Enumerating common molecular substructures

Martin S. Engler[1], **Mohammed El-Kebir**[2], **Jelmer Mulder**[3], **Alan E. Mark**[4], **Daan P. Geerke**[3], **and Gunnar W. Klau**[*,5]

[1]Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[2]Princeton University, NJ, USA
[3]VU University Amsterdam, The Netherlands
[4]University of Queensland, Australia
[5]Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany

ABSTRACT

Finding and enumerating common molecular substructures is an important task in cheminformatics, where small molecules are often modeled as molecular graphs. We introduce the problem of enumerating all maximal k-common molecular fragments of a pair of molecular graphs. A k-common fragment is a common connected induced subgraph that consists of a common core and a common k-neighborhood. It is thus a generalization of the NP-hard task to enumerate all maximal common connected induced subgraphs (MCCIS) of two graphs, which corresponds to the $k = 0$ case.
We extend the MCCIS enumeration algorithm by Ina Koch and apply algorithm engineering techniques to solve practical instances fast for the general $k > 0$ case, which is relevant, for example, for automatically generating force field topologies for molecular dynamics (MD) simulations. We find that our methods achieve good performance on a real-world benchmark of all-against-all comparisons of 255 molecules.
Our software is available under the LGPL open source license at https://github.com/enitram/mogli.

Keywords: molecular graphs, molecular dynamics simulations, subgraph enumeration

INTRODUCTION

Finding and enumerating common molecular substructures is an important task in cheminformatics. Applications include computing distances between molecules, screening chemical libraries for matching fragments, drug lead identification for rational molecular design, protein-ligand docking, predicting biological activity, reaction site modeling, and the interpretation of mass spectra (Raymond and Willett, 2002)

For these tasks, small molecules are often modeled as *molecular graphs*, where nodes represent the atoms and edges the chemical bonds between the atoms. In this setting, common molecular substructures correspond to common connected induced subgraphs, which gives rise to the computational problem of finding a maximum common connected induced subgraph (MCCIS) of two input molecular graphs. Many heuristics (Rahman et al., 2009; Englert and Kovács, 2015) and exact approaches (McCreesh et al., 2016; Droschinsky et al., 2017) have been proposed for MCCIS. A frequent variant of MCCIS is to enumerate all maximal common connected induced subgraphs, which we refer to as MCCIS–E. Koch (2001) proposed an exact algorithm for MCCIS–E. Furthermore, finding common induced subgraphs has also important applications in other fields of science such as computer vision and image recognition (Foggia et al., 2014).

We encountered this problem in the setting of generating force field parametrizations of molecules, or *topologies*, for molecular dynamics (MD) simulations. A molecular topology consists of partial charges for the atoms, van der Waals parameters, bond, angle and (improper) dihedral parameters. The Automated Topology Builder (ATB) (Malde et al., 2011) is a web server that generates *de novo* topologies for the GROMOS force field (Oostenbrink et al., 2004; Schmid et al., 2011). For larger molecules, however, these computations, which are based on quantum-mechanical simulations, can become prohibitively expensive. In previous work we studied the problem of improving the consistency and utility of the partial charges assigned to atoms by identifying atoms that could be used to form *charge groups*, which can be collectively assigned formal charges $(\ldots,-1,0,1,\ldots)$ (Canzar et al., 2013). Currently, the computational bottleneck has shifted towards determining the

[*]Corresponding author: gunnar.klau@hhu.de

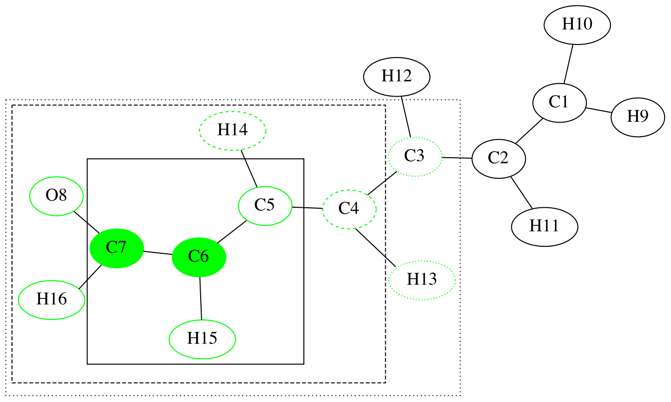


Figure 1. Example of k-neighborhoods and fragments. Boxes illustrate the k-neighborhoods for $k = 1, 2$ and 3 (solid, dashed and dotted lines) of the node C6. Nodes C6 and C7 form a fragment (filled green nodes). The 1-shell includes all nodes with solid green borders, 2-shell all nodes of the 1-shell plus all nodes with dashed green borders and the 3-shell all nodes of the 2-shell and the node C3.

initial partial charges. Since the ATB contains a repository of already parameterized molecules, our aim is to parameterize a query molecule using this database. This requires the identification of all matching substructures between the query molecule and all molecules present in the database.

The primary challenge associated with transferring parameters between molecules is that the properties of a given atom (in particular the partial charges) are heavily dependent on the atoms to which it is attached and the nature of its local environment. Thus, given a matching fragment common to two molecular graphs, the node parameters (labels) of the atoms in the core are more trustworthy than those in the periphery. This is because the atoms at the core of the fragment do not depend so strongly on the neighborhood as the fragment atoms at the periphery, which are likely to differ in the query molecule. We therefore divide matching fragments into a core region and a shell or buffer region in order to assign the query atoms the parameters of the core regions common to many matching fragments in the database. Finding these fragments leads to a generalization of MCCIS–E, and we therefore introduce the problem of finding all maximal k–common fragments (k-MCF–E). These are all matching substructures consisting of a matching core and a matching k–neighborhood.

Note that k-MCF–E is equal to MCCIS–E for $k = 0$. This enables us to use the algorithm by Koch (2001) as a basis, which we then extend to the more general common fragment problem. This algorithm reduces the common subgraph problem to finding all cliques of a certain type in a large product graph. We spend considerable effort in increasing the performance of the clique enumeration algorithm and apply different algorithm engineering techniques to keep the size of this product graph manageable, while still being able to enumerate all maximal common substructures. In addition to facilitating the current application these advances are also beneficial for the special case $k = 0$, where current exact approaches do not scale with the needs to make more and more comparisons, as needed, for example, in screening large chemical libraries.

Our code is publicly available at https://github.com/enitram/mogli under the LGPL v3 open source license.

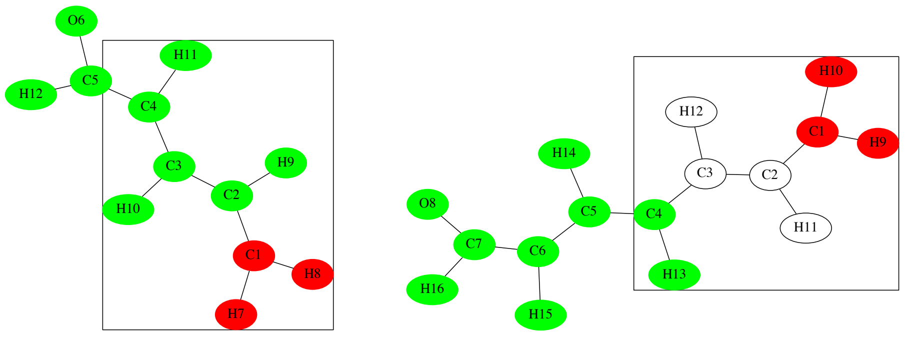
PROBLEM FORMULATION AND COMPLEXITY

A molecular graph is a simple graph $G = (V, E)$ whose nodes and edges correspond to atoms and bonds, respectively. In this work, nodes are labeled by their partial charge $w : V \to \mathbb{R}$ and their atom type $t : V \to \Sigma$ where Σ is the set of all atom types. We could envisage cases where nodes are labeled with more properties.

Definition 1. *The* k-neighborhood *of a node* $u \in V$ *is defined recursively as*

$$N^k(u) = \begin{cases} \{u\}, & \text{if } k = 0, \\ N^{k-1}(u) \cup \{w \mid (v,w) \in E, v \in N^{k-1}(u)\}, & \text{if } k \geq 1. \end{cases}$$

For a subset $V' \subseteq V$*, we define* $N^k(V')$ *to be the set* $\bigcup_{v \in V'} N^k(v) \setminus V'$.



(a) G_1 **(b)** G_2

1	(C4 , C5), (H11 , H14), (C3 , C6), (H10 , H15)
2	(C3 , C6), (H10 , H15), (C2 , C5), (H9 , H14)
3	(C4 , C3), (H11 , H12), (C3 , C2), (H10 , H11)
4	(C3 , C2), (H10 , H11), (C2 , C3), (H9 , H12)
5	(C4 , C3), (H11 , H12), (C3 , C4), (H10 , H13), (C2 , C5), (H9 , H14)
6	(C4 , C4), (H11 , H13), (C3 , C5), (H10 , H14), (C2 , C6), (H9 , H15)
7	(C4 , C2), (H11 , H11), (C3 , C3), (H10 , H12), (C2 , C4), (H9 , H13)
8	(C4 , C5), (H11 , H14), (C3 , C4), (H10 , H13), (C2 , C3), (H9 , H12)
9	(C1 , C1), (H7 , H10), (H8 , H9), (C4 , C4), (H11 , H13), (C3 , C3), (H10 , H12), (C2 , C2), (H9 , H11)
10	(C5 , C7), (H12 , H16), (O6 , O8), (C4 , C6), (H11 , H15), (C3 , C5), (H10 , H14), (C2 , C4), (H9 , H13)

(c) Maximal common fragments

Figure 2. There are ten maximal 1-common fragments. The last row corresponds to the fragments colored in green. Observe that the 1-neighborhood of the green fragment in G_1 consists of C1 and in G_2 it consists of C3, both of which have the same atom type (C). The red fragments are 1-common fragments, but are not maximal—they are each contained within a larger 1-common fragment as shown by the boxes (fragment number 9).

Note that $|N^1(u)| = \deg(u) + 1$, where $\deg(u)$ denotes the degree of node u. We denote a subgraph of a graph $G = (V, E)$ induced by $V' \subseteq V$ as $G[V']$. Given molecules $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with atom types $t_1 : V_1 \to \Sigma$ and $t_2 : V_2 \to \Sigma$ and $k \in \mathbb{N}$, we define a k-common fragment and its shell as follows.

Definition 2 (fragment)**.** *Given $k \in \mathbb{N}$, a* k-common fragment *is a triple (V_1', V_2', h) with $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ and a bijection $h : V_1' \cup N^k(V_1') \to V_2' \cup N^k(V_2')$ such that*

(i) $G_1[V_1']$ and $G_2[V_2']$ are connected,

(ii) (u, v) is an edge in $G_1[V_1' \cup N^k(V_1')]$ if and only if $(h(u), h(v))$ is an edge in $G_2[V_2' \cup N^k(V_2')]$,

(iii) $t_1(v) = t_2(h(v))$ for all $v \in V_1' \cup N^k(V_1')$.

Definition 3 (shell)**.** *The* shell *of a k-common fragment (V_1', V_2', h) is given by $(N^k(V_1'), N^k(V_2'))$.*

Definition 4. *A k-common fragment (V_1', V_2') is* maximal *if there exists no k-common fragment (V_1'', V_2'') such that $V_1' \subsetneq V_1''$ and $V_2' \subsetneq V_2''$.*

See Figure 1 for an illustration of these definitions. We can now formally state the problem of finding all common fragments between two molecules. See also Figure 2 for an example.

Problem 1 (k-MCF–E)**.** *Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with atom types $t_1 : V_1 \to \Sigma$ and $t_2 : V_2 \to \Sigma$ and $k \in \mathbb{N}$, find the set of all maximal k-common fragments.*

Enumerating all maximal k-common fragments (k-MCF–E) generalizes the problem of enumerating all maximal common connected induced subgraphs (MCCIS–E) of two graphs, which corresponds to the case $k = 0$. This problem was already studied by Koch (2001). The underlying optimization problem MCCIS, that is, to find a maximum common induced connected subgraph is NP-complete (Garey and Johnson, 1990), which makes the enumeration problems NP-hard as well.

BASIC ALGORITHM

It is well-known that isomorphic subgraphs correspond to cliques in the product graph of the two input graphs (Levi, 1973). Koch (2001) extended this concept to connected common subgraphs, by introducing different types of edges in the product graph: c–edges and d–edges. The edge type specifies whether the corresponding node pairs in the original graphs are both connected by an edge or both not. The author then showed that the problem of enumerating all common connected induced subgraphs can be reduced to finding all maximal cliques in the product graph that also contain a spanning tree of c-edges. We proceed analogously and, in the following, explain the concepts for k-common fragments. The case $k = 0$ is identical to the construction used in (Koch, 2001).

Definition 5. *Given a graph $G = (V,E)$, a node $v \in V$ and $k \in \mathbb{N}$, the* k-neighborhood subgraph $S^k(v)$ *is the induced subgraph $G[V \cup N^k(v)]$ of G rooted at node v.*

For the fragment problem, the product graph is defined as follows:

Definition 6. *Given $k \in \mathbb{N}$, $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$, the* k-product graph $G_1 \otimes G_2 = (V_{12},E_{12})$ *has node set $V_{12} = \{(u,v) \in V_1 \times V_2 \mid t(u) = t(v)\}$ such that for all (u,v) in V_{12} there is a bijection $h : V_1 \to V_2$ with*

(i) *(u',v') is an edge in $S^k_1(u)$ if and only if $(h(u'),h(v'))$ is an edge in $S^k_2(v)$,*

(ii) *$t_1(u') = t_2(h(u'))$ for all $u' \in V_1 \cup N^k(u)$,*

(iii) *$h(u)$ is root of $S^k_2(v)$*

and an edge in E_{12} between (u,v) and (u',v') if and only if $u \neq u'$ and $v \neq v'$, and either $(u,u') \in E_1$ and $(v,v') \in E_2$, or $(u,u') \notin E_1$ and $(v,v') \notin E_2$.

Following Koch (2001), we distinguish between c- and d-edges in the product graph.

Definition 7. *An edge $((u,v),(u',v')) \in E_{12}$ is a* c-edge *if $(u,u') \in E_1$ and $(v,v') \in E_2$, otherwise it is a* d-edge*.*

The c-neighborhood $N_c(v)$ of a node $v \in V_{12}$ is the set $\{u \mid (u,v) \in E_{12}, (u,v) \text{ is a } c\text{-edge}\}$. Conversely, $N_d(v)$ is the d-neighborhood of a node $v \in V_{12}$. Now we define a c-clique as follows.

Definition 8. *A* c-clique *$C \subseteq V_{12}$ is a clique in G_{12} that also contains a spanning tree of c-edges.*

Definition 9. *A c-clique $C \subseteq V_{12}$ is* maximal *if there is no c-clique C' such that $C \subset C'$.*

In order to ensure connectivity, we aim to find maximal c-cliques. For $k = 0$, Koch showed that c-cliques correspond to 0-common fragments, that is, maximal common connected induced subgraphs, of the same size. The proofs are analogous for $k \geq 1$, and we omit them here.

Lemma 1. *A c-clique C in G_{12} corresponds to a k-common fragment of size $|C|$.*

Proof. See (Koch, 2001). □

See Figure 3 for an illustration of the relation between common fragments and c-cliques in the product graph. As in (Koch et al., 1996; Koch, 2001), we identify maximal c-cliques by adapting the classic Bron-Kerbosch algorithm (Bron and Kerbosch, 1973):

Algorithm 1: CCLIQUES(P,D,R,X,S)

```
Input: $P$, $D$, $X$ and $S$ are disjoint sets of nodes adjacent to all nodes in $R$ whose nodes induce a
       $c$-clique in $G_{12}$.
if $P \cup X = \emptyset$ then
    Report $R$
else
    Choose $u \in P \cup X$
    foreach $v \in P \setminus N(u)$ do
        $P' \leftarrow P \cup (D \cap N_c(v))$
        $D' \leftarrow D \setminus N_c(v)$
        $X' \leftarrow X \cup (S \cap N_c(v))$
        $S' \leftarrow S \setminus N_c(v)$
        CCLIQUES$(P' \cap N(v), D' \cap N(v), R \cup \{v\}, X' \cap N(v), S' \cap N(v))$
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
```

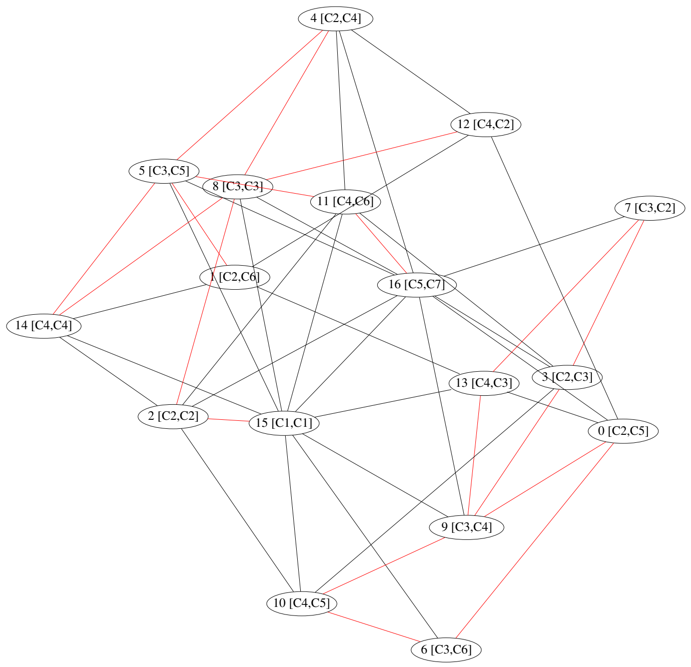


Figure 3. The product graph of the two graphs in Fig. 2. Product nodes corresponding to degree 1 nodes in the original graphs have been ignored. Also, c-edges are colored red and d-edges are colored black. The maximal c-cliques are: $\{10,6\},\{10,9\},\{13,7\},\{7,3\},\{13,9,0\},\{14,5,1\},\{12,8,4\},\{10,9,3\},\{16,11,5,4\}$ and $\{15,14,8,2\}$.

Lemma 2 (Koch (2001)). *There are five invariants in the algorithm.*

1. *R is a c-clique.*
2. *Each node $v \in P$ is adjacent to all nodes in R and c-adjacent to at least one node in R.*
3. *Each node $v \in D$ is d-adjacent to all nodes in R.*
4. *Each node $v \in X$ is adjacent to all nodes in R and c-adjacent to at least one node in R, and all maximal c-cliques containing $R \cup \{v\}$ have already been reported.*
5. *Each node $v \in S$ is d-adjacent to all nodes in R and all maximal c-cliques containing $R \cup \{v\}$ have already been reported.*

Invoking CCLIQUES(P,D,R,X,S) lists all maximal c-cliques in the subgraph comprised of the nodes in R, some of the nodes in $P \cup D$ and none of the nodes in $X \cup S$. So by invoking CCLIQUES($N_c(v)$, $N_d(v)$, $\{v\}$, $\emptyset$, $\emptyset$) all maximal c-cliques containing $v \in V_{12}$ will be listed.

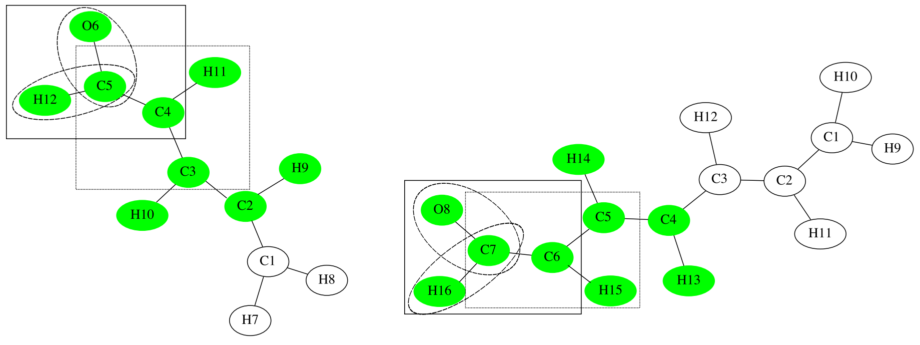
Algorithm 2: ALLCCLIQUES()

```
1 Let $v_1, v_2, \dots, v_n$ be some ordering of the vertices
2 for $i \leftarrow 1$ to $n$ do
3     $P \leftarrow N_c(v_i) \cap \{v_{i+1}, \dots, v_n\}$
4     $D \leftarrow N_d(v_i) \cap \{v_{i+1}, \dots, v_n\}$
5     $X \leftarrow N_c(v_i) \cap \{v_1, \dots, v_{i-1}\}$
6     $S \leftarrow N_d(v_i) \cap \{v_1, \dots, v_{i-1}\}$
7     CCLIQUES($P, D, \{v_i\}, X, S$)
```

In order to speed up the basic algorithm we apply some well-known improvements for Bron-Kerbosch-like algorithms: First, we replace line 4 of Algorithm 1 by

```
4' Choose $u \in P \cup X$ maximizing $|P \cap N(u)|$
```

This pivot rule has been introduced by Bron and Kerbosch (1973) and ensures that when calling CCLIQUES($N_c(v)$, $N_d(v)$, $\{v\}$, $\emptyset$, $\emptyset$) all maximal c-cliques containing v are listed only once and not multiple times. Tomita et al. (2006) proved that this strategy reduces the running time to $O(3^{n/3})$,



(a) G_1 **(b)** G_2

Figure 4. The two graphs G_1 and G_2 from Fig. 2 with the largest fragment colored in green. The 1-neighborhood subgraphs $S_1^1(\text{C5})$ and $S_2^1(\text{C7})$ (solid boxes) are isomorphic. Note that the 1-neighborhood subgraphs of their degree-1 neighbors (dashed ovals) are subgraphs of $S_1^1(\text{C5})$ and $S_2^1(\text{C7})$. The 1-neighborhood subgraphs of their neighbors with degree larger than one (dotted boxes) are not subgraphs of $S_1^1(\text{C5})$ and $S_2^1(\text{C7})$
.

where n is the number of vertices in the input graph. This is optimal as a function of n, because there are graphs with $3^{n/3}$ cliques in a graph (Moon and Moser, 1965). Cazals and Karande (2008) provide a coherent overview of the work of Bron and Kerbosch, Koch and Tomita et al..

The second improvement concerns the order in which the vertices are processed in line 1 of Algorithm 2. The *degeneracy* of a graph is the smallest number d such that every subgraph of that graph contains a vertex of degree at most d. A degeneracy order can be obtained by repeatedly taking and removing a vertex with minimum degree from the remaining subgraph. Eppstein et al. (2010) showed that by considering the nodes in a degeneracy order reduces the runtime complexity of Algorithm 2 even further to time $O(dn3^{d/3})$.

ALGORITHM ENGINEERING

In this section we describe reduction rules to reduce the size of the product graph before we enumerate the cliques. While we cannot prove any theoretical improvements for the running time, they reduce the practical running time significantly as we will demonstrate in the next section.

The runtime complexity of finding c-cliques is known to be $O(dn3^{d/3})$, where n is the number of product nodes and d the degeneracy of the product graph. The upper limit on the degeneracy of a graph is the largest degree of its nodes.

Molecular graphs are connected and have no self-loops and their minimal node degree is one. Their maximal degree is limited by the element valence. Elements of the main group have at most valence seven, the most common elements in organic chemistry – C, H, N, O, P and S – at most six.

In the following, we also use the notation *size* of a graph for the number of its vertices. Given two molecular graphs of size m, the maximal size of the k-product graph is $n = m^2$. The maximal number of adjacent c-edges to any of its product nodes (u,v) is $\deg(u) \cdot \deg(v)$. The maximal number of adjacent d-edges is $(m - deg(u) - 1) \cdot (m - deg(v) - 1)$. The maximal number of d-edges dominates the maximal degree and thus the maximal degeneracy of a k-product graph.

In conclusion, any worthwhile attempts to reduce the running time of finding c-cliques should focus on reducing the numbers of nodes and unnecessary d-edges (and thus—potentially—the degeneracy) of the product graph.

Degree-1 neighbors

By definition, the k-product graph $G_1 \otimes G_2$ contains a node (u,v), if and only if the k-neighborhood subgraph $S_1^k(u)$ of G_1 is isomorphic to the k-neighborhood subgraph $S_2^k(v)$ of G_2.

Given $k > 0$, suppose u has neighbors $u' \in V_1 \cup N^k(u)$ with degree one. Then, all k-neighborhood subgraphs $S_1^k(u')$ are subgraphs of $S_1^k(u)$ (see Figure 4). Since $S_1^k(u)$ is isomorphic to $S_2^k(v)$, all $S_1^k(u')$ are isomorphic to all $S_2^k(v')$ with $t_2(v') = t_1(u')$. Therefore, for $k > 0$ the product graph contains for each nodes (u,v) with $\deg(u) > 1$ and $\deg(v) > 1$ all nodes corresponding to degree-1 neighbors (u',v') with $u' \in V_1 \cup N^k(u)$, $v' \in V_2 \cup N^k(v)$, $t_1(u') = t_2(v')$ and $\deg(u') = \deg(v') = 1$.

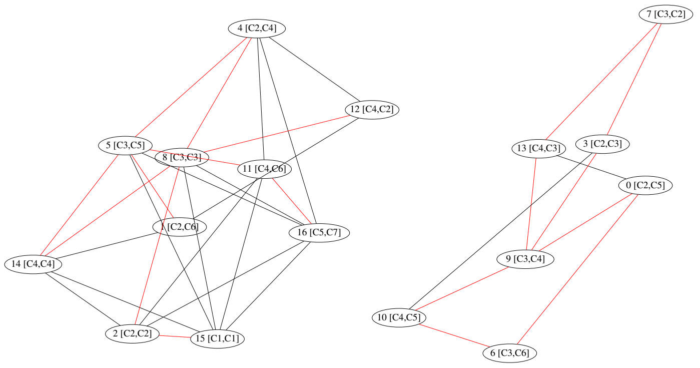


Figure 5. The product graph of the two graphs in Fig. 2. Product nodes corresponding to degree 1 nodes in the original graphs have been ignored. Also, c-edges are colored red and d-edges are colored black. We partitioned the product graph into two connected components by deleting the edges between all product node pairs (u,v) and (u',v') for which no path $((u,v),\dots,(u',v'))$ of c-edges exists.

Due to $\deg(u') = \deg(v') = 1$ and (u',v') and $u' \in V_1 \cup N^k(u)$, all such product nodes (u',v') are connected by c-edges to (u,v) and d-edges to all other nodes in the product graph. Thus, any maximal c-clique that contains a product node (u,v) of nodes u, v must contain all product nodes (u',v') of their respective degree-1 neighbors u', v' and *vice versa*, otherwise the clique would not be maximal.

Therefore we can reduce the size of the product graph by merging all such (u',v') with the node (u,v). A beneficial side effect is that removing product nodes corresponding to degree-1 neighbors also reduces a large number of d-edges, and thus, potentially, the degeneracy of the graph.

In practice, we iterate over all nodes u, v with descending degree. We build the product nodes (u,v) if $S_1^k(u)$ and $S_2^k(v)$ are isomorphic and mark all potential product nodes (u',v') corresponding to pairs of degree-1 neighbors of u and v to be skipped in future iterations.

This reduction rule is specific for our fragment definition, since it exploits the shell neighborhoods and can only be applied for $k > 0$.

Graph partitioning

We are interested in finding c-cliques. A c-clique is characterized by a spanning tree of c-edges over all nodes of the c-clique. Therefore, all pairs of nodes of the product graph $(u,v) \in V_{12}$ and (u',v') of the product graph which are not connected by a path of c-edges, can not be part of the same c-clique (see Figure 5).

Therefore, we can omit d-edges between all product node pairs (u,v) and (u',v') for which no path $((u,v),\dots,(u',v'))$ of c-edges exists. We exploit this property by adding c-edges first and partitioning the graph into connected components. Now each component contains a spanning tree of c-edges. We then add the d-edges for each connected component separately.

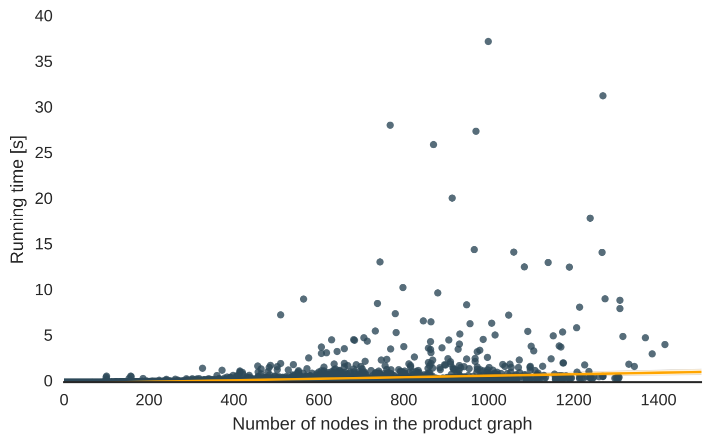
The runtime complexity of Algorithm 2 is determined by the number of nodes and the degeneracy. This rule helps by decreasing the number of edges, and thus, potentially, the degeneracy of the graph. But we also benefit from running the c-clique-finding algorithm separately on multiple smaller connected components. Note that this reduction rule is not specific for our fragment definition and can also be applied when enumerating all maximal common connected induced subgraphs.

In practice, we also usually impose a minimum number of core vertices $\kappa \in \mathbb{N}^+$ on our maximal common fragments. The size of a k-fragment equals the size of its corresponding c-clique, which is limited by the size of its enclosing connected component. If the size of a connected component is smaller than κ, then all of its c-cliques are smaller than κ. We can delete that connected component, further reducing the size of the product graph.

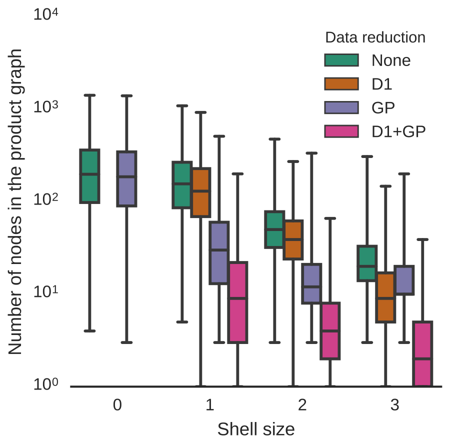
Note, if we applied the degree-1 reduction rule earlier, product nodes might correspond to several fragment nodes and we have to add the number of reduced nodes to the size of the connected component.

RUNTIME EVALUATION

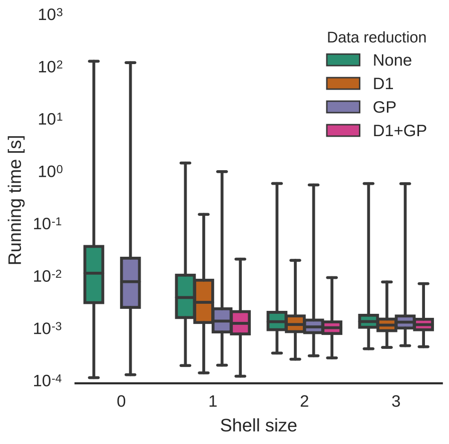
The runtime was evaluated on a random sample of molecular graphs. For this we used a snapshot of the ATB database (Malde et al., 2011) containing roughly 11.000 molecular graphs with nodeset sizes



(a) Running times without data reductions



(b) Product graph sizes



(c) Running times

Figure 6. **(a)** Running time as a function of product graph size for the *c*-clique computations without data reductions. The orange line is a linear fit, illustrating the linear growth exhibited by the instances for which the graph size dominates the degeneracy. **(b, c)** Overview of product graph sizes and running times for all shell sizes and methods: no data reduction (None), degree-1 rule (D1), graph partition rule (GP) and a combination of degree-1 and graph partition rules (D1+GP).

from three to sixty. From each size, we randomly select five molecules, resulting in 255 molecular graphs, which are approximately uniformly distributed in size.

We matched each molecule against every other molecule, resulting in 26983 molecule pairs. We used the shell sizes 0, 1, 2, and 3 as well as a minimal fragment size of 3. We applied four methods: no data reduction (None), degree-1 neighbor reduction (D1), graph partition reduction (GP) and a combination of the degree-1 neighbor and graph partition reductions (D1+GP). We repeated each combination of molecule, shell size and method five times, measuring the wall clock time for each repetition and calculating the mean running times. For each match and method we set a timeout of 600 seconds, resulting in two timed out instances.

Computations were performed in parallel on a 16 core cluster node with 64GB RAM memory. Memory usage did not exceed 30GB, even with 16 parallel computations.

Runtime Results

We established that the runtime complexity of computing *c*-cliques depends essentially linearly on the size and exponentially on the degeneracy of the product graph. The degeneracy itself is limited by the graph size. Figure 6a shows this behavior for the *c*-clique computations without data reductions. The majority of instances exhibit a slow linear growth. In a minority of instances the runtimes are dominated by their degeneracy and thus exhibit exponential growth.

Shell	Product graph size				Running time [s]				Speedup		
	None	D1	GP	D1+GP	None	D1	GP	D1+GP	D1	GP	D1+GP
0	260		247		0.091		0.076			1.5	
1	195	164	43	16	0.01	0.008	0.003	0.002	1.3	3.5	4.1
2	61	47	18	6	0.003	0.002	0.003	0.001	1.5	1.4	1.9
3	28	14	18	4	0.006	0.001	0.007	0.001	2.6	1.0	2.9

Table 1. Average number of nodes of the product graphs, average running times and average speedup for all shell sizes and methods: no data reduction (None), degree-1 rule (D1), graph partition rule (GP) and a combination of degree-1 and graph partition rules (D1+GP).

This runtime behavior is the result of using the degeneracy ordering and pivot rules for enumerating c-cliques, which would otherwise exhibit a full exponential growth in the product graph size.

Product graph sizes and running times are shown in Figure 6b, Figure 6c and Table 1. Note that the size of the product graph and the runtime decreases with increasing shell size. This offers a potential speedup of the MCCIS–E problem. The union of a maximal k-common fragment and its surrounding shell corresponds to a maximal common connected induced subgraph. Thus, by enumerating k-common fragments instead of MCCIS we can trade off runtime speed *vs* the fraction of MCCIS we potentially miss by choosing a large k.

The reduction rules help to further decrease the product graph sizes and runtimes. The largest effect can be observed for shell size one. Note that although the graph partitioning rule decreases the size more than the degree-1 rule, we also observe more runtime outliers for the graph partioning rule than for the degree-1 rule. The partitioning rule partitions the graph into connected components of c-edge spanning trees and deletes components smaller than the minimal fragment size, but does not alter the c-cliques. Thus it helps to reduce the graph size but not the degeneracy. On the other hand, the degree-1 rule reduces the sizes of the c-cliques, which helps to reduce the degeneracy and results in less runtime outliers. Combining both rules leads to the best performance.

CONCLUSIONS

We have introduced the problem of enumerating k-common fragments. We showed that it is NP-hard and proposed a number of algorithm engineering techniques that can make the problem more tractable. We showed that our approach is able to solve practical instances quickly. Our approach can also be used for enumerating maximal common connected induced subgraphs, where current exact approaches limits their use in large scale applications, for example, in screening large chemical libraries.

It is well-known that molecular graphs have special properties, for example, bounded node degree (limited by the atom valence) and low tree-width. It remains to be seen how these properties could be exploited for further speedup.

In this work, we have shown the benefits of our approach for all-against-all comparisons in large databases of small graphs. In the future, it might also be worthwhile to explore the suitability of our approach for large graphs, for example in finding local network alignments between large species-specific biological networks.

ACKNOWLEDGMENTS

The authors thank all members of the project *Enhancing Protein-Drug Binding Prediction* of the Netherlands eScience center for valuable discussions.

REFERENCES

Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577.

Canzar, S., El-Kebir, M., Pool, R., Elbassioni, K., Malde, A. K., Mark, A. E., Geerke, D. P., Stougie, L., and Klau, G. (2013). Charge group partitioning in biomolecular simulation. *Journal of Computational Biology*, 20(3):188–198.

Cazals, F. and Karande, C. (2008). A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3):564–568.

Droschinsky, A., Kriege, N., and Mutzel, P. (2017). Finding Largest Common Substructures of Molecules in Quadratic Time. In *SOFSEM 2017: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, pages 309–321. Springer, Cham.

Englert, P. and Kovács, P. (2015). Efficient Heuristics for Maximum Common Substructure Search. *Journal of Chemical Information and Modeling*, 55(5):941–955.

Eppstein, D., Löffler, M., and Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. In Cheong, O., Chwa, K.-Y., and Park, K., editors, *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin Heidelberg.

Foggia, P., Percannella, G., and Vento, M. (2014). Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(1).

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Koch, I. (2001). Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1):1–30.

Koch, I., Lengauer, T., and Wanke, E. (1996). An algorithm for finding maximal common subtopologies in a set of protein structures. *Journal of Computational Biology*, 3(2):289–306.

Levi, G. (1973). A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *CALCOLO*, 9(4):341–352.

Malde, A. K., Zuo, L., Breeze, M., Stroet, M., Poger, D., Nair, P. C., Oostenbrink, C., and Mark, A. E. (2011). An automated force field topology builder (ATB) and repository: version 1.0. *J. Chem. Theory Comput.*, 7(12):4026–4037.

McCreesh, C., Ndiaye, S. N., Prosser, P., and Solnon, C. (2016). Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems. In *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 350–368. Springer, Cham.

Moon, J. W. and Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28.

Oostenbrink, C., Villa, A., Mark, A. E., and Van Gunsteren, W. F. (2004). A biomolecular force field based on the free enthalpy of hydration and solvation: The gromos force-field parameter sets 53a5 and 53a6. *Journal of Computational Chemistry*, 25(13):1656–1676.

Rahman, S. A., Bashton, M., Holliday, G. L., Schrader, R., and Thornton, J. M. (2009). Small Molecule Subgraph Detector (SMSD) toolkit. *Journal of Cheminformatics*, 1(1):12.

Raymond, J. W. and Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533.

Schmid, N., Eichenberger, A. P., Choutko, A., Riniker, S., Winger, M., Mark, A. E., and van Gunsteren, W. F. (2011). Definition and testing of the GROMOS force-field versions 54A7 and 54B7. *Eur. Biophys. J.*, 40:843–856.

Tomita, E., Tanaka, A., and Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42.