be found for all partitions and the found representatives are most likely not cluster specific. This will result in a high number of clusters to be merged to a few inhomogeneous clusters and in a slow convergence of the *StruClus* algorithm.

To pre-cluster the dataset \mathcal{X} , we compute maximal frequent subgraphs $\mathcal{S}(\mathcal{X}) \subseteq \mathcal{M}(\mathcal{X})$, as described in Section IV-A with a fixed minimum support. These frequent subgraphs serve as representative candidates for the initial clusters. To avoid very similar representatives we will first greedily construct maximal sets of dissimilar graphs. As a measure of similarity we re-use the similarity (7) and the threshold sim_{\min} from cluster merging. In other words, we are picking all graphs G from $\mathcal{S}(\mathcal{X})$ in a random order and add G to our dissimilar set \mathcal{D} , if $\nexists H \in \mathcal{D}$ with $\text{sim}(G, H) < \text{sim}_{\min}$. This process is repeated several times and the largest set \mathcal{D}_{\max} is used to create one cluster for each $H \in \mathcal{D}$ with H as single representative. Afterwards we run a regular assignment phase as described in Section IV-C. As a result of re-using sim_{\min} we can expect, that we have well separated cluster and no cluster merging is necessary in the first iteration of the main loop (excluding the noise cluster).

F. Convergence

StruClus optimizes cluster homogeneity while maintaining a minimal cluster separation. As described in Section IV-B2, homogeneity is defined by two criteria wrt the cluster representatives. With an exception of the noise cluster, the

representative support criteria (a) can be dismissed, because not represented graphs will be sorted out of the cluster. It was further discussed, that an optimal homogeneity can be achieved for singleton clusters. However, the later introduced cluster separation constraint will limit the granularity of the clustering, because two similar representatives will be merged. Nevertheless, there might exist several clusterings with different granularity that respect the separation constraint. For this reason the objective function balances the coverage criteria (b) of the homogeneity and the granularity of the clustering. The parameter α adjusts this granularity:

$$z(\mathcal{C}) := \frac{\sum_{C \in \mathcal{C}} |C| \text{aCov}(C)}{|\mathcal{C}|^\alpha} \quad (9)$$

As a consequence of the cluster splitting and merging, the objective function will fluctuate and contain local optima. We will therefore smooth the objective function. Let \mathcal{C}_i be the clustering after the i -th iteration. Algorithm 1 will terminate after the first iteration c for which the following condition holds:

$$c \geq s + w \wedge \frac{\sum_{c-w < i \leq c} z(\mathcal{C}_i)}{\sum_{c-s-w < i \leq c-s} z(\mathcal{C}_i)} \leq 1 + \beta \quad (10)$$

where w is the averaging width and β is the minimum relative increase of the objective function in s iterations.

G. Runtime Analysis

The subgraph isomorphism problem and the maximum common subgraph problem are both NP-complete even in their decision variants [28]. *StruClus* solves these problems in order to calculate the support and to decide if clusters need to be merged. Furthermore, the size of a representative can be linear in the sizes of the supporting graphs. Thus, *StruClus* scales exponentially wrt the size of the graphs in the dataset \mathcal{X} . Nevertheless, these problems can be solved sufficiently fast for small graphs with a few hundred vertices and edges, e.g., molecular structures. We will therefore consider the graph size a constant in the following analysis and focus on the scalability wrt the dataset size. Let \mathcal{V}_{\max} be the maximal number of vertices for a graph in \mathcal{X} and \mathcal{C}_{\max} the maximal number of clusters during the clustering process.

Representative Mining: As described in Proposition IV-A.1 the number of extensions to mine a single maximal frequent subgraph from a set of graphs $\mathcal{G} \subseteq \mathcal{X}$ is bounded by $(V_{\max}^2 + V_{\max}) |\mathcal{P}(\mathcal{G})|$. The variable V_{\max} is obviously a constant. Also $|\mathcal{P}(\mathcal{G})|$ is only bounded by \mathcal{V}_{\max} and the lowest minimum support threshold ls , as a fraction of $\frac{ls}{V_{\max}^2}$ of all edges in \mathcal{G} must be isomorphic to be frequent. Thus, there exist at most $\frac{\mathcal{V}_{\max}^2}{ls}$ frequent paths of length one. Note, that the number of distinct labels does not have an influence on this theoretical bound. For each mined maximal frequent subgraph we need to calculate the support over \mathcal{G} and this involves $\mathcal{O}(|\mathcal{G}|)$ many subgraph isomorphism tests. Therefore, the runtime to mine a single maximal frequent subgraph is bounded by $\mathcal{O}(|\mathcal{G}|)$.

Representative Update: Updating the representatives involves two steps: representative mining and representative selection. Representative mining includes the calculation of the cluster specific minimum support with the help of relCov . The relCov values for each cluster can be calculated in $\mathcal{O}(\mathcal{C}_{\max})$ if we maintain a sum of the graph sizes of each cluster and update this value during cluster assignment. Since the cluster sizes sum up to $|\mathcal{X}|$ the runtime of the actual frequent subgraph mining is in $\mathcal{O}(\mathcal{X})$. To rank a representative candidate $R \in \mathcal{R}(C)$, the set C_R , the value $\text{sup}(R, C)$, and the value $\text{sup}(R, \mathcal{X})$ can be calculated by a single scan over \mathcal{X} , i.e. calculating whether R is subgraph isomorphic to each graph in \mathcal{X} . As the number of candidates per cluster is a constant, the runtime of the ranking is in $\mathcal{O}(\mathcal{C}_{\max} |\mathcal{X}|)$, which is also the overall runtime of the representative update.

Other Parts: The runtimes of the cluster assignment and the pre-clustering are in $\mathcal{O}(\mathcal{C}_{\max} |\mathcal{X}|)$. Cluster splitting is computable in $\mathcal{O}(\mathcal{C}_{\max} + |\mathcal{X}|)$, cluster merging in $\mathcal{O}(\mathcal{C}_{\max}^2 + |\mathcal{X}|)$, and converge in $\mathcal{O}(\mathcal{C}_{\max})$ time.

Overall Runtime: A single iteration of Algorithm 1 has a runtime of $\mathcal{O}(\mathcal{C}_{\max} |\mathcal{X}|)$, and hence is linear. This is justified by the observation that $\mathcal{O}(\mathcal{C}_{\max}^2) \leq \mathcal{O}(\mathcal{C}_{\max} |\mathcal{X}|)$. Let m be the number of iterations after Algorithm 1 terminates. Then, the overall runtime is in $\mathcal{O}(m \mathcal{C}_{\max} |\mathcal{X}|)$.

V. EVALUATION

In the following evaluation we compare *StruClus* to SCAP [16], PROCLUS [17] and Kernel K-Means [18] wrt their runtime and the clustering quality. Furthermore, we evaluate the influence of our sampling strategy for support counting and evaluate the parallel scaling of our implementation.

Hardware & Software: All tests were performed on a dual socket NUMA system (Intel Xeon E5-2640 v3) with 128 GiB of RAM. The applications were pinned to a single NUMA domain (i.e. 8 cores + HT / 64 GiB RAM) to eliminate random memory effects. Turbo Boost was deactivated to minimize external runtime influences. Ubuntu Linux 14.04.1 was used as operating system. The Java implementations of *StruClus*, PROCLUS and Kernel K-Means were running in an Oracle Java Hotspot VM 1.8.0_66. SCAP was compiled with GCC 4.9.3 and -O3 optimization level. *StruClus* and SCAP are shared memory parallelized.

Test Setup & Evaluation Measures: The tests were repeated 30 times if the runtime was below 2 hours and 15 times otherwise. Quality is measured by ground truth comparisons. We use the Normalized Variation of Information (NVI) [29] and Fowlkes-Mallows (FW) [30] measures for *StruClus*, PROCLUS and Kernel K-Means. While NVI and FW are established quality measures, they are not suited for overlapping clusterings as produces by SCAP. Thus, Purity [31] was used for comparison with SCAP.

Datasets: We evaluate the algorithms on synthetic datasets of different sizes and three real world datasets. The synthetic datasets have ≈ 35 vertices and ≈ 51 edges on average with 10 vertex and 3 edge labels (weights drawn from an exponential distribution). They contain 100 clusters

and 5% random noise graphs. For each cluster 3 seed patterns were randomly generated with a Poisson distributed number of vertices (mean 10) and an edge probability of 10%. Additionally, we created a common seed pool with 100 (noise) seeds. The cluster specific graphs were then generated by connecting the cluster specific seeds with 0 to 2 common noise seeds. The first real world dataset (AnchorQuery) is a molecular de-novo database. Each molecule is the result of a chemical reaction of multiple purchasable building blocks. We have used 11 reaction types, i.e., class labels, from AnchorQuery⁴. The second real world dataset (Heterocyclic) is similar to the first one, but contains heterocyclic compounds and 39 distinct reaction types. The third real world dataset (ChemDB) contains 5 million molecules from ChemDB [32]. ChemDB is a collection of purchasable molecules from 150 chemical vendors. The dataset has no ground truth and will mainly serve as proof that we are able to cluster large real world datasets. Table I shows additional statistics about these real world datasets.

To cluster the datasets with PROCLUS the graph data was transformed to vectors by counting distinct subgraphs of size 3, resulting in 7 000 to 10 000 features on the synthetic datasets. The application to the AnchorQuery and Heterocyclic datasets results in much lower feature counts of 274 and 133. The same vectors were used as features space representation for Kernel K-Means. Additionally, we have evaluated various other graph kernels with explicit feature mapping for PROCLUS and Kernel K-Means, such as the Weisfeiler-Lehman shortest path, Weisfeiler-Lehman subtree and fixed length random walk kernels [33]. However, we observed feature vectors with even higher dimensionality and clustering results with lower quality. For this reason, the results of these graph kernels are omitted here.

Algorithm Configuration: The parameters of *StruClus* were set as follows: maximal error for support counting: 50%; number of representative candidates: 25; minimum support threshold calculated with: $lr = 0$; $ls = 0.4$; $hr = 0.9$; $hs = 0.99$; splitting threshold: $aCov \leq 0.6$; minimum separation $sim_{\min} = 0.3$; $sim_{\max} = 3$; convergence determined with: $\alpha = 1$, $\beta = 0.01$, $w = 3$, $s = 3$. SCAP has two parameters: (a) the minimum size for a common substructure, that must be present in a single cluster and (b) a parameter that influences the granularity of the fingerprint-based pre-partitioning. (a) was set to 80% of the mean seed size for the synthetic dataset, otherwise to 8. (b) was set to the highest number that results in a reasonable clustering quality, as the clustering process is faster with fine grained pre-partitioning, but cannot find any clusters over the pre-partition boundary. For the synthetic dataset this was 0.2. PROCLUS has two parameters as well: (a) the number of clusters and (b) the average dimensionality of the cluster subspaces. (a) was set to the number of clusters in the ground truth. The ChemDB dataset, which has no ground truth, was too large for PROCLUS. (b) was set to 20, for which we got best quality results. The number of clusters

⁴<http://anchorquery.cs.pitt.edu/reactions/>

TABLE I
REAL WORLD DATASET STATISTICS. CUMULATED VALUES ARE GIVEN AS TRIPLE MIN / MAX / AVERAGE.

dataset	size	classes	# vertices	# edges	# vertex labels	# edge labels
AnchorQuery	65 700	11	11 / 90 / 79.19	11 / 99 / 86.02	6	5
Heterocyclic	10 000	39	9 / 69 / 42.99	10 / 79 / 47.35	25	5
ChemDB	5 000 000	–	1 / 684 / 50.74	0 / 745 / 53.20	86	5

is the only parameter of Kernel K-Means and was set in exact the same manner as for PROCLUS. Please note, that the a priori selection of this optimal value gives PROCLUS and Kernel K-Means an advantage over their competitors during the evaluation.

Results: As shown in Table II, *StruClus* outperforms all competitors in terms of quality on the synthetic datasets. The differences of all corresponding mean quality values are always larger than small multiples of the standard deviations. For small synthetic datasets, SCAP was the fastest algorithm. This changes for larger data set sizes. *StruClus* outperforms SCAP by a factor of ≈ 13 at size 500 000. Furthermore, we were unable to cluster the largest dataset with SCAP in less than 2 days. PROCLUS and Kernel K-Means were the slowest algorithms with a huge gap to their next competitor. The sublinear growth of *StruClus*'s runtime for the smaller datasets sizes is caused by our sampling strategy for support counting. Running *StruClus* with exact support counting yields a linear growth in runtime. It is important to mention that the clustering quality is not significantly influenced by the support counting strategy.

Our implementation of *StruClus* scales well with the number of cores as shown in Fig. 2. With 8 cores we get a speedup of 7.15. Including the Hyper-Threading cores, we get a speedup of 9.11.

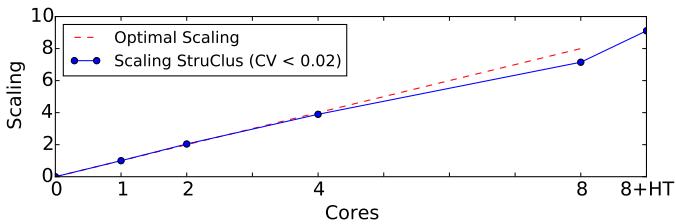


Fig. 2. Parallel scaling: synthetic dataset of size 10 000

The evaluation results wrt the real world datasets are summarized in Table III. We were unable to cluster the AnchorQuery and the ChemDB datasets with PROCLUS, Kernel K-Means and SCAP (even for high values of parameter (b)) in less than 2 days. For the chemical reaction datasets, *StruClus* also needs more time compared to a synthetic dataset of similar size. This runtime increase can be partially explained by the larger graph sizes. However, other hidden parameters of the datasets, such as the size of maximal common substructures have an influence on the runtime as well. *StruClus* had some runtime outliers on the Heterocyclic dataset. They were

caused by small temporary clusters with large (> 60 vertices) representatives. The high runtime of the SCAP algorithm on the AnchorQuery dataset is a bit surprising, as the common substructures processed by SCAP are limited in their size (maximum 8 vertices). We consider a larger frequent pattern search space to be the reason for this runtime increase. Kernel K-Means was surprisingly slow on the Heterocyclic dataset and took more than 24 hours for a single run. We have therefore created a random subset with a size of 5000 graphs for it.

StruClus always outperforms the competitors wrt the quality scores. For the AnchorQuery dataset *StruClus* created 26 clusters on average. The high score for the Purity measure shows, that *StruClus* splitted some of the real clusters, but keeps a well inter cluster separation. ChemDB was clustered by *StruClus* in ≈ 19 hours with ≈ 117 clusters. As a consequence of the high runtime of the ChemDB measurement and the lack of competitors, we repeated the test only 3 times. The aCov value for the final clustering was 0.49 on average. This highlights the ability of *StruClus* to cluster large-scale real-world datasets.

VI. CONCLUSION

In this paper we have presented a new structural clustering algorithm for large scale datasets of small labeled graphs. With the help of explicitly selected cluster representatives, we were able to achieve a linear runtime in the worst case wrt the dataset size. A novel support counting sampling strategy with multiple hypothesis testing correction was able to accelerate the algorithm significantly without lowering the clustering quality. We have furthermore shown, that cluster homogeneity can be balanced with a dynamic minimum support strategy for representative mining. A cluster merging and splitting step was introduced to achieve a well separated clustering even in the high dimensional pattern space. Our experimental evaluation has shown that our new approach outperforms the competitors wrt clustering quality, while attaining significantly lower runtimes for large scale datasets. Although we have shown that *StruClus* greatly improves the clustering performance compared to its competitors, de-novo datasets with several billion of molecules are still outside the scope of this work. For this reason, we consider the development of a distributed variant of the algorithm to be future work. Another consideration to further improve the quality of the algorithm is to integrate a discriminative frequent subgraph miner for representative mining. The integration of the discriminative property into the mining process has the advantage, that higher

TABLE II

RESULTS FOR THE SYNTHETIC DATASETS. RUNTIMES ARE GIVEN IN HOURS. BEST RESULTS ARE MARKED BOLD. CV IS THE COEFFICIENT OF VARIATION. THE CV VALUE IS GIVEN AS THE MAXIMUM FOR EACH COLUMN. ALL OTHER VALUES ARE AVERAGED. QUALITY MEASURES ARE: NORMALIZED VARIATION OF INFORMATION (NVI), FOWLKES-MALLOWS INDEX (FW), AND PURITY.

Size	StruClus				StruClus (Exact Support)				SCAP				PROCLUS				Kernel K-Means			
	Runtime	NVI	FM	Purity	Runtime	NVI	FM	Purity	Runtime	Purity	Runtime	NVI	FM	Runtime	NVI	FM	Runtime	NVI	FM	
CV < 0.08	< 0.03	< 0.07	< 0.03	< 0.07	< 0.02	< 0.02	< 0.04	< 0.06	< 0.02	< 0.32	< 0.11	< 0.22	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	
1 000	0.05	0.90	0.75	0.90	0.05	0.90	0.77	7.5 × 10⁻⁴	0.83	0.13	0.58	0.26	0.02	0.77	0.54	—	—	—	—	
5 000	—	—	—	—	—	—	—	—	—	4.99	0.50	0.24	2.87	0.84	0.67	—	—	—	—	
10 000	0.19	0.95	0.87	0.99	0.59	0.93	0.85	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	2.69	0.92	0.85	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	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	

TABLE III

RESULTS FOR THE REAL WORLD DATASETS. RUNTIMES ARE GIVEN IN HOURS. BEST RESULTS ARE MARKED BOLD. CV IS THE COEFFICIENT OF VARIATION. THE CV VALUE IS GIVEN AS THE MAXIMUM FOR EACH COLUMN. ALL OTHER VALUES ARE AVERAGED. THE CHEMDB MEASUREMENT WAS REPEATED ONLY 3 TIMES. WE HAVE NOT CALCULATED THE —OTHERWISE MEANINGLESS— CV FOR IT. RESULTS FOR KERNEL K-MEANS ARE ARE GIVEN FOR A RANDOM SUBSET OF THE HETEROCYLCIC DATASET. QUALITY MEASURES ARE: NORMALIZED VARIATION OF INFORMATION (NVI), FOWLKES-MALLOWS INDEX (FW), AND PURITY.

dataset	StruClus				SCAP				PROCLUS				Kernel K-Means			
	Runtime	NVI	FM	Purity	Runtime	Purity	Runtime	NVI	FM	Runtime	NVI	FM	Runtime	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	< 0.01	< 0.01	< 0.01	< 0.01	
AnchorQuery	2.47	0.44	0.63	0.89	—	—	—	—	—	—	—	—	—	—	—	
Heterocyclic	1.07	0.46	0.53	0.66	0.01	0.58	0.01	0.29	0.29	<i>3.03</i> (Subset)	0.27	0.29	—	—	—	
ChemDB	≈ 19	—	—	—	—	—	—	—	—	—	—	—	—	—	—	

quality representative candidates are mined. This will result in a lower number of necessary candidate patterns, which has a positive effect on the runtime. Furthermore, it allows to mine highly discriminant, non-maximal subgraphs. However, it is non-trivial to extend the support counting sampling strategy to such miners. Additionally, discriminative scores are usually non-monotonic on the subgraph lattice [34, 35], which imposes another runtime burden.

ACKNOWLEDGMENT

This work was supported by the German Research Foundation (DFG), priority programme *Algorithms for Big Data (SPP 1736)*. We would like to thank Nils Kriege for providing a fast subgraph isomorphism and Madeleine Seeland, Andreas Karwath, and Stefan Kramer for providing their SCAP implementation.

REFERENCES

- [1] C. Kalinski, M. Umkehrer, L. Weber, J. Kolb, C. Burdack, and G. Ross, “On the industrial applications of mcrs: molecular diversity in drug discovery and generic drug synthesis,” English, *Molecular Diversity*, vol. 14, no. 3, pp. 513–522, 2010.
- [2] E. Chávez and G. Navarro, “A probabilistic spell for the curse of dimensionality,” in *Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Washington, DC, USA, January 5-6, 2001, Revised Papers*, A. L. Buchsbaum and J. Snoeyink, Eds., ser. Lecture Notes in Computer Science, vol. 2153, Springer, 2001, pp. 147–160.
- [3] A. Gupta, R. Krauthgamer, and J. R. Lee, “Bounded geometries, fractals, and low-distortion embeddings,” in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, IEEE, 2003, pp. 534–543.
- [4] 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*, ser. ICDT ’99, London, UK, UK: Springer-Verlag, 1999, pp. 217–235.
- [5] M. Ackerman and S. Ben-David, “Clusterability: A theoretical study,” in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, D. A. V. Dyk and M. Welling, Eds., ser. JMLR Proceedings, vol. 5, JMLR.org, 2009, pp. 1–8.
- [6] N. Wale, I. A. Watson, and G. Karypis, “Comparison of descriptor spaces for chemical compound retrieval and classification,” *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 347–375, 2008.
- [7] 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, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, D. A. V. Dyk and M. Welling, Eds., ser. JMLR Proceedings, vol. 5, JMLR.org, 2009, pp. 488–495.
- [8] S. V. N. Vishwanathan, N. N. Schraudolph, R. I. Kondor, and K. M. Borgwardt, “Graph kernels,” *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.
- [9] P. Foggia, G. Percannella, and M. Vento, “Graph matching and learning in pattern recognition in the last 10 years,” *IJPRAI*, vol. 28, no. 1, 2014.
- [10] 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.
- [11] N. Kriege, P. Mutzel, and T. Schäfer, “Practical sahn clustering for very large data sets and expensive distance metrics,” *Journal of Graph Algorithms and Applications*, vol. 18, no. 4, pp. 577–602, 2014.
- [12] H. Bunke and K. Shearer, “A graph distance metric based on the maximal common subgraph,” *Pattern Recognition Letters*, vol. 19, no. 3-4, pp. 255–259, 1998.
- [13] W. D. Wallis, P. Shoubridge, M. Kraetzl, and D. Ray, “Graph distances using graph union,” *Pattern Recognition Letters*, vol. 22, no. 6/7, pp. 701–704, 2001.
- [14] M.-L. Fernández and G. Valiente, “A graph distance metric combining maximum common subgraph and minimum common supergraph,” *Pattern Recognition Letters*, vol. 22, no. 6–7, pp. 753 – 758, 2001.

- [15] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.
- [16] M. Seeland, A. Karwath, and S. Kramer, "Structural clustering of millions of molecular graphs," in *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, Y. Cho, S. Y. Shin, S. Kim, C. Hung, and J. Hong, Eds., ACM, 2014, pp. 121–128.
- [17] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, "Fast algorithms for projected clustering," in *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, A. Delis, C. Faloutsos, and S. Ghahrezaee, Eds., ACM Press, 1999, pp. 61–72.
- [18] M. A. Girolami, "Mercer kernel-based clustering in feature space," *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 780–784, 2002.
- [19] K. Tsuda and K. Kurihara, "Graph mining with variational dirichlet process mixture models," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, SIAM, 2008, pp. 432–442.
- [20] 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, 13th International Conference, CAIP 2009, Münster, Germany, September 2-4, 2009. Proceedings*, X. Jiang and N. Petkov, Eds., ser. Lecture Notes in Computer Science, vol. 5702, Springer, 2009, pp. 342–350.
- [21] M. Seeland, S. A. Berger, A. Stamatakis, and S. Kramer, "Parallel structural graph clustering," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III*, 2011, pp. 256–272.
- [22] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, P. Berkhin, R. Caruana, and X. Wu, Eds., ACM, 2007, pp. 46–55.
- [23] M. L. Yiu and N. Mamoulis, "Frequent-pattern based iterative projected clustering," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, 2003, pp. 689–692.
- [24] 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 (VLDB 2009), Lyon, France, PVLDB Journal, Vol. 2, No. 1*, VLDB Endowment, 2009, pp. 1270–1281.
- [25] A. Patrikainen and M. Meila, "Comparing subspace clusterings," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 7, pp. 902–916, 2006.
- [26] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki, "ORIGAMI: mining representative orthogonal graph patterns," 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. 153–162.
- [27] K. Klein, N. Kriege, and P. Mutzel, "CT-Index: fingerprint-based graph indexing combining cycles and trees," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 2011, pp. 1115–1126.
- [28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [29] M. Meilă, "Comparing clusterings—an information based distance," *Journal of multivariate analysis*, vol. 98, no. 5, pp. 873–895, 2007.
- [30] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, pp. 553–569, 1983.
- [31] X. Hui and L. Zhongmon, "Clustering validation measures," in *Data clustering: algorithms and applications*, C. C. Aggarwal and C. K. Reddy, Eds., New York: CRC Press, 2013, ch. 23, pp. 571–605.
- [32] J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi, "Chemdb update—full-text search and virtual chemical space," *Bioinformatics*, vol. 23, no. 17, pp. 2348–2351, 2007.
- [33] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.
- [34] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed., ACM, 2008, pp. 433–444.
- [35] 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," *Statistical Analysis and Data Mining*, vol. 3, no. 5, pp. 302–318, 2010.
- [36] D. A. V. Dyk and M. Welling, Eds., *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, vol. 5, ser. JMLR Proceedings, JMLR.org, 2009.