Parallel Structural Graph Clustering

Madeleine Seeland[1], Simon A. Berger[2], Alexandros Stamatakis[2], and Stefan Kramer[1]

[1] Technische Universität München,
Institut für Informatik/I12,
85748 Garching b. München, Germany
{madeleine.seeland, stefan.kramer}@in.tum.de
[2] Heidelberg Institute for Theoretical Studies,
69118 Heidelberg, Germany
bergers@in.tum.de, Alexandros.Stamatakis@h-its.org

Abstract. We address the problem of clustering large graph databases according to scaffolds (i.e., large structural overlaps) that are shared between cluster members. In previous work, an online algorithm was proposed for this task that produces overlapping (non-disjoint) and non-exhaustive clusterings. In this paper, we parallelize this algorithm to take advantage of high-performance parallel hardware and further improve the algorithm in three ways: a refined cluster membership test based on a set abstraction of graphs, sorting graphs according to size, to avoid cluster membership tests in the first place, and the definition of a cluster representative once the cluster scaffold is unique, to avoid cluster comparisons with all cluster members. In experiments on a large database of chemical structures, we show that running times can be reduced by a large factor for one parameter setting used in previous work. For harder parameter settings, it was possible to obtain results within reasonable time for 300,000 structures, compared to 10,000 structures in previous work. This shows that structural, scaffold-based clustering of smaller libraries for virtual screening is already feasible.

1 Introduction

Structured databases in various application areas, such as chemistry, provide a rich source of data that, in many cases, contain groups of structurally similar and dissimilar objects. To detect such groups in databases of graphs, graph clustering methods have been extensively investigated over the past few years. Basically, there exist two complementary approaches to graph clustering [7]. The simpler and more established one is to calculate a vectorial representation of the graphs and use standard similarity or distance measures in combination with standard clustering algorithms. The feature vector can be composed of properties of the graph and / or of subgraph occurrences [5, 12]. Methods from this category have been found to be highly efficient, but imply a loss of information with respect to the graph topology. Moreover, a problem with vectorial graph representations is that it is unclear what a good or even optimal vectorial representation is.

The second approach to graph clustering is to use the structure of the graphs directly [1, 6, 8–10], e.g., by computing the maximum common subgraph (MCS) between a set of graphs. These techniques have the desirable property that the calculated similarity measure is intuitive and can be visualized easily. However, the efficiency and scalability of these methods is still an open problem.

In this paper, we address the problem of clustering large graph databases according to scaffolds, i.e., large structural overlaps that are shared among all cluster members. More precisely, we require the cluster members to share at least one common subgraph that covers a specific fraction of the graphs in the cluster. An important challenge in this endeavor is the scalability to large graph data sets (of the order of 10^5 to 10^6 graphs). Graph databases such as the ones representing chemical compounds routinely encompass several hundred thousand graphs; thus, clustering methods that are able to explore and structure the vast graph space are highly desirable. Clustering large databases has emerged as a challenging research area with a large variety of applications, such as in the field of virtual screening, where the task is to analyze large databases of chemical compounds to identify possible drug candidates. By applying clustering techniques it is, for example, possible to prestructure the chemical space, e.g., for local modeling to capture the multi-mechanistic nature of many endpoints, the rediscovery of analog series or visualization. The majority of structural (i.e., scaffold-based) graph-based clustering algorithms, involving e.g., the computation of the MCS, is hardly suitable for such data sets. Graph data sets covered in related papers typically contain only several hundred graphs [1, 4, 6], and hardly any effort has been spent on characterizing the performance of the clustering algorithms. Only recently, a scaffold-based structural graph clustering algorithm [8] has been shown to handle graph data sets of at least 10,000 graphs. As this algorithm is still limited in performance, we present a parallel, scalable version of this algorithm in this paper. The algorithm, called PSCG (parallel structural clustering of graphs) in the following, is based on the idea of task partitioning in conjunction with refined cluster membership tests. More precisely, we used a set abstraction of graphs and a size-based clustering criterion to reduce the number of expensive subgraph search computations, which are not affordable exhaustively on large databases. Moreover, to avoid cluster comparisons with all cluster members, which grow computationally more expensive with increasing cluster size, we define a cluster representative for each cluster once a unique cluster scaffold is found.

The remainder of the paper is organized as follows: In Section 2, we present a few basic concepts and the sequential algorithm on which PSCG is based. In Section 3, we describe PSCG in detail. Section 4 presents a description of the data sets and experiments as well as an interpretation of the results. In Section 5, we give a conclusion.

2 Background

2.1 Notation and Definitions

In the following, all graphs are assumed to be labeled, undirected graphs. To be more precise, a graph and its subgraphs are defined as follows: A labeled *graph* is represented as a 4-tuple $g = (V, E, \alpha, \beta)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges representing connections between all or some of the vertices in V. $\alpha : V \rightarrow L$ is a mapping that assigns labels to the vertices, and $\beta : V \times V \rightarrow L$ is a mapping that assigns labels to the edges. Given two labeled graphs $g = (V, E, \alpha, \beta)$ and $g' = (V', E', \alpha', \beta')$, g' is a *subgraph* of g, $(g' \subseteq g)$ if:

- $V' \subseteq V$
- $E' \subseteq E$
- $\forall x \in V' : \alpha'(x) = \alpha(x)$
- $\forall (x, y) \in V' \times V' : \beta'((x, y)) = \beta((x, y))$

Given two arbitrary labeled graphs $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$, a *common subgraph* of g_1 and g_2, $cs(g_1, g_2)$, is a graph $g = (V, E, \alpha, \beta)$ such that there exists a *subgraph isomorphism* from g to g_1 and from g to g_2. This can be generalized to sets of graphs. The set of common subgraphs of a set of graphs $\{g_1, ..., g_n\}$ is then denoted by $cs(\{g_1, ..., g_n\})$. Moreover, given two graphs g_1 and g_2, a graph g is called a *maximum common subgraph* of g_1 and g_2 if g is a common subgraph of g_1 and g_2 and there exists no other common subgraph of g_1 and g_2 that has more vertices than g. Finally, we define the size of a graph as the number of its vertices, i.e., $|V|$.

2.2 Problem Definition

Structural clustering is the problem of finding groups of graphs sharing some structural similarity. Instances with similar graph structures are expected to be in the same cluster provided that the common subgraphs match to a satisfactory extent. Only connected subgraphs are considered as common subgraphs. The similarity between graphs is defined with respect to some user-defined size threshold. The threshold is set such that the common subgraphs shared among a query graph and all cluster instances make up at least a certain proportion of the size of each graph. A graph is assigned to a cluster provided that there exists at least one such common subgraph whose size is equal or bigger than the threshold. In this way, an object can simultaneously belong to multiple clusters (overlapping clustering) if the size of at least one common subgraph with these clusters is equal or bigger than the threshold. If an object does not share a common subgraph with any cluster that meets the threshold, this object is not included in any cluster (non-exhaustive clustering). Figure 1 provides a sample clustering output for a data set of molecular graphs. The figure illustrates the overlapping and non-exhaustive character of the structural clustering algorithm.

Formally, we frame the problem of structural clustering as follows. Given a set of graph objects $X = \{x_1, ..., x_n\}$, we need to assign them into clusters which may

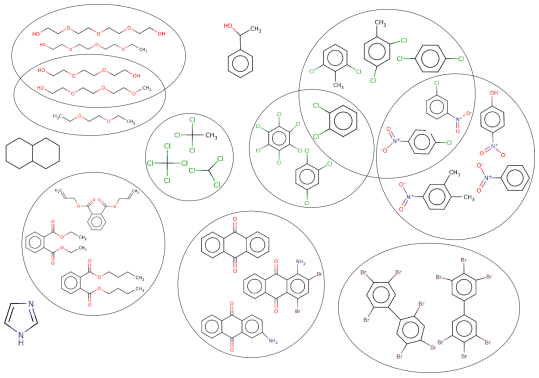


Fig. 1. Example output of PSCG on a subset of the RepDose database (`http://www.fraunhofer-repdose.de`) for $\theta = 0.7$.

overlap with each other. In clustering these objects, one objective is considered: to maximize the average number of objects contained in a cluster, such that at any time for each cluster C there exists at least one common subgraph that makes up a specific proportion, θ, of the size of each cluster member. Considering the state of a cluster $C = \{x_1, ..., x_m\}$[3] at any point in time, the criterion can formally be defined as:

$$\exists\, s \in cs(\{x_1, ..., x_m\}) \forall x_i \in C : |s| \geq \theta |x_i| \tag{1}$$

where s is a subgraph and $\theta \in [0, 1]$ is a user-defined similarity coefficient. According to this goal, a minimum threshold for the size of the common subgraphs shared by the query graph x_{m+1} and the graphs in cluster C can be defined as

$$minSize = \theta\ max(|x_{max}|, |x_{m+1}|), \tag{2}$$

where $\theta \in [0, 1]$ and x_{max} is the largest graph instance in the cluster. To obtain meaningful and interpretable results, the minimum size of a graph considered for cluster membership is further constrained by a $minGraphSize$ threshold. Only graphs whose size is greater than $minGraphSize$ are considered for clustering. Thus, the identification of the general cluster scaffold will not be impeded by the presence of a few graph structures whose scaffold is much smaller than the one the majority of the cluster members share. This will be especially useful in real-world applications that often contain small fragments.

2.3 Sequential Structural Clustering

The proposed parallel clustering approach PSCG extends and improves a structural graph clustering approach proposed recently [8]. In short, the algorithm works as follows. Let $minGraphSize$ be the minimum threshold for the graph size and $minSize$ be the minimum threshold for the size of the common subgraphs

[3] In slight abuse of notation, we use the same indices as above.

specified by the user and defined in Equation 2. In the first step, an initial cluster is created containing the first graph object that is larger than $minGraphSize$. In the following steps, each instance is compared against all existing clusters. In case the query instance meets the $minGraphSize$ threshold and shares at least one common subgraph with one or more clusters that meets the cluster criterion in Equation 2, the instance is added to the respective cluster. Unlike many traditional clustering algorithms, a graph object is allowed to belong to no cluster, since it is possible that an object is not similar to any cluster. In this case, a new singleton cluster is created containing the query graph instance.

For computing common subgraphs, a modified version of the graph mining algorithm gSpan [11] that mines frequent subgraphs in a database of graphs satisfying a given minimum frequency constraint is used. The structural clustering approach requires a minimum support threshold of $minSup = 100\%$ in a set of graphs, i.e., all common subgraphs have to be embedded in all cluster members. For experiments with molecular graph data, gSpan′, an optimization of gSpan for mining molecular databases (`http://wwwkramer.in.tum.de/projects/gSpan.tgz`) is used. As the only interest lies in the determination of at least one common subgraph that meets the minimum size threshold defined in Equation 2, the graph mining algorithm gSpan [11] was modified, to mine frequent subgraphs with a maximum size of $minSize$. More specifically, once the size of the current subgraph reaches $minSize$, it will not be grown any more and search terminates. In this way, the computation of all frequent common subgraphs can be avoided, thus achieving a substantial performance improvement.

3 Parallel Structural Graph Clustering

In this section, we present enhancements and optimizations of the structural clustering algorithm proposed by Seeland *et al.* [8] that enable PSCG to handle large data sets. The main idea of PSCG is to partition the clustering task into independent tasks which are distributed among a set of processes, i.e., each process is responsible for one cluster. The motivation behind partitioning the set of clusters instead of the graph data set is that each process can compare all relevant graph objects, i.e., all graph objects with an index greater than the index of the graph that initiated the singleton cluster, against the assigned cluster without the need to wait for the intermediate results of the other processes. To achieve this, we need a master process which is responsible for managing the cluster results of all processes.

We adopt the *master-worker paradigm* to implement PSCG. The master-worker programming model consists of two kinds of entities: a single master and multiple workers. The master is responsible for decomposing a clustering problem into a subset of clustering tasks and distributing these tasks among a farm of workers (by putting the tasks in a shared queue), as well as for gathering the partial results in order to produce the final computation result. A queue, shared between the master and the workers, is used to represent the shared space where the pending clusters reside. Each worker is responsible for only one cluster at any

Algorithm 1 Master

```
1: stable_sort(graph[]) //see Section 3.2
2: for (i ← 0, num_procs − 1) do
3:     w ← new Worker()
4:     w.start()
5: end for
6: first = 0
7: while |graph[first]| < minGraphSize do
8:     first + +;
9: end while
10: c ← new Cluster(graph[first])
11: queue.add(c)
12: for (i ← first + 1, |graph[]| − 1) do
13:     graph[i].nrClusterComparisons + +
14: end for
15: while (true) do
16:     if (!workers.active && queue.isEmpty) then
17:         for (i ← 0, num_procs − 1) do
18:             w.terminate()
19:         end for
20:         break
21:     end if
22: end while
```

point in time, independently computing one iteration: It pulls a clustering task (input) from the queue, processes the task by comparing all relevant graphs in the graph database against the cluster, and sends the result, i.e., the processed cluster, back to the master (output).

One of the advantages of using this pattern is that the algorithm is based on a dynamic load balancing of the cluster queue, i.e., the algorithm automatically balances the load. This is possible due to the adoption of a receiver-initiated dynamic load balancing approach based on polling: the work set is shared, and the workers continue to pull work from the set until there is no more work to be done. A static load balancing policy is not adequate for our algorithm as the work load is not known in advance and cannot be estimated easily.

In the following sections, we describe the parallel structural clustering algorithm PSCG in more detail.

3.1 Cluster Comparisons

Let *minGraphSize* be the minimum threshold for the graph size and *minSize* be the minimum threshold for the size of the common subgraphs specified by the user and defined in Equation 2. At the beginning of algorithmic execution, we start with an empty set of clusters. In the first step, the master initiates the computation by creating an initial cluster containing the first graph object that is larger than *minGraphSize* (Algorithm 1, line 6-10). The master process is re-

Algorithm 2 Worker

```
1: while (!terminationSignal) do
2:     c ← queue.getCluster()
3:     if (c != null) then
4:         PSCG(c, graphStartIdx, θ, minGraphSize)
5:     end if
6: end while
7: w.terminate()
```

Algorithm 3 Structural Clustering

```
1: procedure PSCG(c,graphStartIdx,θ, minGraphSize)
2:     graphEndIdx ← idx(graph : |graph| ≤ θ · c.min)] //see Section 3.2
3:     for (j ← graphStartIdx, graphEndIdx) do
4:         if (graph[j] ≥ minGraphSize) then
5:             hasCluster ← false
6:             if (s(f_graph[j], f_c) < θ max(|graph[j]|, |c.min|)) then //see Section 3.3
7:                 MISMATCH(c.id, j, j + 1)
8:                 continue
9:             else
10:                minSize ← θ · max(|graph[j]|, |c.max|)
11:                if (!UniqueScaffold) then //see Section 3.4
12:                    minSup ← |c| + 1
13:                    ret ← gSpan'''(graph[j] ∪ c.graphs, minSup, minSize)
14:                else
15:                    minSup ← 2
16:                    ret ← gSpan''(graph[j] ∪ c.scaffold, minSup, minSize)
17:                end if
18:                if (ret = 1) then
19:                    c[last + 1] ← graph[j]
20:                    hasCluster ← true
21:                end if
22:            end if
23:            if (hasCluster = false) then
24:                MISMATCH(c.id, j, j + 1)
25:            end if
26:        end if
27:    end for
28:    if (graphEndIdx + 1 < |graph[]|)) then
29:        MISMATCH(c.id, graphEndIdx + 1, |graph[]|)
30:    end if
31:    results.add(c)
32: end procedure
```

sponsible for putting the initial cluster in the cluster queue (line 11) which stores cluster objects that are exchanged with the workers. Subsequently, the master increases the number of necessary cluster comparisons for all subsequent graphs

Algorithm 4 Maintainance of Cluster Membership Information

```
1: procedure MISMATCH(cId,startId,endId)
2:     for (graphId ← startId, endId − 1) do
3:         graph[graphId].nrMismatches + +
4:         if (graph[graphId].nrCluComp = graph[graphId].nrMism) then
5:             c ← new Cluster(graph[graphId])
6:             queue.add(c)
7:             for (i ← graphId + 1, |graph[]| − 1) do
8:                 graph[i].nrClusterComparisons + +
9:             end for
10:        end if
11:    end for
12: end procedure
```

(explained in more detail later in this section) (line 12-13). In the following steps, idle workers continue to pull one cluster at a time from the queue (Algorithm 2, line 2) and perform clustering (line 4) by comparing all graph instances in the graph database that lie within a specified index range (which will be explained in more detail in Section 3.2) against the assigned cluster (Algorithm 3, line 3). In case a query instance meets the *minGraphSize* threshold and shares at least one common subgraph with the cluster that meets the cluster criterion in Equation 2 (line 18), the instance is added to the respective cluster (line 19). In case a graph object does not belong to any cluster, a new cluster is created. In contrast to the sequential clustering setting, however, in the parallel setting the information whether a graph belongs to a cluster is distributed over the set of workers. Since a new cluster can only be created if it is not assigned to any existing cluster, the master needs to maintain the cluster membership information for all graph instances. In particular, for each graph we need to maintain two cluster membership parameters: the number of necessary cluster comparisons as well as the numbers of clusters the graph does not fit into (denoted as the number of cluster mismatches). If a graph does not belong to a cluster the worker forwards the non-membership information to the master (Algorithm 3,

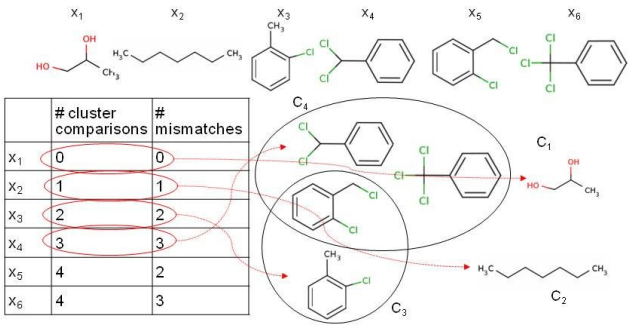


Fig. 2. Graphical illustration of PSCG on a sample data set ($\theta = 0.8$).

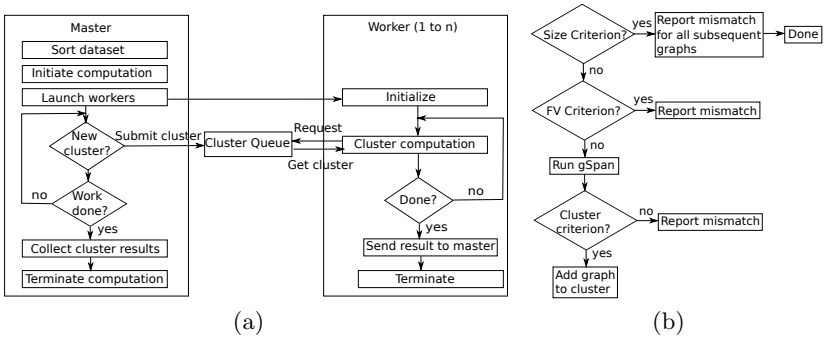


Fig. 3. Flow charts of the parallel structural clustering algorithm PSCG.

line 24). Note, that due to the overlapping nature of the clustering algorithm, a graph can be directly assigned to a cluster in case it meets the cluster criterion without informing the master. Each time a worker reports a cluster mismatch for a graph, the master first increases the mismatch parameter for the graph (Algorithm 4, line 2-3) and then checks the two cluster membership parameters. If the number of necessary cluster comparisons is equal to the number of cluster mismatches (line 4), suggesting that the corresponding graph does not belong to any cluster, a new cluster is created (line 5). The master puts the cluster in the task queue (line 6) and increases the cluster comparison parameter for all subsequent graphs in the graph data set (line 7-8). A graphical illustration of the clustering process on a sample data set of molecular graphs is shown in Figure 2, where large circles represent clusters and the single structures outside denote singleton clusters. The table contains the cluster membership parameters maintained by the master. Once a worker is done with an iteration, the resulting cluster is added to the result queue managed by the master (Algorithm 3, line 31). Figure 3 illustrates the master-worker paradigm of PSCG in a flow chart.

As in the sequential clustering algorithm, we use gSpan″ for computing common subgraphs. Given that pairwise subgraph similarity computation is very expensive, it would be highly desirable to reduce the number of subgraph computations. Therefore, we introduced the following cluster exclusion criteria to avoid unnecessary calls to the gSpan″ algorithm in the first place: a refined cluster membership test based on node feature vectors of graphs, and a clustering exclusion criterion based on the size of graph objects which requires the graph data set to be sorted according to size. These criteria are used to perform a search space pruning on the actual clustering. The aim of search space pruning is to reduce the number of graph candidates in the database that need to undergo an expensive, full fledged graph matching process. Further, to reduce gSpan running times for larger clusters, we define a cluster representative for each cluster composed of the common cluster scaffold once this scaffold is unique and thus also minimal. In the following three subsections, we describe the employed cluster exclusion criteria and the intuition behind the definition of the cluster representative in more detail. The impact of these algorithmic improvements will be investigated in Section 4.

3.2 Size based Exclusion Criterion

The cluster criterion defined in Equation 2 constrains the set of graphs being considered for clustering. More precisely, only graphs in a certain size range are considered for comparison with a specific cluster, i.e., graphs whose sizes lie in the range $[\lceil \theta x_{max} \rceil, \lfloor \frac{1}{\theta} x_{min} \rfloor]$, where x_{min} is the smallest and x_{max} is the largest graph instance in the cluster. The lower bound of the size range ensures that only graph instances that are equal to or larger than the minimum required size for at least one common subgraph, $minSize$, are considered for cluster membership. This is necessary since at any point in time at least one common subgraph should make up a proportion θ of the size of each cluster member. The upper bound excludes query instances that are larger than $minSize$ and thus would break up an existing cluster. Incorporating this information in the clustering process would give us the possibility to avoid comparing a cluster to the complete database.

To effectively employ the size based criterion, we sort the data set in increasing order of graph size. Thus, we do not need to compare the subsequent graphs against a cluster, once a query graph exceeds the upper bound of the size range (see Figure 3(b)). To preserve the incremental character (i.e., each graph in the graph database is only processed once by comparing it against all existing clusters) of the structural clustering algorithm [8], we need to make sure that the graph index corresponding to the lower bound is greater than the index following the index of the graph instance that initiated the assigned singleton cluster. However, due to the ordering of the data set by size, the graph index corresponding to the lower bound is always equal to or smaller than the index of the graph that initiated the singleton cluster. Thus, the graph indices that are considered for comparison against a cluster lie in the range $[idx(x_{min}) + 1, idx(x : |x| \leq \lfloor \frac{1}{\theta} x_{min} \rfloor)]$, where x_{min} is the smallest graph in the cluster. Due to the ordering of the data set, this graph corresponds to the graph that initiated the clustering. Figure 4 illustrates the use of the size based exlusion criterion during the clustering process on a data set of eight molecular graphs.

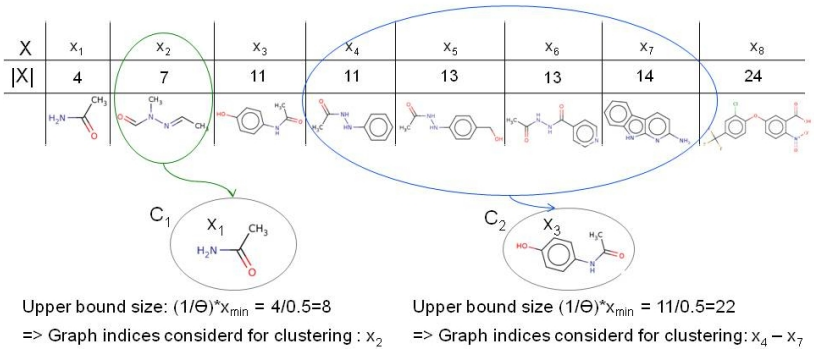


Fig. 4. Example use of the size based cluster exclusion criterion on a data set of chemical compounds containing eight graphs ($\theta = 0.5$).

3.3 Exclusion Criterion based on Node Feature Vectors

The second clustering exclusion criterion is based on a set abstraction of graphs, i.e., a numerical feature vector representing the number of node types in a graph. The underlying idea is that for two graphs the overlapping node set represents an upper bound for the size of the maximum common subgraph. Thus, given a query instance, we can skip the common subgraph computation with a cluster if the size of the overlapping node set of the query graph and the cluster representantive is smaller than $minSize$.

Formally, during the preprocessing phase of structural clustering, we represent each graph g_i by a numerical feature vector $\boldsymbol{f}_{g_i} = (f^1_{g_i}, .., f^n_{g_i})$ corresponding to a set of vertex types $l_1, ..., l_n$. Each entry in the feature vector records the number of a specific vertex type occuring in the respective graph. Let $f^k_{g_i}$ denote the numerical feature associated with the vertex type v_k. Each cluster C_j is represented by a vector $\boldsymbol{f}_{C_j} = (f^1_{C_j}, ..., f^n_{C_j})$ defined in terms of the overlap of the feature vectors of the instances in that cluster, i.e., the common vertex type set shared by all cluster instances. The similarity s between $\boldsymbol{f}_{g_i}$ and $\boldsymbol{f}_{C_j}$ is computed by summing up the minimum of each pair of feature vector components

$$s(\boldsymbol{f}_{g_i}, \boldsymbol{f}_{C_j}) = \sum_k (min(\{\boldsymbol{f}^k_{g_i} \in \boldsymbol{f}_{g_i}\} \cup \{\boldsymbol{f}^k_{C_j} \in \boldsymbol{f}_{C_j}\})) \tag{3}$$

representing an upper bound on the size of the maximum common subgraph (Algorithm 3, line 6). If the similarity $s(\boldsymbol{f}_{g_i}, \boldsymbol{f}_{C_j})$ is lower than the minimum threshold for the size of the common subgraphs, $minSize$ (Equation 2), i.e., $s(\boldsymbol{f}_{g_i}, \boldsymbol{f}_{C_j}) < minSize$, we omit the computation of the common subgraphs, report the cluster mismatch to the master (line 7) and continue with the next cluster comparison (line 8). In this way, we eliminate graphs with a limited degree of resemblance to the target cluster, and increase the overall speed of the algorithm. Figure 5 shows a sample application of the feature vector criterion. In this example, the query graph x_{m+1} is compared against a cluster containing two graphs. As the similarity between the node feature vector of the query graph and the cluster is lower than $minSize$, the query graph is not considered for the cluster membership test, i.e., the computation of the common subgraphs can be omitted.

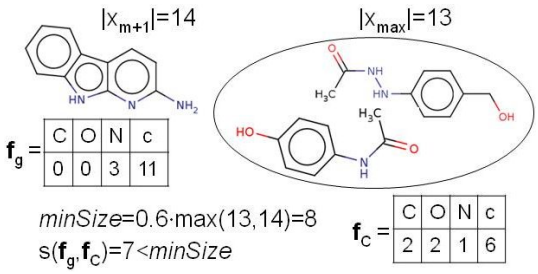


Fig. 5. Example use of the feature vector based cluster exclusion criterion ($\theta = 0.6$).

3.4 Definition of a Cluster Representative

As mentioned in Section 2.3, the structural clustering algorithm limits subgraph mining to the search of one common subgraph that satisfies the minimum size threshold, $minSize$ to avoid the computation of all frequent common subgraphs. This limitation forces us to compare each query graph against all cluster members which may have a remarkable impact on the runtime of gSpan, in particular for larger clusters. To reduce running times, we define a cluster representative for each cluster once all cluster members share a unique cluster scaffold, i.e., the minimum required common subgraph is the only common subgraph all cluster members have in common. Since in the structural clustering algorithm [8] subgraph mining is terminated once a common subgraph is found that satisfies $minSize$, the existence of further common subgraphs is unknown. Therfore, we need to go one level deeper in the subgraph mining process and check if there exists at least another common subgraph with size equal to or greater than $minSize$. In the pseudocode, this modification of gSpan is called gSpan$'''$ (Algorithm 3, line 13). As soon as all graphs in a cluster share no more than one common subgraph, this unique subgraph is used as the cluster representative. In the following, all subsequent query graphs are compared against the cluster representative instead of comparing it against all graphs in the cluster (line 15-16). Further, subgraph mining is terminated as soon as a common subgraph of size $minSize$ is found that is covered by the query graph and the cluster representative, i.e., gSpan$''$ is used. Note, that the reason for not defining a cluster representative before the existence of a unique cluster scaffold is due to the following two reasons. First, there may exist at least another common subgraph of size $minSize$. By using the first common subgraph found as cluster representative, it may be the case that the query graph and the cluster representative share a common subgraph of size $minSize$ that is not the first common subgraph. In this case, by mistake the query graph would not be assigned to the cluster. Second, there may exist larger subgraphs. By ignoring the existence of these subgraphs and using the first common subgraph found as cluster representative, it may be the case that the $minSize$ threshold is smaller than the size of the common subgraph shared by the query graph and the cluster representative. Thus, the query graph would not be assigned to the cluster even if there exist larger common subgraphs that fulfill the size threshold.

4 Experimental Results

To evaluate the efficiency of our parallel structural clustering algorithm PSCG, introduced in Section 3, we conducted several experiments on several publicly available data sets of molecular graphs. In this section, we describe the data sets, the experimental set-up and the results.

4.1 Test Environment and Data Sets

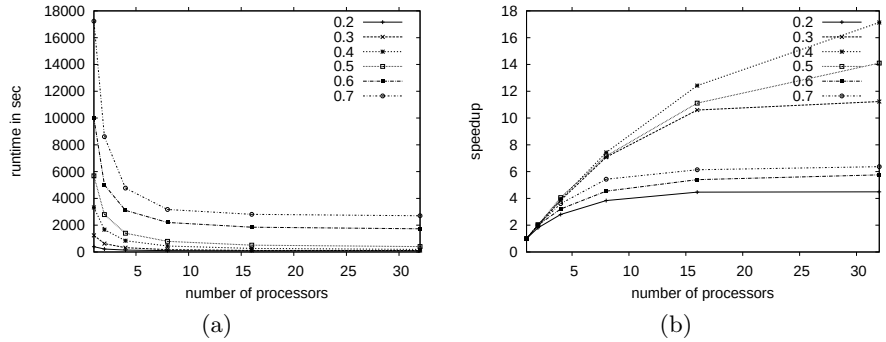
The clusterings on the data sets containing up to 200,000 graphs were carried out on a SUN x4600 system with 32 AMD Opteron CPU cores (8 CPU sock-

ets with 4 cpu cores) using the multi-threaded version of the algorithm. The processor in each node runs at 2.5 GHz with 2 GB of main memory. The clusterings on the data set containing 300,000 structures were carried out using the MPI parallelized version of the algorithm. Here, the compute cluster consists of 2016 AMD Opteron (Magny-Cours) CPU cores (42 Dell R815 nodes with 48 cpu cores and 128-256 GB main memory) and Qlogic infiniband interconnects. The algorithm was implemented in C++ using the boost libraries (`www.boost.org`) for multi-threading support. For the experiments, we employed the chemical domain as our application area by using real data sets of molecular graphs. The first data set contains the first 10,000 structures of the NCI anti-HIV database (`http://dtp.nci.nih.gov/docs/aids/aids_data.html`) which contains 36,255 compounds. The second data set, ChemDB, contains nearly 5 M commercially available small molecules [2, 3]. We created data sets sized from 100,000 to 300,000 graphs from this data set using random sampling.

4.2 Performance Evaluation

We investigated the runtime performance of PSCG for different numbers of processors (1, 2, 4, 8, 16 and 32) and different values of θ using the first 10,000 graph structures from the NCI anti-HIV database. The runtime performance of PSCG was evaluated according to the speedup factor. Speedup (S) is defined as a ratio of the time taken in running the sequential algorithm (T_s) to the time taken in running the parallel algorithm (T_p) with P processors, i.e., $S = \frac{T_s}{T_p}$.

Figure 6 shows the execution time and the speedup for different values of θ. The results indicate that our algorithm scales well with the number of processors and has a good speedup which is close to linear for certain parameter settings, i.e., for smaller values of θ. For larger similarity coefficients, there is a higher number of computationally more demanding cluster comparisons, especially at the end of the clustering when the graphs become larger and the runtime degenerates.



(a) (b)

Fig. 6. (a) Execution time and (b) speedup of PSCG on the first 10,000 graphs of the NCI anti-HIV data set.

4.3 Effects of Algorithm Improvements

We investigated the impact of the algorithm improvements presented in Section 3.2, 3.3 and 3.4 on the performance of PSCG. For this, we ran the algorithm on the NCI anti-HIV data set with 32 processors using (i) no optimizations, (ii) only the size based exclusion criterion, (iii) only the feature vector based exclusion criterion, (iv) both the size and feature vector based criteria and (v) all optimizations including the definition of a cluster scaffold once it is unique. Figure 7 shows the runtime reduction and an overview of the relative frequency of both exclusion criteria as well as the frequency of gSpan calls. The results indicate that significant performance improvements, especially for $\theta \leq 0.5$, can be achieved with the application of the cluster exclusion criteria and the definition of a cluster scaffold.

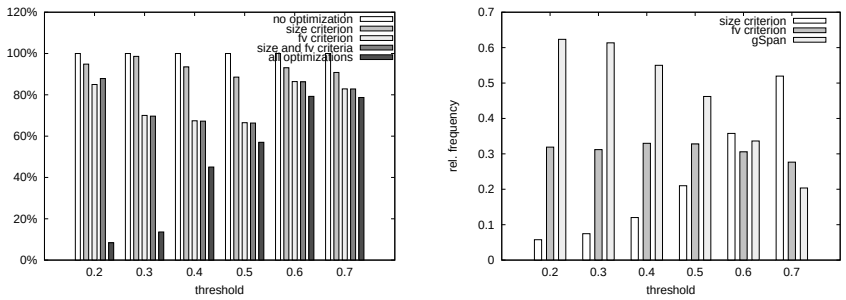


Fig. 7. a) Runtime reduction due to algorithm improvements and b) relative frequency of size-based and feature vector based exclusion criterion and number of gSpan calls.

4.4 Comparison to Sequential Structural Clustering

We compared the runtime performance of the sequential structural clustering algorithm [8] with PSCG on the first 10,000 structures of the NCI anti-HIV data set. For accurate comparison, we used the same experimental setup. We only show the experimental results for $\theta \in [0.2, 0.5]$, since for $\theta \geq 0.5$, the sequential algorithm did not terminate within a certain timeout period. Table 1 shows the runtime performance of both clustering versions. The runtime advantage of PSCG over the sequential clustering version is clear, showing improved computation efficiency by factors of 300 fold to 1900 fold for PSCG. The reasons for this can be explained by the following improvements in PSCG. First, the clustering task is partitioned into independent tasks which are distributed among a set of workers. Each worker compares the graph structures in the data set against the assigned cluster without the need to wait for the intermediate results of the other processes. Second, we introduced two clustering exclusion criteria which reduce the number of cluster membership tests. Third, we defined a cluster representative once the scaffold of a cluster is unique, to avoid cluster

comparisons with all cluster members. Fourth, we reduced the invocation overhead of the individual gSpan runs. This optimization is especially efficient for gSpan runs with low overall runtimes.

Table 1. Runtime (in sec) of the sequential clustering version vs. PSCG on the first 10,000 graphs of the NCI anti-HIV data set for different values of θ.

θ	0.2	0.3	0.4	0.5
t_{seq}	747,000	1,068,420	1,434,780	2,087,280
t_{par}	396	1,244	3,394	6,235

4.5 Experiments on Large Graph Data Sets

We tested PSCG on three data sets sampled from the ChemDB data set containing 100,000, 200,000 and 300,000 graphs respectively. For the experiments, we used 32 CPUs for the data sets with 100,000 and 200,000 graphs. For the data set containing 300,000 graphs we used 96 CPUs for $\theta = 0.4$. For $\theta = 0.6$, we used 96 (48) CPUs to cluster the first (second) half of the data set. The rationale for the change in the CPU number is that the parallel efficiency of the algorithm can change over the runtime of the algorithm (i.e., towards the end a large number of workers may be idle constantly). The MPI version contains a checkpoint/restart facility which allowed us to adjust the number of used CPU cores to account for this by manually balancing the workload on the cluster. Tables 2 and 3 show the runtime performance as well as the number of created clusters on the sampled data sets for $\theta = 0.4$ and $\theta = 0.6$ using all three previously described algorithmic improvements.

Table 2. Runtime (in sec) for the sampled data sets.

$\|D\|$	$\theta = 0.4$	$\theta = 0.6$
100,000	31,103 •	67,563 •
200,000	122,204 •	349,568 •
300,000	610,577 ○	1,163,761 ⋆

Table 3. Number of clusters for the sampled data sets.

$\|D\|$	$\theta = 0.4$	$\theta = 0.6$
100,000	4,112	16,295
200,000	6,096	25,685
300,000	9,811	38,775

•: 32 processors ○: 96 processors ⋆: first half: 96 processors, second half: 48 processors

5 Conclusion

In this paper, we presented PSCG, a parallel and improved version of a recently proposed structural graph clustering algorithm [8]. PSCG uses a task partitioning approach and makes use of two clustering exclusion criteria to reduce cluster membership tests. Further, to reduce gSpan running times for larger clusters, we define a cluster representative for each cluster composed of the common cluster scaffold once this scaffold is unique. To study the effectiveness of our proposed algorithm for clustering large data sets, we conducted extensive experiments. The

experimental results suggest that the algorithm scales well with the increasing size of the data and, for certain parameter settings, speeds up nearly linearly with the increasing number of processors. For real world data sets, this algorithm is able to handle a much greater number of graph objects compared to previously proposed structure-based clustering algorithms. Given these performance improvements, our algorithm should already be applicable to the large structure databases from virtual screening.

References

1. C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. XProj: a framework for projected structural clustering of XML documents. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 46–55, New York, NY, USA, 2007. ACM.
2. J. Chen, S. J. Swamidass, Y. Dou, and P. Baldi. ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinf.*, 21:4133–4139, 2005.
3. J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi. ChemDB updatefull-text search and virtual chemical space. *Bioinf.*, 23:2348–2351, 2007.
4. M. S. Hossain and R. A. Angryk. GDClust: A graph-based document clustering technique. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 417–422, Washington, DC, USA, 2007. IEEE Computer Society.
5. M. J. McGregor and P. V. Pallai. Clustering of large databases of compounds: Using the MDL "keys" as structural descriptors. *Journal of Chemical Information and Computer Sciences*, 37(3):443–448, 1997.
6. J. W. Raymond, C. J. Blankley, and P. Willett. Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures. *J. Mol. Graph. Model.*, 21(5):421–433, 2003.
7. J. W. Raymond and P. Willett. Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases. *J. Comput. Aided. Mol. Des.*, 16(1):59–71, 2002.
8. M. Seeland, T. Girschick, F. Buchwald, and S. Kramer. Online structural graph clustering using frequent subgraph mining. In *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases*, ECML PKDD'10, pages 213–228, 2010.
9. M. Stahl and H. Mauser. Database clustering with a combination of fingerprint and maximum common substructure methods. *J. Chem. Inf. Model.*, 45:542–548, 2005.
10. K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 953–960, New York, NY, USA, 2006. ACM.
11. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724, 2002.
12. T. Yoshida, R. Shoda, and H. Motoda. Graph clustering based on structural similarity of fragments. In *Jantke, K.P., et al. (eds.) Federation over the Web. LNCS (LNAI)*, volume 3847, pages 97–114. Springer Heidelberg, 2006.