

TDK dolgozat

Englert Péter



EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS
DEPARTMENT OF ALGORITHMS AND APPLICATIONS

Efficient maximum common subgraph search on molecular graphs

Advisors:

Tichler, Krisztián

Eötvös Loránd University
Department of Algorithms
and Applications

Kovács, Péter

ChemAxon Ltd.

Author:

Englert, Péter

Software development MSc

Budapest, 2013

Acknowledgements

First and foremost, I would like to thank my advisors for their hard work. I would like to thank Dr. Krisztián Tichler for his advice and constructive suggestions concerning both the content and the style of this paper. I would like to thank Péter Kovács for his guidance and help with research, and all the time he so generously sacrificed to help improve every aspect of this paper.

My grateful thanks are also extended to Dr. István Fekete, for introducing me to this topic and my advisors, and for his continuous support.

This paper was written in cooperation with ChemAxon Ltd., and I would like to thank the company for giving me the opportunity for this research. I would like to express my appreciation to the staff of ChemAxon Ltd. for their assistance, it was a privilege to work with them.



Contents

1	Introduction	6
2	Theoretical background	7
2.1	Applications	7
2.1.1	Similarity search	7
2.1.2	Clustering	8
2.1.3	Matched molecular pairs	8
2.1.4	Visualization	8
2.1.5	Reaction mapping	9
2.2	Problem specification	9
2.2.1	Definitions	10
2.2.2	Terminology	13
2.2.3	ΔY exchange	14
2.3	Algorithms	15
2.3.1	Max-clique based algorithms	16
2.3.2	Build-up algorithm	17
2.4	Screening	18
3	Improvements	20
3.1	Representation	20
3.2	Heuristics	22
3.2.1	Early termination	22
3.2.2	ECFP	22
3.2.3	Connectivity	24
3.3	Mapping	26
3.3.1	Keeping fragments together	27
3.3.2	Enumerating all mappings	29
3.3.3	Enumerating interesting mappings	30
3.3.4	Scoring mappings	31

4	Results	34
4.1	Memory usage	34
4.2	Accuracy and running time	36
5	Summary	40

1 Introduction

Chemical informatics, also called cheminformatics or chemoinformatics, is a relatively new and rapidly expanding interdisciplinary field. More and more methods in the field of chemistry rely heavily on information technology. As these often computationally intensive, *in silico* methods proliferate, and more and more chemical information is collected, the demand for effective solutions from information technology grows [1, 5].

One of the imposed challenges is the demand for efficient algorithms for a variety of problems. One such problem is the *maximum common substructure* (MCS) problem, which is usually formulated as finding the maximum common subgraph of two molecular graphs, and has numerous uses in the field of chemoinformatics.

The MCS problem is, however, a computationally complex problem, and its applications impose strict, often conflicting requirements. The running time of the algorithms should be low, but the results should be equal or at least very close to the actual maximum. Also, features that make sense for a chemist, but are hard to grasp formally are sometimes desired, such as keeping rings and functional groups together, not mixing fragments, or aligning rings and chains similarly in the input molecules.

At ChemAxon Ltd., we have developed and implemented MCS algorithms to satisfy these requirements. We have taken algorithms, methods and heuristics from the available literature, thoroughly measured and compared their performance, and improved upon them with our own ideas. This paper details our results, the methods we have found most effective, and our improvements.

2 Theoretical background

The maximum common substructure problem, and its applications in the field of chemoinformatics, are covered extensively by the existing body of literature. In this section, we provide a short overview of its applications (Section 2.1), the different ways to specify the problem (Section 2.2), and some of the available methods for solving it (Sections 2.3 and 2.4). We also explain the terminology used throughout this paper (Section 2.2.2), mention a way to generalize the problem to molecules with generic features and how this generalization affects existing methods (Section 2.2), and provide a detailed look at the ways of handling a recurring problem called the ΔY exchange (Section 2.2.3).

2.1 Applications

First, we take a closer look at some applications of maximum common substructure search, and their often conflicting requirements.

2.1.1 Similarity search

An important application of MCS search is similarity search, where a measure of the similarity of two molecules can be defined based on the size of their MCS [27, 19]. Methods of finding similar molecules in a database are indispensable early in the process of drug discovery, and having a variety of tools and measures is also important [23]. The structural similarity described by the MCS closely approximates chemical similarity, based on the similar property principle [12], which states that structurally similar compounds usually have similar properties.

When used for similarity search, an MCS algorithm is usually run on a whole database consisting of many molecules, comparing each one to a lead molecule. Since the algorithm is run many times, and on a variety of molecules, it is expected to be reasonably fast. It should also be resistant to combinatorial explosion, i.e. have no cases where it takes orders of magnitude more time to finish. Even though finding the exact MCS is usually not a requirement, finding a common substructure very close to it in size is, as to avoid false negatives.

The filtering methods described in Section 2.4 are especially helpful in this applica-

tion. Given a minimum similarity threshold, they can be used to quickly reduce the number of molecules the algorithm must be run on, and they never reject molecules which have an MCS based similarity higher than the given threshold, i.e. they do not cause false negatives.

2.1.2 Clustering

Clustering is another method that is widely used in drug discovery, and can benefit from MCS search [25, 8, 2]. It allows for a highly intuitive clustering of a compound library, both hierarchical and non-hierarchical. Every cluster has an associated substructure, which is present in all molecules in the cluster. This results in clusters whose characteristics are easy to see and interpret, as opposed to many hierarchical clustering methods, where common features in clusters can only be discerned by looking at the individual compounds in a cluster.

The requirements of this application are similar to those of similarity search. The algorithm potentially has to be run on all pairs of molecules in a compound library, though, as before, this can be reduced significantly using the filtering methods described in Section 2.4. An additional expectation is that the resulting substructures should be chemically relevant, for example, the substructures representing clusters are preferred not to contain broken rings.

2.1.3 Matched molecular pairs

MCS search can also be used to identify matched molecular pairs, which can be loosely described as pairs of molecules where the replacement of a small fragment or a functional group has occurred, and are frequently used in medicinal chemistry [22, 24, 18]. The requirements are similar to those of similarity search, with even more strict constraints on the accuracy of the algorithm.

2.1.4 Visualization

A different application of MCS search is visualization. Highlighting a common substructure of two molecules, and aligning them so that the common substructures are drawn the same way greatly helps in visually discerning why two molecules are similar, and at

which places do they differ. As this is usually done in an interactive environment, the algorithm should be fast enough not to make the user wait for longer than fractions of a second, or a few seconds at most. A more profound requirement comes from the fact that the size of the found common substructure is not the only thing that is important. We would like the parts of the molecules that are not part of the common substructure to also be aligned as similarly as possible, and in case of a disconnected common substructure, that the fragments of the common substructure have similar positions relative to each other in the two molecules. For example, if both molecules contain three equivalent rings, and in both molecules, one ring is connected to the two others, albeit with different chains; then we would like this middle ring in both molecules to represent the same ring in the common substructure, even though the MCS is three rings, and could be mapped to the two molecules in any way. This requirement is harder to describe precisely, and is elaborated upon in Section 3.3.

2.1.5 Reaction mapping

MCS search can also serve as a first step in automated reaction mapping [15]. Finding the common substructures of the reactant molecule(s) and the product molecule(s) can help identify the parts of the molecules which did not change during the reaction, and in this way, remove the noise and make the mapping of the reaction center easier. As in the case of visualization, the exact atom-atom correspondence between the two molecules is also very important, along with the size of the common substructure. In general, we would like atoms which are close to the reaction center in one molecule to be mapped to atoms which are close to the reaction center in the other molecule.

2.2 Problem specification

The problem of finding the maximum common substructure (MCS) of two molecules in chemistry applications must first be formulated as a mathematical problem so it can be analyzed and solved. It can be formulated either as finding a *maximum common induced subgraph* (MCIS), or finding the *maximum common edge subgraph* (MCES) of the graphs representing the molecules. In the first case, the problem is to find a graph containing as

many nodes as possible, which is an induced subgraph of both input graphs. In the second case, the problem is to find a graph containing as many edges as possible, which is a subgraph of both input molecules.

It can be argued that the MCES of two molecular graphs is more relevant in the field of chemoinformatics, as it is closer to the notion of chemical similarity of the two molecules [20]. Figure 1 shows an example of the difference between the two. Still, many state-of-the-art MCS algorithms solve the MCIS problem [3, 13]. Luckily, there’s a simple transformation that allows us to use these algorithms to solve the MCES problem. The molecular graphs are transformed into their line graphs, where the nodes represent the edges of the original graph. A MCIS of two line graphs is a MCES of the original graphs, except if a so-called ΔY exchange has occurred [26], which is discussed in Section 2.2.3.



Figure 1: An example of the difference between the MCIS and MCES of two graphs

Further distinction can be made between *connected* and *disconnected* MCIS or MCES problem. In the connected case, a common subgraph must be composed of a single component, while in the disconnected case, a common subgraph may consist of arbitrarily many components. In some solutions, the number of components is constrained, to limit the search space by discarding solutions composed of many components, which are arguably less relevant [3].

2.2.1 Definitions

The following definitions for MCS related terms are used.

Definition 2.1 (Molecular graph) *Molecular graphs – or molecule graphs – are simple, undirected, labeled graphs, where the nodes are atoms, the edges are bonds, and the labels of the atoms and the bonds are the atom and bond types, respectively. We denote the graph as $G = (V, E, L_V, L_E, l_V, l_E)$, where*

- V is the set of nodes (atoms),
- $E \subseteq V \times V$ is the set of edges (bonds),
- L_V is the set of node labels (atom types),
- L_E is the set of edge labels (bond types),
- $l_V : V \rightarrow L_V$ is the node labeling,
- $l_E : E \rightarrow L_E$ is the edge labeling.

This is sometimes referred to as a 2D molecular graph. There are also 3D molecular graphs, which are complete graphs where the nodes also represent atoms, but the edges are weighted and represent the geometric distance of the atoms [20]. As their usage is significantly more common, we focus only on 2D molecular graphs.

Representing molecules as graphs has many complexities on its own. For example, hydrogen atoms are usually not represented as nodes, as the number of hydrogen atoms bound to a heavier atom can be determined based on chemical knowledge. These are called implicit hydrogen atoms, however, molecular graphs may contain explicit hydrogen atoms too, as separate nodes. Another example is representing delocalized electrons. For example, an aromatic ring is usually drawn with alternating single and double bonds, however, it is arguably better to represent these bonds with an edge having a special, aromatic type.

Definition 2.2 (Subgraph) *A molecular graph G' is a subgraph of a molecular graph G , denoted as $G' \subseteq G$, if there exists an injective mapping f from the atoms of G' to the atoms of G with the following properties: each atom maps to an atom of the same type, and if two atoms u' and v' in G' have a bond connecting them, $f(u')$ and $f(v')$ in G also have a bond of the same type that connects them.*

The definition of an *induced subgraph* is similar, the only addition being that $f(u')$ and $f(v')$ in G' have a bond connecting them *if and only if* u' and v' in G also have a bond connecting them. Figure 2 shows an example of the difference between induced and non-induced subgraphs. G_1 is a subgraph of G_2 in every example, but not always an induced subgraph.

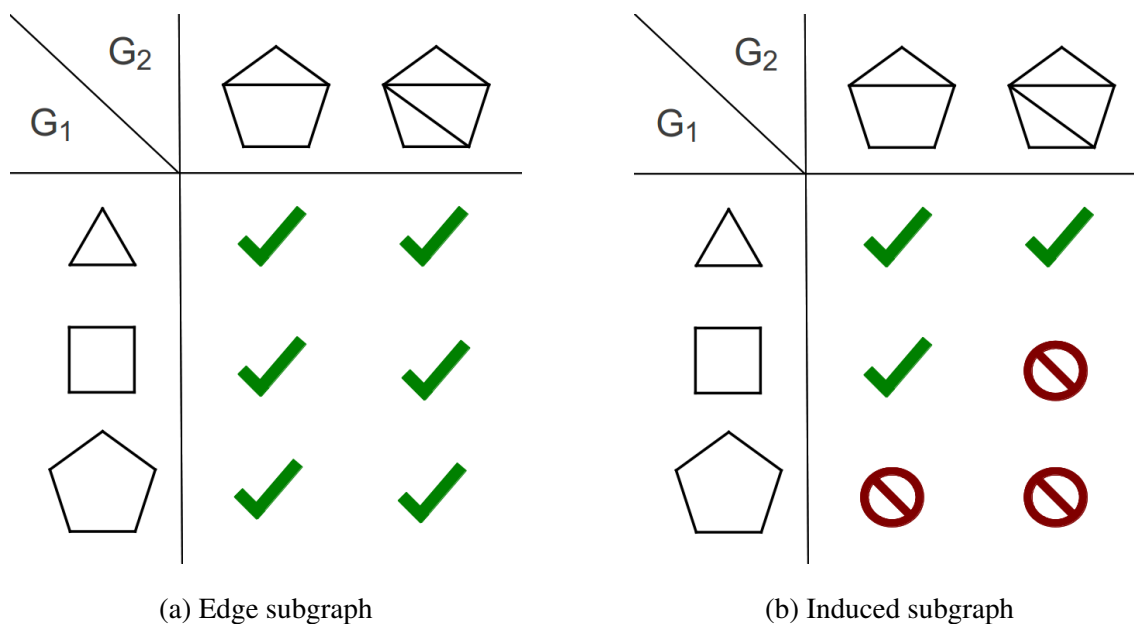


Figure 2: Difference between induced and non-induced subgraph

Deciding whether a graph is a subgraph of another graph is a classic NP-complete problem [9]; which also has many applications in the field of chemoinformatics.

Definition 2.3 (Common Subgraph) A molecular graph G' is a common subgraph of two molecular graphs G_1 and G_2 if $G' \subseteq G_1$ and $G' \subseteq G_2$.

Definition 2.4 (Maximum Common Edge Subgraph) A molecular graph is a maximum common edge subgraph of two molecular graphs if it is their common subgraph, and no other common subgraph contains more edges.

It is easy to see that the MCES problem, finding a maximum common edge subgraph, is NP-hard, or, if stated as a decision problem, NP-complete [9], since it is at least as general as deciding whether a graph is a subgraph of another graph. The reduction is based on the simple observation that a graph is a subgraph of another graph if and only if their maximum common subgraph is equal to the smaller graph.

Common induced subgraphs and maximum common induced subgraphs are defined similarly for induced subgraphs. Deciding whether a graph is an induced subgraph of another, and the more general MCIS problem as a decision problem, are also NP-complete.

In practice, we use slightly more complicated definitions in order to support what we call *query* features. These are, for example, generic atoms and bonds which do not have a specific type, but may represent atoms and bonds of different types. Examples include *halogen* atom, *metal* atom, *any* atom, as well as *single-or-double* bond and *any* bond. We can include these features in the definitions by assigning a non-empty set of labels to each node and edge, and allow mapping two nodes or two edges if the intersection of their label sets is non-empty.

Following the needs of applications at *ChemAxon*, we allow query features in one of the input molecules. Despite this generalization, we will use the simpler definitions throughout this paper, since the algorithms do not need to be changed substantially if one or the other definition is used. Upper bound calculation methods, however, have to be changed to preserve their correctness, see Section 2.4.

The input molecular graphs are referred to as the *query* and the *target*. Usually, this is done just to distinguish between the two input molecular graphs, but if query features are considered, we refer to the molecular graph that may contain query features as the query.

2.2.2 Terminology

From now on, MCS is used as a synonym for the MCES of two molecular graphs, a common substructure is taken to mean a common subgraph, all the algorithms discussed are assumed to run on line graphs to find common edge subgraphs of the original graphs, and the size of the result refers to the number of bonds in such a subgraph. To avoid confusion, if we are referring specifically to the original molecular graphs, we will often refer to nodes as atoms and edges as bonds. Since the nodes of the line graph represent the edges of the original graph, they may also be referred to as bonds.

A common substructure of a query and a target is also referred to as a *mapping* between the query and the target. The reason for this is that the implemented algorithms usually keep track of a common substructure as a selection of bonds from the query, and they associate a bond from the target, which represents the same bond in the common substructure, with each bond in the selection. This is essentially a mapping, a partial function from the bonds of the query to the bonds of the target. Adding a new bond to the common substructure can be described as adding a query-target bond pair to the mapping. When

discussing the MCIS problem, the mapping is composed of atom pairs instead of bond pairs.

An important observation is that multiple mappings may define equivalent common substructures. Section 3.3 elaborates upon the importance of a the found mapping, how it has important features apart from the common substructure it defines.

2.2.3 ΔY exchange

A ΔY exchange can happen if one of the graphs contains a Δ graph (K_3) and the other contains a Y graph ($K_{1,3}$). These graphs are not isomorphic, but their line graphs are, as shown in Figure 3. This becomes apparent only when projecting the mapping of the line graphs onto the original graphs, since it is impossible to create a consistent atom-atom mapping from an edge-edge mapping if a ΔY exchange has occurred.

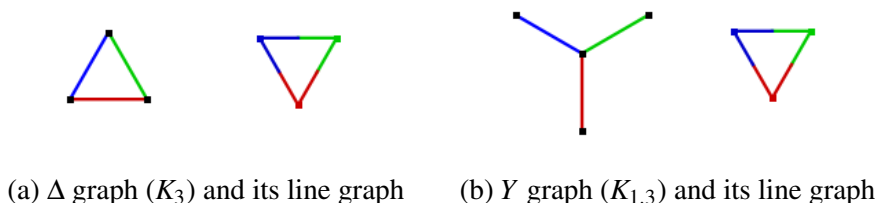


Figure 3: Delta-Y exchange

A ΔY exchange is rare when molecular graphs are concerned. The Δ graph is typically not a subgraph of molecular graphs, as it is not a very stable compound, and has high reactivity. For reference, only about 2.56% of structures in the PubChem Compound database contain the Δ graph as a subgraph [7].

Based on these, such an exchange can be handled in multiple ways. One way is to completely ignore it. This can be done only if the exact atom-atom mapping is not important, only the size of the MCS. This is the simplest approach, and on average, a ΔY exchange does not have much influence on the size of the result. However, in small molecules, even one bond can make a large difference in similarity, and the correctness of the algorithm also suffers, since it may return a result larger than the actual maximum, or a previously computed upper bound on it.

Another way is to continuously check if an exchange has occurred during search, as in [4, 17]. If the exchange is discovered, different actions can be taken depending on the

algorithm. For example, a backtrack can be initiated, the current common subgraph can be invalidated, or a different bond pair can be chosen instead of the one which caused the ΔY exchange to occur. In this case, the atom-atom mapping must be kept track of along with the bond-bond mapping during the search. This approach is easier to implement in some algorithms and harder in others, but performance usually suffers.

A third approach is in between the other two. The exchange is ignored during search, but afterwards, when the atom-atom mapping is calculated, a bond is removed from the common subgraph for each ΔY exchange that has occurred to create a consistent mapping. This is simple to implement, has negligible effect on performance, and produces a consistent result. However, it may produce suboptimal results even on some small inputs, and cannot be used in an exact MCS algorithm. An example can be seen in Figure 4. The bond removed after search is colored red. An algorithm using this approach would find no difference in size between the common subgraph shown in Figure 4a and Figure 4b, so it may return the one containing a ΔY exchange. In this case a bond, for example the one colored red, must be removed afterwards, and the final result is not optimal.

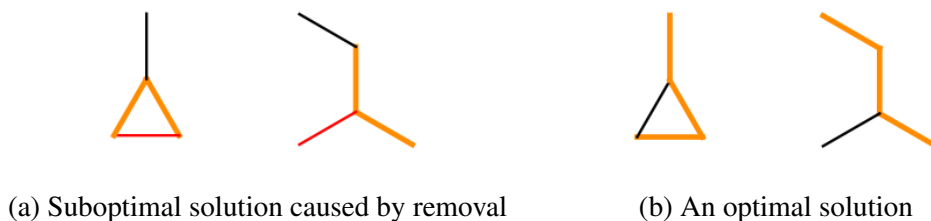


Figure 4: Disadvantage of detecting ΔY exchange after MCS search

2.3 Algorithms

There are many published approximate algorithms for MCS search. The most popular approaches include backtracking and reducing the problem to finding a maximum clique of the modular product graph, but various other approaches have also been published [20]. The first MCS search implementation at ChemAxon used backtracking, and the newly developed implementations are based on either the max-clique approach, or a recently published algorithm called the build-up method [13]. Sections 2.3.1 and 2.3.2 contain a summary of the last two methods, respectively.

2.3.1 Max-clique based algorithms

Many state-of-the-art algorithms, both exact [17, 4, 16] and approximate, use this approach due to the extensive literature and efficient algorithms available for clique detection. In their original form, these algorithms solve the MCIS problem. They build a modular product graph from the two molecular graphs, which contains a node for each pair of atoms, where the atoms are from different molecules, but have matching labels. Two nodes of the product graph are connected by an edge if and only if a mapping can contain both pairs at the same time.

Formally, given two molecular graphs G_1 and G_2 , their modular product graph $G = G_1 \times G_2$ is a simple, undirected graph, where the set of nodes is $V = \{(v_1, v_2) \in V_1 \times V_2 \mid l_{V_1}(v_1) = l_{V_2}(v_2)\}$, and given two nodes $u = (u_1, u_2), v = (v_1, v_2) \in V$, they have an edge $e = (u, v) \in E$ connecting them if and only if $u_1 \neq v_1, u_2 \neq v_2$, and one of the following holds:

- $(u_1, v_1) \notin E_1$ and $(u_2, v_2) \notin E_2$,
- $e_1 = (u_1, v_1) \in E_1, e_2 = (u_2, v_2) \in E_2$, and $l_{E_1}(e_1) = l_{E_2}(e_2)$.

The edges in the modular product graph represent the compatibility of atom pairs represented by the nodes the edge connects. Compatibility of two pairs of atoms means that the two pairs can be present in the same mapping (defining a common substructure), and general compatibility means that a set of atom pairs may all be present together in the same mapping. The equivalence of pairwise compatibility and general compatibility of a set of atom pairs follows from the definitions, as the definition of a subgraph only places constraints on atoms or pairs of atoms.

A clique in a modular product graph is therefore a set of pairwise compatible atom pairs, which defines a unique query-target mapping, and, consequently, a common induced subgraph. Similarly, a query-target mapping, representing a common substructure, defines a unique clique in the modular product graph, since it is a set of pairwise compatible atom pairs.

If we consider common induced subgraphs different if the atom-atom mappings they define are different, then every common induced subgraph determines, and is determined

by, a unique clique in the modular product graph. The MCIS problem is thus equivalent to finding a maximum clique in the modular product graph.

Using the transformation from MCIS to MCEs described in Section 2.2, finding an (approximate) MCEs can be reduced to finding an (approximate) maximum clique in the modular product of the line graphs of the input molecular graphs.

Figure 5 shows an example of how the product graph is built from two input molecular graphs, when solving the MCEs problem (the example is taken from [17]). The G_1 and G_2 are first transformed into their line graphs $L(G_1)$ and $L(G_2)$. The product graph, symbolized by the table on the right, contains a node for each line graph node pair, where one node is taken from $L(G_1)$ (a bond of G_1), the other node from $L(G_2)$ (a bond of G_2), and their labels are the same. The resulting product graph contains two maximal cliques. The first is composed of nodes $(3, 1')$ and $(2, 2')$, the second is composed of nodes $(2, 2')$, $(3, 3')$ and $(4, 4')$, so the maximum clique is the second one. This corresponds to the common substructure made up of edges 2, 3 and 4 in the upper graph and edges $2'$, $3'$ and $4'$ in the lower graph.

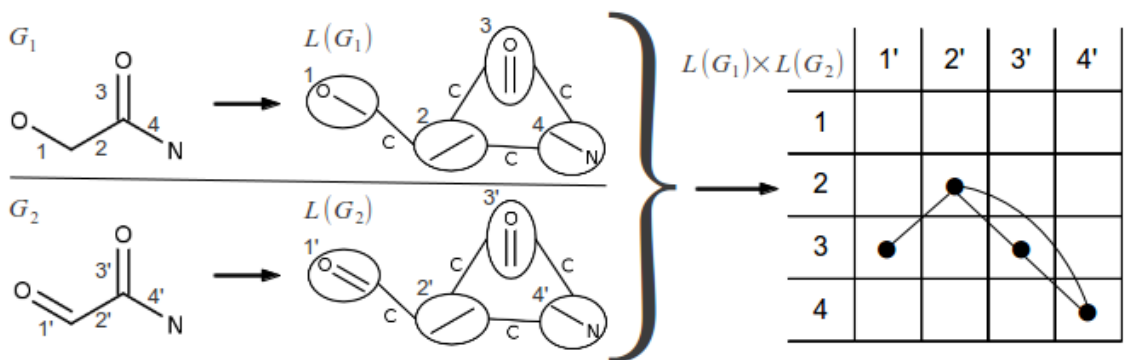


Figure 5: Modular product graph example

Our implementations of this approach are based on the approximate maximum clique search algorithm by Grosso, Locatelli and Pullan [10], with many modifications, most of which are detailed in Section 3.

2.3.2 Build-up algorithm

A recent paper by Kawabata [13] describes a greedy heuristic algorithm called the build-up method. The topologically constrained MCS and topology-based heuristic scores

described along with the algorithm are of special interest, since they have the potential to improve the quality of the mapping found, and give results which are chemically more relevant.

The original algorithm was developed for the MCIS problem, however, as noted in Section 2.2, it is easy to adapt such an algorithm to the MCES problem. We have implemented such a modified version of the algorithm. While initially underperforming the max-clique based implementations, after augmenting it with various heuristics described in Section 3.2, it turned out to be a viable alternative (based on the results shown in Section 4), which is able to compete in effectiveness with other approaches, both in terms of run time and the size of the results.

According to our experiments, for the considered purposes, the topological constraint and heuristics based on topology did not improve the quality of the mappings significantly. We abandoned them in favor of the methods described in Section 3.3.

2.4 Screening

An important aspect of MCS search is the existence of multiple upper bound calculation methods. These methods are generally much faster than a full MCS algorithm that results in a common subgraph, even if approximate MCS algorithms are used. They can be an effective tool for screening molecule pairs and filtering out those which are guaranteed to have a small MCS. This is very useful, for example, when executing a similarity search on a set of many compounds (Section 2.1.1). It is also very useful for accelerating the search for molecules with large common substructures during clustering (Section 2.1.2) or matched molecular pair search (Section 2.1.3).

We use the two-step screening procedure described by Raymond, Gardiner and Willett in [17], which consists of two upper bound calculation algorithms. The first one tries to match as many bonds of the input molecules as possible, based only on their types and the type of their endpoints, irrespective of their neighborhood. The second one performs a maximum weight matching of the atoms of the input molecules, where the weight of a pair is proportional to the number of neighboring bonds they have that can be matched. The second one is significantly slower than the first, but always gives an upper bound that is less than or equal to the bound given by the first algorithm. Hence, they can be

used in a two-step screening. The first algorithm is run on a pair of molecules, and if the similarity of the two molecules, based on the resulting upper bound, is larger than a given threshold, the second algorithm is run. If the resulting similarity score is still larger than the threshold, only then the actual MCS algorithm is run, otherwise, the pair is discarded.

The algorithms in [17] must be modified slightly to account for generic atoms or bonds that may match atoms or bonds of different types. During the first algorithm, bonds, along with their endpoints, are grouped by type in both molecules, and the minimum is summed for each type. If generic atoms or bonds are present, target bonds are grouped first, the type groups are defined according to them, and bonds of the query are included in each query type group which they may match.

In the second algorithm, atoms are grouped by type, and the maximum weight matching is calculated between the query and the target atoms of each type. Here, type groups should also be based on the target, and generic atoms in the query must be included in each type group which they might match. Alternatively, instead of calculating a matching for each type group, we can calculate one matching between all query and all target atoms, with zero (or negative) weight given to atom pairs that do not match. This alternative solution may give better results, since generic atoms are not included in multiple type groups, so they can only match one other atom. However, this approach may also be slower, since one large matching must be calculated instead of a few smaller ones.

In the second algorithm, calculating the score for a pair of atoms can be done similarly to the first algorithm. Their incident bonds can be grouped by target types, and the minimum of each type group summed up, with query bonds or bonds with query atoms included in multiple type groups. This results in an upper bound on the number of incident bonds that may match.

3 Improvements

This section details the methods that helped us improve upon the original algorithms the most. Some of them are new ideas (Sections 3.1 and 3.3), some of them are existing techniques from other problems, adapted to the MCS problem (Section 3.2.2), and some of them are already widely used.

3.1 Representation

One of our main improvements concerns the representation of the modular product graph in max-clique based algorithms. The size and density of this graph is one of the main disadvantages of the max-clique approach, as it impacts both memory usage and running time. However, we argue that the graph is so dense, that the complement graph should be stored instead, as an adjacency list.

Atoms in a molecular graph G have limited degree, and the average degree is very low. In a random selection of 10^6 molecules from the PubChem Compound database [6], the absolute maximum degree of an atom was found to be 6, the average maximum degree in a molecule was found to be approximately 3.308, and the average degree of an atom was found to be approximately 2.148. Let d denote the maximum degree of an atom in the input molecular graphs G_1 and G_2 . In the line graphs $L(G_1)$ and $L(G_2)$, the maximum degree of a node corresponding to the bond (u, v) is $2(d - 1)$, since both u and v may have $d - 1$ other bonds connected to them. Let the number of bonds in G_1 be m_1 and the number of bonds in G_2 be m_2 . The modular product graph has at most $m_1 m_2$ nodes. Two distinct nodes (e_1, e_2) and (f_1, f_2) of the modular product graph $L(G_1) \times L(G_2)$ (where e_1 and f_1 are bonds in G_1 , and e_2 and f_2 are bonds in G_2) are not adjacent if either:

1. $e_1 = f_1$;
2. $e_2 = f_2$;
3. e_1 and f_1 are adjacent, but e_2 and f_2 are not;
4. e_2 and f_2 are adjacent, but e_1 and f_1 are not;

5. e_1 and f_1 , as well as e_2 and f_2 are adjacent, but the common atoms have different type.

Let us define $N(e)$, where e is a node of a line graph, as the union of $\{e\}$ and the set of nodes adjacent to e . We may notice, that if (e_1, e_2) and (f_1, f_2) are not adjacent, at least one of $f_1 \in N(e_1)$ and $f_2 \in N(e_2)$ holds. Thus, the number of nodes that are not adjacent to (e_1, e_2) in the modular product graph is at most

$$\begin{aligned} |N(e_1)|m_2 + |N(e_2)|m_1 &= (\deg(e_1) + 1)m_2 + (\deg(e_2) + 1)m_1 \\ &\leq (2(d - 1) + 1)m_2 + (2(d - 1) + 1)m_1 = (2d - 1)(m_1 + m_2) = O(d(m_1 + m_2)). \end{aligned}$$

Considering that d is a small constant (its value is at most 4 for most molecules), the number of edges in the complement of the modular product graph is $O(m_1m_2(m_1 + m_2))$. In contrast, the number of edges the original product graph may contain is $O((m_1m_2)^2)$, which is why it is usually represented as an adjacency matrix. Assuming that $m = \max(m_1, m_2)$, this is a reduction in memory usage from $O(m^4)$ to $O(m^3)$, albeit with a larger constant.

This change can also significantly improve running time. The algorithms we studied in depth often access the product graph in one way only. When adding or removing a node from the current clique, they loop through the nodes which are not adjacent to that node, and update internal data structures. This is made a lot faster with the new representation, as it is not necessary to loop through a row in the adjacency matrix, the nodes that are not adjacent to a given node are stored in a list.

Another way in which the new representation improves running time is that the modular product graph can also be built in $O(m^3)$ instead of $O(m^4)$ time. This is done by only investigating the above mentioned five categories of bond pairs, which may form non-adjacent nodes, for each node. Based on our experiences profiling the implementations, the running time is often dominated by building the modular product graph, especially in case of large molecules, or when the algorithm terminates early because an optimal solution was found (see Section 3.2.1).

If querying whether two nodes are adjacent is required in an algorithm, it can be calculated on-the-fly after a little preprocessing. For both molecules, the adjacency matrix of the line graph should be calculated, with the common atom (or its type) stored in the matrix. Two distinct nodes (e_1, e_2) and (f_1, f_2) of the modular product graph are adjacent

if bonds e_1 and f_1 are not adjacent in the query molecule, and bonds e_2 and f_2 are not adjacent in the target molecule; or if both are adjacent and the common atoms match. The memory requirement of this solution is $O(m_1^2 + m_2^2)$, which is not significant in comparison to the memory requirements of storing the modular product graph, even when using the improved representation. The running time is slightly worse compared to the original adjacency matrix representation of the product graph. However, it is still $O(1)$, and as we noted, this operation is usually not needed. Edge properties of the product graph, for example the topological distance used in the original build-up method [13], can also be calculated using similar methods.

3.2 Heuristics

This section details the heuristics used to guide the algorithms in the search space.

3.2.1 Early termination

Early termination is a common heuristic in max-clique search algorithms. It involves terminating the algorithm if it can be guaranteed that an optimal solution has been found. In our case, we can provide such a guarantee by calculating an upper bound on the size of the MCS using the methods described in Section 2.4. The algorithms then check every time they find a better solution if its size is equal to the computed upper bound, and if yes, they terminate. We implemented this heuristic in our max-clique based algorithms, using the second upper bound calculation described in Section 2.4, with satisfactory results, described in Section 4.

3.2.2 ECFP

The main heuristic we developed for use in our algorithms is based on Extended Connectivity Fingerprints (ECFPs) [21], which already have many applications in similarity search. Based on our measurements, this heuristic is by far the most effective in helping the algorithm find a near-maximum clique early in the search. However, as the search progresses, we consider the score given by this heuristic less and less, to allow the algorithm to find a better solution if the heuristic leads to a local maximum.

The ECFP hash for an atom of a molecule is an integer hash code of its environment with a given radius. The r -radius environment of an atom is taken to mean the subgraph induced by atoms at most r distance from that atom. The ECFP hash of radius 0 is a hash value assigned to the atom, which is the same for atoms of the same type. A similar hash value is assigned for each bond too. The ECFP hash of radius $r + 1$ of an atom is created in the following way. For each adjacent atom, its ECFP hash of radius r is combined with the hash of the bond leading to it. These combined hashes are then hashed in an order-independent way to produce the ECFP hash of radius $r + 1$ of the atom. As a result, given two atoms, their environments of radius r are equivalent only if their ECFPs of radius r are the same.

The basic idea is to prefer matching atoms and bonds which have large equivalent environments. Similar molecules usually contain large substructures which are equivalent, and this heuristic is very effective in finding and matching such structures. We define the ECFP score of a pair of atoms u_1 and u_2 , denoted as s_{u_1, u_2} , as the largest r that for each $r' \leq r$, the ECFP hash values of radius r' of the two atoms are the same. The ECFP score of a node in the modular product graph would be the ECFP score of the corresponding atom pair if we were to implement an algorithm for the MCIS problem. However, when solving the MCES problem, the product graph is a product of the line graphs of the original molecules, and its nodes correspond to bond pairs, so to determine ECFP scores for these nodes, the ECFP scores of atom pairs must be combined somehow. For a node in the product graph which represents the bond pair (e_1, e_2) , where $e_1 = (u_1, v_1)$ is a bond of the query, and $e_2 = (u_2, v_2)$ is a bond of the target, we define its ECFP score as $\max(\min(s_{u_1, u_2}, s_{v_1, v_2}), \min(s_{u_1, v_2}, s_{v_1, u_2}))$. Since there might be two ways to pair the atoms of a bond pair to each other, we use the score of the better one. However, to determine the score of a bond pair if the pairing of their atoms is fixed, we use the worse score, since we want to know where the closest difference in the bonds' environments is. Essentially, this is the idea behind our definition for the ECFP score of a bond pair.

During search, we favor choosing nodes with higher ECFP scores. This often leads to finding a near-maximum clique early in the search. Of course, ECFP scores are not perfect, and as such, they can guide the search towards a local minimum. To avoid always choosing the same nodes and getting stuck in a local minimum due to misleading ECFP

scores, the ECFP scores of nodes chosen during search can be lowered between iterations in iterated search algorithms. We use this technique in our max-clique based implementations.

A drawback of using ECFP scores is that they perform poorly when query features are involved. If correctness for query features is desired, a hash assigned to a query atom should equal the hash assigned to each atom that it may match. Due to transitivity, this would mean the hash values of atoms with different types would have to be equal, and the ECFP heuristic would become pointless. Another solution is to store a set of hash values for each r -radius environment of each atom by calculating all possible hash values of the environment depending on the specific types a generic atom or bond in that environment may have. The problem with this approach is that the size of these sets can quickly grow too large to manage, and even in moderate cases (few generic atoms or bonds, small r), it would significantly complicate and slow down the calculation of ECFP scores.

A third approach is to assign hash values to query atoms and bonds that are different from the hash values of non-query atoms and bonds. This is simple to implement and quick, but the resulting scores can be misleading if the environment contains a generic query atom or bond. More specifically, the ECFP scores might be different even if the environments are isomorphic. Our implementations use this approach, as query features are present in only a small fraction of use cases, and even in these cases, if the ECFP scores lead the algorithm to a sub-optimal solution, subsequent iterations can correct this and improve the solution.

3.2.3 Connectivity

Another heuristic found to greatly enhance the speed and accuracy of the algorithms is the connectivity heuristic. This method has been mentioned many times in the literature [3]. The basic idea is to increase the size of the current common subgraph by first trying to add edges that are connected to it. This is natural when looking for a connected MCS, but even in the disconnected case, it greatly improves the results, and also decreases its fragmentation, i.e. the number of fragments (components) the found MCS consists of.

The idea, when translated to terms closer to the implementations, is to try to add a bond pair (e_1, e_2) to the current mapping for which there is at least one bond pair (f_1, f_2)

in the current mapping where e_1 and f_1 are adjacent. Since adding a bond pair to the current mapping requires it to be compatible with all pairs currently included, the adjacency of e_1 and f_1 implies the adjacency of e_2 and f_2 , so it does not need to be checked. In backtracking algorithms, this is usually straightforward to implement, and the build-up method can also be simply augmented with this heuristic.

In max-clique algorithms, the connectivity heuristic can be implemented by distinguishing D-edges and C-edges in the modular product graph [14]. C-edges connect nodes where the corresponding bond pairs are adjacent, both in the query and in the target. D-edges connect nodes where the corresponding bond pairs are not adjacent. This can be calculated on-the-fly as mentioned in Section 3.1, by storing the adjacency matrix of the input molecules. However, this forces us to loop through all other nodes when adding/removing a node, or when checking if a node can be added, both of which are very frequently used parts of code in most max-clique based algorithms. A better idea is to pre-compute for each node the list of nodes that are connected to it by a C-edge. This shortens the aforementioned loops, and does not require as many indirections, i.e. indexing arrays, as the previous solution. It also does not increase the memory usage significantly. Using the terms explained in Section 3.1, a node may have at most $(2(d-1))^2 = O(d^2)$ such neighbors, that is, each adjacent query bond paired with each adjacent target bond. The total time complexity of looping through all adjacent nodes connected by C-edges, assuming both molecule contain $O(m)$ bonds, is thus reduced from $O(m^2)$ to $O(d^2)$ where d is a small constant. The space complexity of this solution is $O(m^2 d^2)$, which is asymptotically smaller than the space complexity of the sparse representation described in Section 3.1.

A few additional choices have to be made when using the connectivity heuristic. The first one is how to consider the number of C-edges connecting a candidate node to the current clique, the *connectivity score*. This number can be given a priority ranging from ignoring it to always favoring nodes with the highest connectivity score. The second one is how to combine this heuristic with other heuristics. Based on our experiments, we found that it is beneficial to use this heuristic very aggressively, and always choose the next node from the nodes with the highest number of C-edges to the current clique. Even though this may lead to sub-optimal solutions, such as the one seen in Figure 6, we found it more effective to correct these afterwards by trying to swap not included nodes with

included ones, or using an iterated search.



Figure 6: Sub-optimal solution due to aggressive use of the connectivity heuristic

3.3 Mapping

This section details the methods developed to provide better mappings for important applications, such as reaction mapping and molecule alignment.

The original algorithms do not consider anything but the size of the found common substructure as a measure of its quality. However, the topological features of the result such the orientation of the common substructure in the query and the target are also important in certain applications.

Figure 7 shows an example where the query is an aromatic ring with a chlorine atom connected to it, the target is an aromatic ring of the same size with a bromine atom connected to it. The MCS is clearly the aromatic ring, however, a mapping that matches the atoms that are bonded with the halogen atom is often preferred.

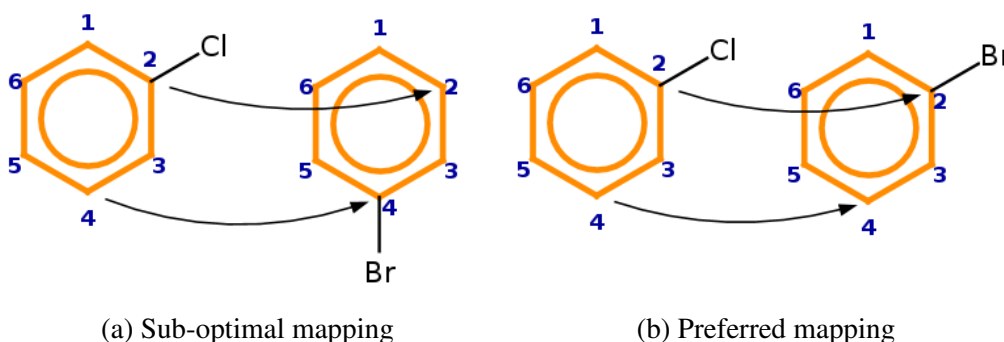


Figure 7: An example of a MCS with different mappings

Figure 8 shows an example of another situation where the same MCS has different mappings, and some mappings are preferred more than others. The atoms at the ends of the unmapped chains between the rings should be mapped to the atoms at the ends of the chain connecting the corresponding rings in the other molecule.

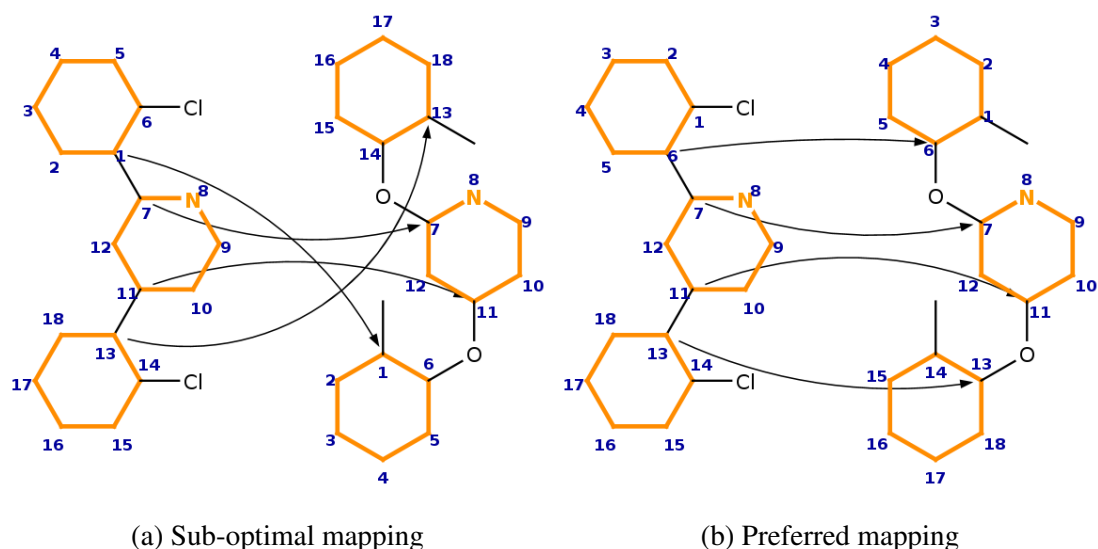


Figure 8: An example of a MCS with different mappings

Yet another example is shown in Figure 9. Let the query be a long chain of carbon atoms, and the target the same chain with a heteroatom somewhere in the middle. The MCS is composed of two shorter chains, however, if only the size of the MCS is taken into account, the chains might be flipped, which means that a carbon atom in the middle of the query chain is matched to an atom at the end of the target chain. In this case, a mapping that matches the atoms that are close to each other in one molecule to atoms that are close to each other in the other molecule is preferred.

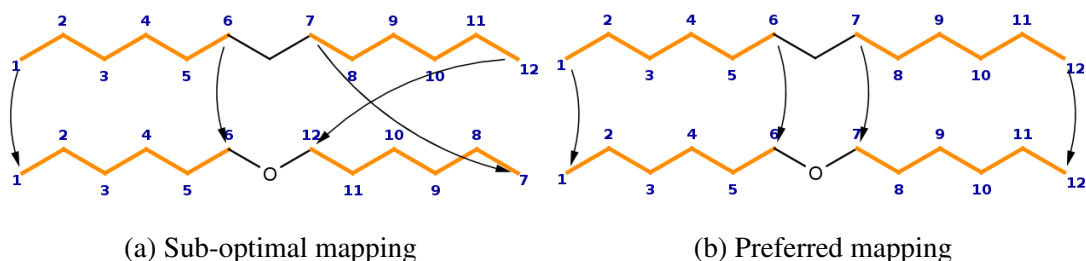


Figure 9: An example of a MCS with different mappings

3.3.1 Keeping fragments together

We start with a special case of these problems, namely the breaking of fragments. An example of such a situation is illustrated by Figure 10. Both the query and the target consists of two fragments, all of which are structurally very similar. They all contain

substructures R_1 and R_2 , but they all have a different substructure (R_3 , R_4 , R_5 or R_6) connecting them. The maximum common substructure consists of four fragments, two of both R_1 and R_2 . However, multiple mappings may define the MCS, and one where parts of one fragment of the query are not mapped to different fragments in the target is strongly preferred.

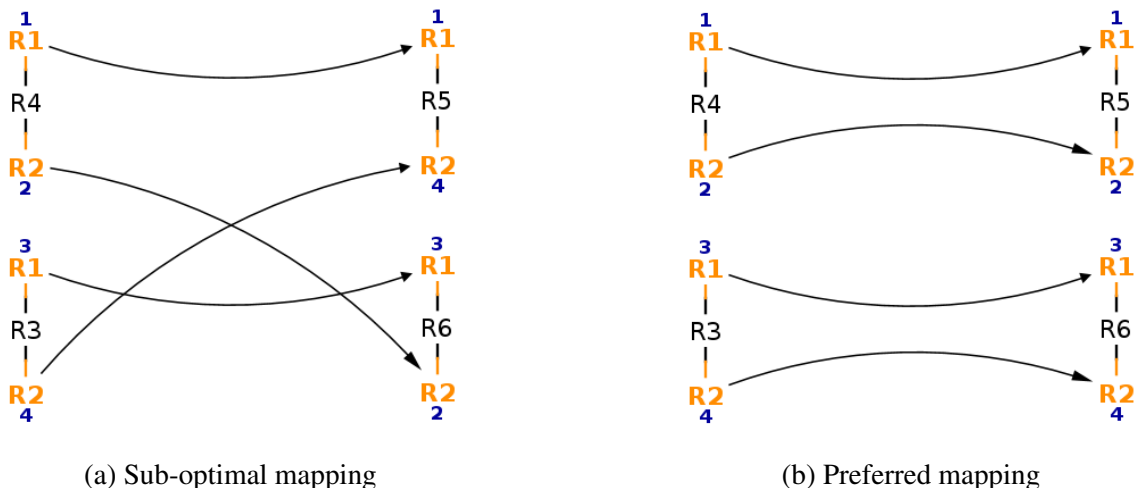


Figure 10: An example of a MCS with different mappings

To prevent mixing fragments too much, we store which fragment each bond in the query and the target belongs to. We say that a query-target bond pair associates a fragment of the query with a fragment of the target, and when adding the pair to a common substructure, we add this association to the set of associations the common substructure creates. When trying to add new bond pairs, we always prefer the ones which do not add a new association to this set.

This is a reasonable heuristic even if the mappings are not important, since fragments of the common substructure which are matched to parts of the same fragment both in the query and in the target may merge into one fragment during local search. Also, it can be implemented in a computationally efficient way, for example, by precomputing the fragments of the input molecules, and representing the set as a two-dimensional array of counters. An element $A[i, j]$ of this array represents the number of bond pairs in the mapping that associate fragment i of the query with fragment j of the target. Updating this array when a bond pair is added to or removed from the mapping, as well as querying whether a bond pair adds a new association ($A[i, j] = 0$) all take $O(1)$ time.

3.3.2 Enumerating all mappings

A general approach to providing more relevant mappings is to enumerate all mappings possible for the found common substructure, and let the users of the algorithm decide which one they prefer. This can be achieved by running a substructure search algorithm to find all possible mappings between the common substructure and the query, and between the common substructure and the target, and generating mappings between the query and the target by merging each pair of substructure-query and substructure-target mappings. More precisely, let G_c be the common subgraph, G_1 the query and G_2 the target. A substructure-query mapping is a pair of functions $f_{V_1} : V_c \rightarrow V_1$ and $f_{E_1} : E_c \rightarrow E_1$. A substructure-target mapping is similarly defined for G_2 . The atom and bond mapping from the query to the target that results from merging these two mappings is $f_V = f_{V_2} \circ f_{V_1}^{(-1)}$ and $f_E = f_{E_2} \circ f_{E_1}^{(-1)}$.

However, this would result in some mapping being produced multiple times. For example, if the query and the target are both Δ graphs (K_3), then the MCS is also a Δ graph. There are 6 substructure-query and 6 substructure-target mappings, which results in 36 mappings in total between the query and the target. However, the two triangles also have only 6 different mappings that define the same MCS, so each of them will be represented 6 times. These could be filtered, but it is more efficient to filter them before merging. This can be done by running one substructure search in a special, *order-insensitive* way. In such a search two mappings are considered different only if the set of bonds the substructure is mapped to are different. Note that if isolated atoms may be present in the substructure, two mappings should be considered different if the set of matched bonds *or* the set of matched atoms is different.

The problem with this approach is the number of mappings that might be produced. Even simple query-target pairs can have a MCS which has millions, or even orders of magnitude more, different mappings. For example, Figure 11 shows a situation where the query is a long chain, the target has many smaller chain fragments. The resulting MCS can be produced by $11! \cdot 2^{10} > 4 \cdot 10^{10}$ different mappings. This situation can be improved by allowing the user to request results from the algorithm one-by-one, so they can be evaluated and the whole process stopped when an adequate mapping is found. However, deciding if a mapping is adequate is often far from easy, and many unnecessary mappings

may still be enumerated.

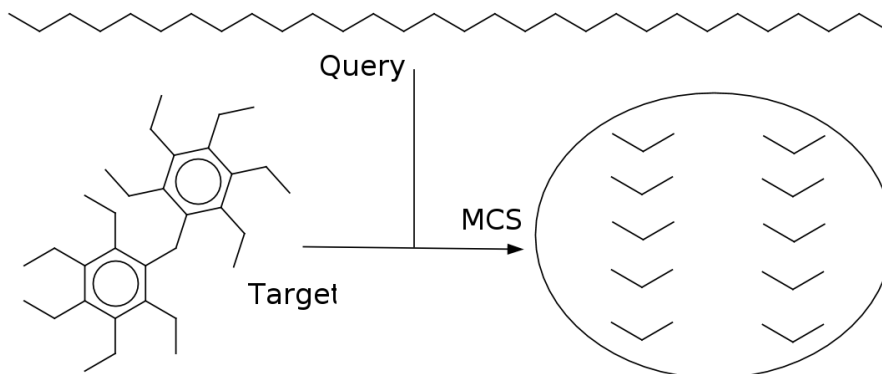


Figure 11: A query-target pair where the MCS has $11! \cdot 2^{10}$ different mappings

3.3.3 Enumerating interesting mappings

In all examples we investigated, where the mapping was found inadequate, the problem was with atoms at the edge of the substructure, i.e., the atoms that are incident to a non-matched bond in the query or the target. For example, if the found common substructure contains a ring that is a separate fragment in the query or the target, then the way it is mapped is irrelevant. However, if the same ring has attached chains in both molecules, then the mapping might be important, and, generally, we would like atoms that are incident to a non-matched bond in the query to be also adjacent to a non-matched bond in the target. The basic idea is to enumerate only mappings where these atoms are mapped differently. This can be done in multiple ways, but in all of them, an approximate MCS must be found first.

One way is to use substructure search. Based on a given common substructure, we call an atom in the query interesting if it is part of the substructure, but has an incident bond that is not part of the substructure. We define interesting atoms in the target similarly. Figure 12 shows an example, with the MCS highlighted in orange and the interesting atoms for the highlighted common substructure circled in red. We enumerate mappings by first fixing some interesting atoms in the query to map to certain interesting atoms in the target, running a substructure search algorithm, and returning the first result, if there is one. However, there can still be many ways to map interesting atoms in the query to interesting atoms in the target. If an enumeration with less results is desired, we can for

example try to choose fixed mappings so that each included interesting atom in the query maps to an interesting atom in the target that is in the same fragment of the substructure. This still allows us to rotate rings, flip chains or re-map more complicated symmetric structures, but results in less mappings.

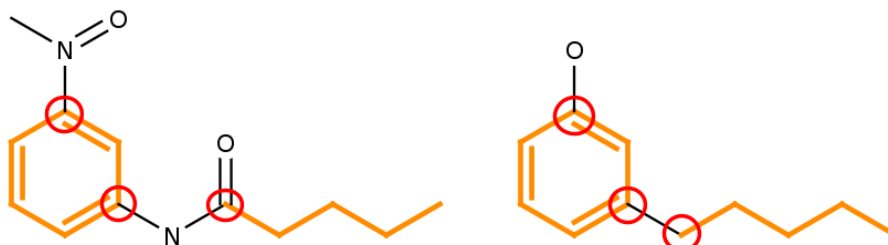


Figure 12: Query-target pair with interesting atoms circled in red

Another way is to bring the enumeration closer to the level of abstraction of the algorithm. We can define interesting bonds instead of interesting atoms, also based on a given common substructure. A bond in the query or the target is interesting if it has an adjacent bond that is not part of the substructure. We call a query-target bond pair interesting if both bonds are interesting. The enumeration is done by trying to add each such interesting bond pair to the mapping one by one, removing the fragments of the substructure to which any of them belonged, and using a local search to try and increase the size of the mapping back to, or higher than, its original value. There are still many choices left to make when implementing a similar method, for example, when to keep the resulting mapping, and when to keep the new mapping as the base for the next interesting bond pair that will be tried.

There are many other ways of implementing this reduced enumeration based on the same idea. The bonds or atoms next to unmapped bonds are interesting, and multiple ways to pair them should be tried.

3.3.4 Scoring mappings

Enumerating multiple mappings is often convenient, as it allows the users of the algorithm to select a mapping that is the most appropriate for their purposes. However, it also forces them to write a scoring function, deciding when a mapping is adequate, and the enumeration can still take a long time. This situation can be improved by using a small, reduced

enumeration after the algorithm, scoring the results with a general scoring function, and returning the best mapping found. Then, if the mapping is not adequate, more can be enumerated using substructure search. This allows us to provide a satisfactory first result in most use cases, and also preserves the ability for the user to control which mapping they prefer.

If more customization is desired, or a general scoring function cannot be agreed upon, it is also possible to allow the user to inject the scoring function, provide it as parameter to the algorithm.

Our max-clique based algorithm implementation uses the following approach. First, an approximate MCS is found as usual. Then, we try to add interesting bond pairs (nodes of the product graph), removing their fragments, and running a local search to increase the size of the common substructure. If a larger mapping is found, or if the size is the same as before, but the built-in scoring function returns a better score for the new mapping, it is kept, otherwise, it is reset to its previous state. We try to add each interesting node one-by-one in this way. Finally, the best mapping found is returned, but more results can also be requested, and more mappings will be enumerated and returned using substructure search.

We experimented with different scoring functions, and found the following to be adequate in most use cases. Let the set of query bonds be E_1 and the set of target bonds be E_2 . The mapping is $M \subseteq E_1 \times E_2$ (actually a bijective function between subsets of E_1 and E_2). Let us define the following functions:

- $deg_1 : E_1 \rightarrow \mathbb{N}$ and $deg_2 : E_2 \rightarrow \mathbb{N}$ assign the number of adjacent bonds to each bond in the query and the target, respectively;
- $dist_1 : E_1 \times E_1 \rightarrow \mathbb{N}$ and $dist_2 : E_2 \times E_2 \rightarrow \mathbb{N}$ assign the distance to each bond pair in the query and the target, respectively.

The scoring function is then:

$$\sum_{(e_1, e_2) \in M} |deg_1(e_1) - deg_2(e_2)| + \sum_{(e_1, e_2) \in M} \sum_{(f_1, f_2) \in M} (dist_1(e_1, f_1) - dist_2(e_2, f_2))^2$$

The lower this value is, the better we consider the mapping among mappings of the same size. The first part of the function sums the differences in degree for each query-target bond pair. This allows us to align chains or rings so that their attached parts which

are not in the common substructure are also aligned. A good example is matching two aromatic rings with different halogens attached, as seen in Figure 7. The second part is the sum of squares of the difference in distance in the query and in the target for each two bonds in the common substructure. This allows us to align fragments of the substructure that are in one fragment both in the query and in the target, so that the attachments connecting them are aligned. One example is shown in Figure 8, and another example is matching two chains with a little difference in the middle of the chain, so that the common substructure is composed of two chains, as seen in Figure 9.

4 Results

The different algorithms, heuristics and implementations were extensively benchmarked. This section contains a selection of measurements and results.

The algorithms were implemented in the Java programming language, using ChemAxon's Marvin Beans library for the input, output and representation of molecules. The benchmarking was conducted on a PC with an Intel Pentium Dual-Core T4200 CPU running a 32-bit Ubuntu 12.04 operating system. The algorithms were run in a separate Java Virtual Machine with a maximum heap size of 512 MB, unless stated otherwise.

The best implementations became part of the ChemAxon's JChem program suite from version 6.0 on.

4.1 Memory usage

To support our calculations regarding the decreased memory usage of the representation described in Section 3.1, the maximum memory usage of max-clique based algorithms using different modular product graph representations was measured. However, reliably measuring peak memory usage of an algorithm in a garbage collected environment is notoriously hard, so a practical approach was taken. The algorithms were run in a Java Virtual Machine with constrained maximum heap size, to see if they run out of heap space.

The input molecules used were graphene structures, i.e. many fused rings of length six (with a few attached atoms if needed to reach the desired atom count), consisting only of carbon atoms and single bonds, as shown in Figure 13. The reason for using these structures is that their graphs have high average degree, compared to most molecular graphs, and only contain one type of atom and bond, so the resulting product graph will be large compared to the size of the input molecular graphs.

The number of atoms in the input structures were multiples of ten, and the same molecule was used both as the query and as the target. The virtual machine was given a maximum heap size that is a power of two in megabytes, and the largest input molecule for which it still terminates without running out of heap space was determined. The main difference in the compared implementations was that one used an adjacency matrix to represent the modular product graph, and the other used the representation described in

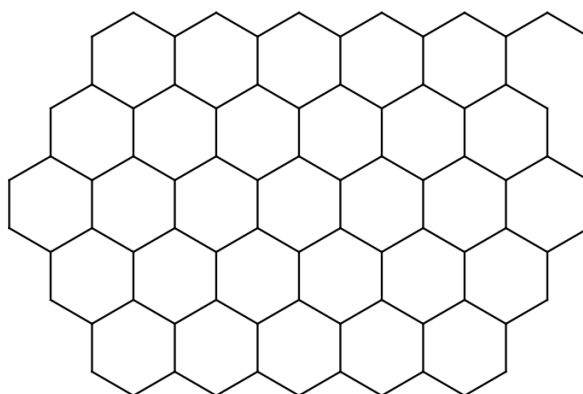


Figure 13: Graphene-like structure with 80 atoms

Section 3.1. The results are shown in Figure 14 as the number of atoms in the largest structure the algorithm could solve, as a function of the maximum heap size of the virtual machine.

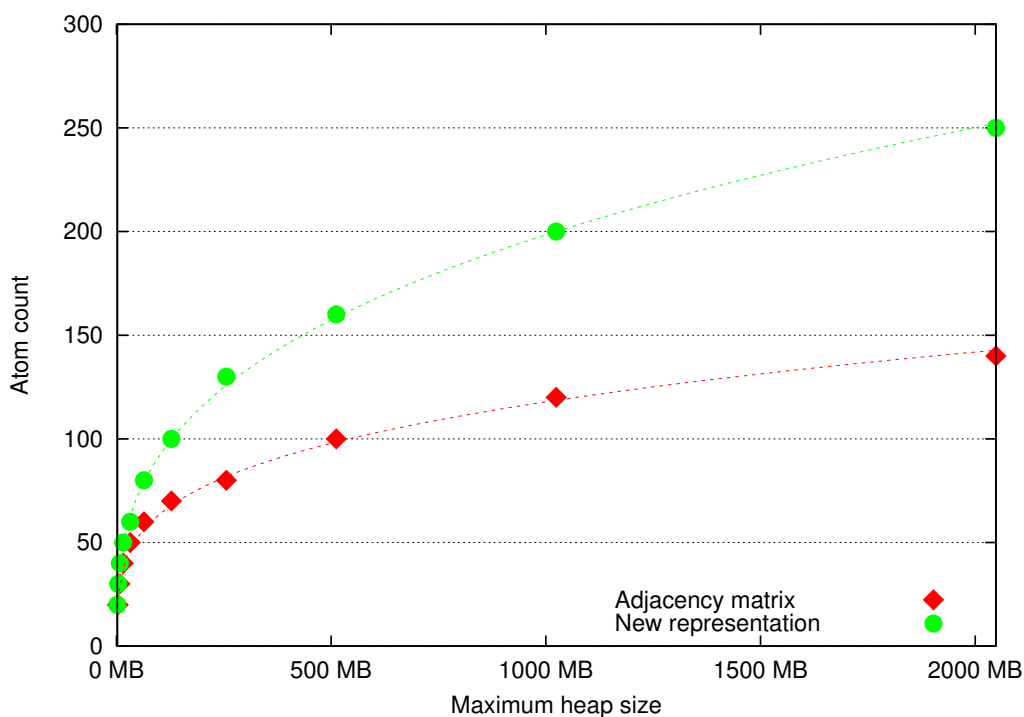


Figure 14: Memory usage of different product graph representations

In Section 3.1, we calculated the memory usage as a function of the bond count of the input graphs to be $O(m^4)$ for the adjacency matrix, and $O(m^3)$ for the new representation. As mentioned in Section 3.1, molecular graphs have bounded degree (and the average degree is very low). Let this bound be d . Let n denote the number of atoms in a molecular graph, and let m denote the number of bonds in the same graph. It is clear that $m \leq \frac{d}{2}n$, and

since d is a small constant, $m = O(n)$. The memory usage as a function of the atom count of the input graphs is thus calculated to be $O(n^4)$ for the adjacency matrix, and $O(n^3)$ for the new representation.

Using the results of the above mentioned measurements, we estimate the degree of the polynomial describing the actual memory usage in the following way. Functions of the form ax^b were fit to the data using the method of least squares. The resulting functions, $19.724x^{0.334}\frac{1}{MB}$ for the new and $18.525x^{0.268}\frac{1}{MB}$ for the adjacency matrix representation, are shown in Figure 14 as dashed curves. The empirical memory usage is the inverse of these fitted functions, roughly $0.133x^{2.992}KB$ for the new and $0.018x^{3.734}KB$ for the adjacency matrix representation. Thus, our empirical functions nicely support our calculations about the order of magnitude difference in memory usage between the two representations.

4.2 Accuracy and running time

The test data for the following benchmarks are from the public database of the National Cancer Institute [11]. Multiple test sets were created, each test set containing molecules similar to each other in characteristics such as size or ring count. The reason for this is that MCS search is usually run on similar molecules, or is run to find similar molecules, so an algorithm should be slightly more optimized for similar molecule pairs rather than dissimilar ones. To diversify the tests, however, different test sets contain molecules which have different characteristics.

For each test set, each molecule was paired with each molecule to produce the input for the algorithm, so if a set contains k molecules, k^2 inputs are created, k^2 MCS searches are performed. For each test set, we calculate the average running time and the average result size of these searches. The size of the result here is not simply the number of bonds, but the ratio of the number of bonds in the found common substructure and an upper bound on the number of bonds in the optimal MCS, calculated with the second method described in 2.4. This is done to normalize the sizes before averaging them, and so that the accuracy of the results becomes more clear when results from multiple test sets with molecules of different sizes are viewed together.

In this way, a ratio gives a bound on the error of both the upper bound calculation

and the MCS algorithm. For example, a ratio of 0.95 can mean that the result of the MCS algorithm is optimal, but the upper bound is $0.95^{-1} \approx 1.053$ times the optimal solution; it can mean that the upper bound is optimal, but the result of the MCS algorithm has 0.95 times as many bonds as an optimal solution; and it can also mean anything inbetween, for example that the number of bonds in an optimal solution is $0.95^{-\frac{1}{2}} \approx 1.026$ times the number of bonds in the found common substructure, but the upper bound is also $0.95^{-\frac{1}{2}} \approx 1.026$ times the size of the optimal solution.

Table 1 shows the results from running four different max-clique based implementations on the same eight test sets. The results are shown as the average size of the found common substructure, where the sizes are first normalized as described. The only difference between the algorithms is whether the ECFP and connectivity heuristics are used or not. It shows nicely how both heuristics improve the solutions, sometimes one improves it more, sometimes the other, and if both are used, the results are better than using any one of them alone.

Test set	Inputs	Atom count	Neither	ECFP	Connectivity	Both
1	1024	16 ± 4	0.973	0.973	0.973	0.973
2	1024	24	0.936	0.942	0.942	0.945
3	576	29 ± 2	0.941	0.974	0.969	0.977
4	576	34 ± 3	0.946	0.948	0.951	0.953
5	256	43 ± 3	0.884	0.891	0.897	0.898
6	256	75 ± 3	0.890	0.901	0.905	0.910
7	144	115 ± 8	0.866	0.883	0.887	0.907
8	16	234 ± 18	0.944	0.969	0.970	0.984

Table 1: Measurement of ECFP and connectivity heuristics

If early termination was turned off, there was no systematic, significant difference between their running times, so we conclude that using these heuristics does not have a significant negative impact on the running time of the algorithms. On the contrary, due to early termination, they can actually improve running times as well as accuracy.

Table 2 shows the average search running times of a max-clique based implementation (in seconds) on the same eight test sets, with early termination turned on and off. It shows

how the upper bound equals the actual MCS size often enough for early termination to have a significant effect on the running times.

Test set	Inputs	Atom count	No early term.	With early term.
1	1024	16 ± 4	0.008 s	0.005 s
2	1024	24	0.045 s	0.036 s
3	576	29 ± 2	0.087 s	0.054 s
4	576	34 ± 3	0.037 s	0.032 s
5	256	43 ± 3	0.068 s	0.071 s
6	256	75 ± 3	0.236 s	0.201 s
7	144	115 ± 8	0.846 s	0.826 s
8	16	234 ± 18	3.670 s	2.750 s

Table 2: Measurement of early termination

Table 3 shows a comparison of our final max-clique and build-up based implementations, on the same eight test sets as before. The column labeled *Fast* shows the results from running the max-clique based algorithm in fast mode, where the iteration count is much lower and mapping optimizations are turned off. On the last test set, this fast mode produces better results than the normal mode. This is atypical, but not impossible. Due to the small instance count of the last test set, such statistically rare events may easily occur.

Our result is that even though both are heavily optimized, our max-clique based implementation outperforms our implementation of the build-up algorithm. The max-clique based implementation finds solutions closer to the optimum, though on average, it has a larger running time on molecules up to a certain size. When run in fast mode, however, it is consistently faster than the build-up implementation, and still produces better results. An advantage of the build-up algorithm is that it is easier to implement.

The accuracy of the implementations was also measured by running them on isomorphic molecular graphs. Instead of running the algorithm on each pair of molecules from a test set, larger and more diverse test sets were created, the algorithms were run once for each molecule, and both the query and the target were the same molecule. Here, of course, the optimal solution was the entire molecule as a common substructure. The algorithms did not incorporate any special checks to identify if the two input molecules are actually

Test set	Result size			Running time		
	Build-up	Max-clique	Fast	Build-up	Max-clique	Fast
1	0.966	0.973	0.970	0.004 s	0.006 s	0.003 s
2	0.911	0.945	0.937	0.016 s	0.039 s	0.011 s
3	0.975	0.977	0.977	0.027 s	0.055 s	0.017 s
4	0.939	0.953	0.947	0.018 s	0.035 s	0.011 s
5	0.846	0.899	0.880	0.031 s	0.078 s	0.021 s
6	0.882	0.911	0.901	0.117 s	0.214 s	0.053 s
7	0.893	0.907	0.902	1.117 s	0.823 s	0.248 s
8	0.980	0.983	0.988	20.227 s	3.309 s	3.182 s

Table 3: Comparison of the build-up and max-clique implementations

isomorphic, and the representation of the target was changed first, the order of atoms and bonds was randomized. Both of these were done to ensure that the results are not distorted too much because the inputs are two equivalent molecules.

The result was that on eight different test sets containing molecules ranging from a few atoms to more than two hundred atoms in size, our max-clique based algorithm could always find that the MCS was the molecule itself. If either the ECFP or the connectivity heuristic was not used, this was not the case. However, it is not surprising that the combination of the two heuristics could recognize two isomorphic molecular graphs, as the ECFP heuristic gives maximum score to the correct bond pairs, and the connectivity heuristic helps not to ruin the solution by, for example, mixing disconnected bond pairs which are from different automorphisms.

5 Summary

To summarize, we have achieved substantial improvements in maximum common substructure (MCS) search.

The representation described in Section 3.1 greatly improves the speed, memory usage and scalability of max-clique based algorithms, which represent a significant fraction of state-of-the-art MCS algorithms.

Multiple heuristics for MCS search were also implemented, evaluated and compared (see Section 3.2). Extended-connectivity fingerprints were adapted for MCS search, which resulted in remarkable improvements both in the ratio of approximation, and, in conjunction with the early termination heuristic, the running time of the algorithms.

The problem of providing a chemically relevant mapping was also addressed. In Section 3.3, different solutions were explored. A general solution which does not result in a combinatorial explosion, but handles all the basic cases well, was proposed.

Besides providing a brief overview of the applications, their requirements, and the methods and solutions in the available literature, the different ways of handling ΔY exchange were collected and compared.

The results have been thoroughly tested and benchmarked, and the best implementations have been integrated into the products of ChemAxon Ltd., which are used by several leading international companies in the pharmaceutical industry.

References

- [1] Nathan Brown. Chemoinformatics – An introduction for computer scientists. *ACM Computing Surveys*, 41(2):8–38, 2009.
- [2] Alexander Böcker. Toward an improved clustering of large data sets using maximum common substructures and topological fingerprints. *Journal of Chemical Information and Modeling*, 48(11):2097–2107, 2008.
- [3] Yiqun Cao, Tao Jiang, and Thomas Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):366–374, 2008.
- [4] Paul J. Durand, Rohit Pasari, Johnnie W. Baker, and Chun-che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17):1–16, 1999.
- [5] Thomas Engel. Basic Overview of Chemoinformatics. *Journal of Chemical Information and Modeling*, 46(6):2267–2277, 2006.
- [6] National Center for Biotechnology Information. PubChem Compound. <http://pubchem.ncbi.nlm.nih.gov>, 2004. Accessed 2012 Apr.
- [7] National Center for Biotechnology Information. PubChem Compound Structure Search. <http://pubchem.ncbi.nlm.nih.gov/search/search.cgi>, 2004. Accessed 2013 May.
- [8] Eleanor J. Gardiner, Valerie J. Gillet, Peter Willett, and David A. Cosgrove. Representing clusters using a maximum common edge substructure algorithm applied to reduced graphs and molecular graphs. *Journal of Chemical Information and Modeling*, 47(2):354–366, 2007.
- [9] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

- [10] Andrea Grosso, Marco Locatelli, and Wayne Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6):587–612, 2008.
- [11] National Cancer Institute. Downloadable Structure Files of NCI Open Database Compounds. <http://cactus.nci.nih.gov/download/nci/>, 2000. Accessed 2012 Nov.
- [12] M.A. Johnson, G.M. Maggiora, and American Chemical Society. Meeting. *Concepts and applications of molecular similarity*. Wiley-Interscience Publication. Wiley, 1990.
- [13] Takeshi Kawabata. Build-up algorithm for atomic correspondence between chemical structures. *Journal of Chemical Information and Modeling*, 51(8):1775–1787, 2011.
- [14] Ina Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1-2):1–30, 2001.
- [15] James J. McGregor and Peter Willett. Use of a maximum common subgraph algorithm in the automatic identification of ostensible bond changes occurring in chemical reactions. *Journal of Chemical Information and Computer Sciences*, 21(3):137–140, 1981.
- [16] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316, 2002.
- [17] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. RASCAL: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45:631–644, 2002.
- [18] John W. Raymond, Ian A. Watson, and Abdelaziz Mahoui. Rationalizing lead optimization by associating quantitative relevance with molecular structure modification. *Journal of Chemical Information and Modeling*, 49(8):1952–1962, 2009.

- [19] John W. Raymond and Peter Willett. Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases. *Journal of Computer-Aided Molecular Design*, 16(1):59–71, 2002.
- [20] John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521–533, 2002.
- [21] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010.
- [22] Robert P. Sheridan. The most common chemical replacements in drug-like compounds. *Journal of Chemical Information and Computer Sciences*, 42(1):103–108, 2002.
- [23] Robert P. Sheridan and Simon K. Kearsley. Why do we need so many chemical similarity search methods? *Drug Discovery Today*, 7(17):903–911, 2002.
- [24] Noel T. Southall. Kinase patent space visualization using chemical replacements. *Journal of Medicinal Chemistry*, 49(6):2103–2109, 2006.
- [25] Martin Stahl and Harald Mauser. Database clustering with a combination of fingerprint and maximum common substructure methods. *Journal of Chemical Information and Modeling*, 45(3):542–548, 2005.
- [26] Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:150–168, 1932.
- [27] Peter Willett. Similarity-based approaches to virtual screening. *Biochemical Society transactions*, 31(3):603–606, 2003.