Graph Rewriting Based Search for Molecular Structures: Definitions, Algorithms, Hardness

Ernst Althaus, Andreas Hildebrandt, Domenico Mosca

Johannes Gutenberg-Universität Mainz, Staudingerweg 9, 55128 Mainz, Germany
[ernst.althaus,andreas.hildebrandt,mosca]@uni-mainz.de

Abstract. We define a graph rewriting system that is easily understandable by humans, but rich enough to allow very general queries to molecule databases. It is based on the substitution of a single node in a node- and edge-labeled graph by an arbitrary graph, explicitly assigning new endpoints to the edges incident to the replaced node. For these graph rewriting systems, we are interested in the subgraph-matching problem. We show that the problem is NP-complete, even on graphs that are stars. As a positive result, we give an algorithm which is polynomial if both rules and query graph, have bounded degree and bounded cut size. We demonstrate that molecular graphs of practically relevant molecules in drug discovery conform with this property. The algorithm is not a fixed-parameter algorithm. Indeed, we show that the problem is W[1]-hard on trees in the degree as the parameter.

1 Introduction

Small molecules are of crucial importance in molecular biology. They serve in various functions, e.g., as inhibitors or activators of proteins, as carriers of information, or energetic storage. Consequently, small molecules are a focus of numerous research fields (e.g., [4], [3], [2]). A prominent example is drug design, where small molecules are used to inhibit or activate proteins to achieve a desired biological function, or to prevent undesirable ones. Often, certain substructures of these small molecules are crucial for different aspects of their biological role: For example, a given molecular substructure might be favorable for an interaction with a given target, while another substructure may be responsible for toxic activity. Working with molecular substructures is hence an important task in computational chemistry, biology, and pharmacy. In fact, substructure representation is a core component of most applications in computational chemistry, including structure drawing, database search, or virtual screening [5].

Traditionally, substructures are modeled in chemical description languages such as Daylight's SMARTS[1]. These languages tend to be very complex and require significant training efforts before they can be routinely used. Furthermore, these languages are very restricted in their ability to describe patterns of typologies of the underlying graphs. This is relevant e.g. for searching certain cyclic peptides. To make matters worse, parsing and matching such patterns against databases of molecules is NP-complete. Furthermore, these description languages often suffer from the lack of a formal definition of their semantics.

[1] http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

To circumvent these problems, we propose a simple graph rewriting system (or graph transformation) to describe substructures: experience shows that it is more intuitive to use methods to describe graphs directly compared to a string based description. Even graph rewriting systems of very simple form allow a high expressive power which almost reaches that of SMARTS and allow to specify some interesting substructures beyond the expressive power of SMARTS. Although the related search problem remains NP-complete, it becomes polynomial if each minimal cut of the query graph has bounded size, which we empirically find to be true for most molecules contained in the standard databases.

Using graph grammars instead of established molecular description languages not only allows a more intuitive representation of graph topology, but also enables queries that cannot be realized within the framework of, e.g., SMARTS and would typically have to be formulated as an additional external filter. One such example of practical relevance is the search for cyclic peptides. Many important drugs have such a structure, since they tend to resist digestion quite effectively.

Molecular function is rooted in molecular interactions. Such interactions typically do not require contributions from all atoms of the molecule, but rather involve specific groups of atoms. Hence, molecules that share certain molecular subgraphs can be expected to share some of their interactions, and hence, of their function. We find that functional groups can be intuitively described by devising graph grammars that produce the molecular subgraphs belonging to this kind of group. Hence, matching such a graph grammar against a molecular database allows to query for instances that contain certain chemical groups and thus potentially show a desired chemical function.

A graph rewriting system is used to define a language of graphs, similarly to well-known string grammars. Therefore, graph rewriting systems are also known as ‘graph grammars’. A graph rewriting system consists of a set of rules defining how a graph can be transformed into another. The corresponding language is the set of all graphs that can be constructed from the rules starting with a graph consisting of a single node of a specific label. There are several known definitions of the transformation rules, such as node replacement, hyperedge replacement, and single- and double-pushout. We will relate our definition to those latter ones. Our definition is particularly simple and intuitive, as the graphs we are interested in are of bounded degree (atoms in molecules can form only a finite and typically very small number of covalent bonds) and thus we typically have rewriting systems of bounded degree. This allows a model where the rules explicitly reroute all edges that lose an endpoint due to the removal of a node of the graph. More precisely, we assume that the graphs are node- and edge-labeled multigraphs without self-loops. A rule replaces a single node with a specific label and specific labels of the incident edges by a subgraph. The edges originally incident to the removed node are given new endpoints in the new subgraph. The labels of these edges may not be changed.

The paper is structured as follows. In the following sections, we give the formal definitions and review some related work. Before we state our algorithm for graphs with bounded size of minimal cuts in Section 5, we show how the rules of our graph grammar can be transformed in some easier form. In Section 6, we show that the problem is NP-complete even on stars. If we use the degree as parameter, the problem is W[1]-hard even on trees, as shown in Section 7. For graphs of degree 5, the problem is even NP-complete, even if they have bounded pathwidth. The proof of this result is not given in this extended abstract. Finally, we give some experimental justification for the assumption that the graphs have bounded size of minimal cuts in Section 8 and conclude in Section 9.

2 Definitions and Summary of the Results

We now turn to the formal definitions. Let $\mathcal{L}^V$ and $\mathcal{L}^E$ be a set of node- and edge-labels. A node- and edge-labeled multi-graph over $(\mathcal{L}^V, \mathcal{L}^E)$ is a tuple $G = (V, E, N, L_V, L_E)$, where V and E are finite disjoint sets, $N : E \to V^{(2)}$ is assigning each edge its two endpoints, $L_V : V \to \mathcal{L}^V$ is assigning each node a label and $L_E : E \to \mathcal{L}^E$ assigns each edge a label. Notice that $V^{(2)}$ denotes the subsets of cardinality 2 of V. In the following, we always mean a node- and edge-labeled multi-graph when we talk about a graph.

For a graph $G = (V^G, E^G, N^G, L^G_V, L^G_E)$ and $U, U' \subseteq V^G$ with $U \cap U' = \emptyset$, let $\delta_G(U) = \{e \in E^G \mid \emptyset \neq U \cap N(e) \neq N(e)\}$, $\delta_G(U, U') = \{e \in E^G \mid U \cap N(e) \neq \emptyset$ and $U' \cap N(e) \neq \emptyset\}$ and let $G[U]$ be the induced subgraph of U. If the graph is clear from the context, we write $\delta(U)$ instead of $\delta_G(U)$ and furthermore, we write $\delta(v)$ for $\delta(\{v\})$. Let $\Delta(G)$ be the maximal degree of a node of G. A cut $(U, V \setminus U)$ is a partition of the nodes of a graph into two disjoint subsets, often referred to as the cut U. A cut U is minimal, if there is no cut U' such that $\delta(U')$ is a proper subset of $\delta(U)$. It is well known that in a connected graph, a cut U is minimal, if $G[U]$ and $G[V \setminus U]$ are connected.

A graph rewriting system in our definition is a tuple (S, P), where $S \in \mathcal{L}^V$ is a label and P is a finite set of replacement rules of the form $(L, L_1, \ldots, L_d) \to (G, n_1, \ldots, n_d)$, where $L \in \mathcal{L}^V$, $L_1, \ldots, L_d \in \mathcal{L}^E$, G is a graph and $n_1, \ldots, n_d$ are nodes of G. We call d the degree of the rule and G the replacement graph. For a graph rewriting system (S, P), let $\Delta(S, P)$ be the maximal degree of any rule in P. Such a rule allows to replace a node with label L whose incident edges have labels $L_1, \ldots, L_d$ by the graph G. The edges incident to the removed node are assigned new endpoints $n_1, \ldots, n_d$ in this order. More formally, given a graph $H = (V^H, E^H, N^H, L^H_V, L^H_E)$, a rule $(L, L_1, \ldots, L_d) \to ((V^G, E^G, N^G, L^G_V, L^G_E), n_1, \ldots, n_d)$ can be applied to a node $v \in V$, if there is a bijection $m : \{1, \ldots, d\} \to \delta_H(v)$ such that $L^E(m(i)) = L_i$ for $1 \leq i \leq d$. If we apply this rule using m at v, the resulting graph is $H' = (V^{H'}, E^{H'}, N^{H'}, L^{H'}_V, L^{H'}_E)$ with

- $V^{H'} = V^H \setminus \{v\} \cup V^G$, where the nodes of V^G are assumed to be disjoint from the nodes V^H (if they are not, we first make an isomorphic copy of G),
- $E^{H'} = E^H \setminus \{m(1), \ldots, m(d)\} \cup E^G \cup \{e_1, \ldots, e_d\}$, where $e_1, \ldots, e_d$ are new edges,
- $N^{H'}$ defined by $N^{H'}(e) = N^H(e)$, if $e \in E^{H'} \cap E^H$, $N^{H'}(e) = N^G(e)$ if $e \in E^G$ and $N^{H'}(e_i) = \{n_i\} \cup (N^H(m(i)) \setminus \{v\})$ for $1 \leq i \leq d$,
- $L^{H'}_V$ defined by $L^{H'}_V(v) = L^H_V(v)$ for $v \in V^{H'} \cap V^H$ and $L^{H'}_V(v) = L^G_V(v)$ for $v \in G$
- $L^{H'}_E$ defined by $L^{H'}_E(e) = L^H_E(e)$ for $e \in E^{H'} \cap E^H$, $L^{H'}_E(e) = L^G_E(e)$ for $e \in E^G$, $L^{H'}_E(e_i) = L^H_E(m(i))$ for $1 \leq i \leq d$.

In the definition, we do not distinguish terminal and non-terminal node-labels. Nevertheless, in our examples there are node-labels that do not appear in the query graph and only those appear on the left hand side of the replacement rules. Hence these labels are in principal the non-terminals and drown in boxes in the examples and all other labels are the terminals and drawn in circles. We refer to Figure 1 for a pictorial description. In the following, we will sometimes additionally require that the graph G in a replacement rule is connected.

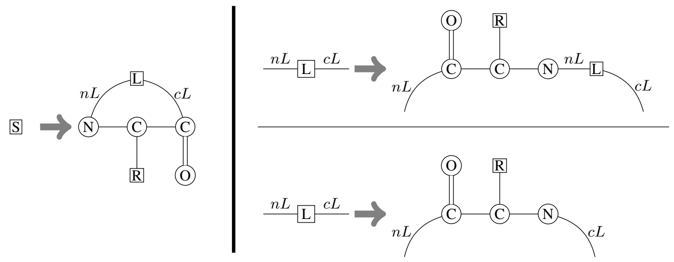

Fig. 1: A grammar for cyclic peptides: The starting label is S. The left-hand side shows the rule in which the starting symbol can be replaced by an initial graph, where the label L stands for an arbitrary chain of amino acids and R for a single amino acid side chain residue. With the upper rule, one can expand the chain by one further amino acid. After application of the lower rule, the chain can not be extended further. The edge labels are used to ensure that an N-node is always connected to a C-node and not to another N-node. An edge has label nL if it is between an N-node and an L-node on its creation and has label cL if it is between a C-node and an L-node. Whether there are rules that transform R into one of the amino acids or that transform R to a single node depends on whether it is necessary to describe that we have a peptide or not.

Let $\mathcal{G}(S, P)$ be the set of all graphs that can be obtained by iteratively applying rules starting from the graph consisting of a single node with label S and no edges. Given a graph G and a graph rewriting system (S, P), we are interested in the existence of a subgraph of G contained in $\mathcal{G}(S, P)$. Note that this query differs from the question typically addressed in the literature on graph rewriting systems, where one is typically interested in whether G is contained in $\mathcal{G}(S, P)$. Another related question would be the existence of an induced subgraph of G in $\mathcal{G}(S, P)$. Our algorithm can be generalized to these questions easily.

We further note that we will use the edge-labels only to restrict the applicability of the rules to have a richer expressive power, while our input graphs typically have node-labels only. If we are given a graph G without edge-labels, we are interested whether there exists a labeling of the edges, such that the resulting graph has a subgraph contained in $\mathcal{G}(S, P)$. Our algorithm can easily be extended to this question.

Finally, it is important to note that the complexity of graph rewriting algorithms is usually investigated under the assumption that the graph rewriting system is fixed (and hence a polynomial algorithm can exponentially depend on the description size of the system), whereas we assume that the graph rewriting system is part of the input.

Notice that if the degree of a replacement graph of a rule is larger than the maximum of $\Delta(S, P)$ and $\Delta(G)$ for a query graph G, the corresponding rule can not be used to derive a subgraph of G, as a node of the replacement graph with degree larger than this maximum can neither be a node of the query graph nor replaced by any rule of the rewriting system. Hence, we will assume in the following that the degree of any replacement graph is at most $\max(\Delta(S,P),\Delta(G))$ if we apply our algorithm for query graph G.

We will show that the problem is NP-complete, even if the pattern graph (and hence all graphs in the rules) is a star. Let $\zeta(G)$ be the maximal size of a minimal cut of G. In other words, $\zeta(G)$ is the maximal cardinality of a cut $U \subseteq V$ such that $G[U]$ and $G[V \setminus U]$ are connected. For a graph rewriting system (S,P) let $\zeta(S,P)$ be the maximum value $\zeta(G)$ for a replacement graph G. We give a polynomial time algorithm solving the subgraph matching problem for a graph G and a graph rewriting system (S,P) for the special case where $\Delta(G)$, $\zeta(G)$, $\Delta(S,P)$ and $\zeta(S,P)$ are bounded. Notice that $\zeta(T)=1$ for every tree T (and hence for every star). We sketch a proof for the fact that the problem is NP-complete even for graphs with bounded degree and bounded treewidth. Hence this well known parameter to describe the complexity of a graph can not be used for our problem.

Our algorithm is not a fixed-parameter algorithm in the degree and the maximal size of a minimal cut. We show the problem is W[1]-hard even on trees with the degree as the parameter and hence, there is no such algorithm unless W[1]=FPT.

Notice that the algorithm of Lautemann [7] gives an algorithm for the graph matching problem if the degree of the input graph is bounded without the need to additionally bound an other parameter. We show that the subgraph matching problem is NP-hard in this case and hence, it is unlikely that the algorithm can be generalized without increasing the running time. Nevertheless, our algorithm can be considered as a generalization of the algorithm of Lautemann.

Figure 2 gives a summary of our results.

3 Related Work

There are many chemical description languages that are currently in use, such as the Sybyl Line Notation (SLN) [1] or the SMILES Arbitrary Target Specification (SMARTS). All of those describe molecular structures in the form of strings. Using graph rewriting systems to describe structures is more direct and hence more intuitive.

Graph rewriting systems have been studied since the 1960s. Three main approaches to formalize the basic idea to successively replace graphs by other graphs to define classes of graphs have been proposed. We relate our definition to the most important other definitions of graph rewriting systems, i.e. algebraic methods (single- and double-pushout-method), node-replacement grammars and hyperedge replacement grammars. The following definitions are sometimes abbreviated where the full details are not needed for this work. We refer to [9] for details.

In the double-pushout method, a rule of a graph rewriting system is described by two graphs $L=(V^L,E^L,N^L,L_V^L,L_E^L)$ and $R=(V^R,E^R,N^R,L_V^R,L_E^R)$ with a common set of nodes K (formally, we need a mapping of the nodes K to the nodes V^L and V^R respectively). To apply a rule, we look for an isomorphic copy of L in G. We remove $L[V^L \setminus K]$ and all edges in $\delta_L(K)$. Then we insert $R[V^R \setminus K]$ and add the edges in

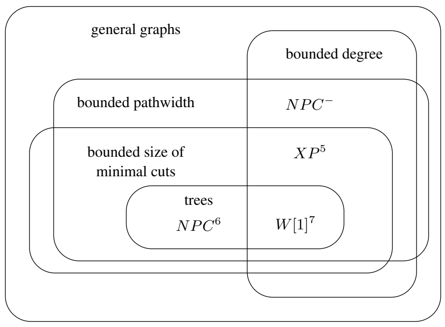

Fig. 2: Overview of our results for the subgraph matching problem. The algorithm which is polynomial-time if the size of the parameters are bounded (XP-algorithm), can clearly also be applied on more specialized classes, whereas the hardness results (NP-completeness (NPC) and W[1]-hardness (W[1])) generalize to larger graph classes. The superscript numbers the results indicate the section of the proof, the hardness-proof for graphs with fixed degree and pathwidth is not given in this extended abstract. All hardness results except 6 also hold for a fixed set of rules, whereas the algorithm can be used if the rules are part of the input. For the graph matching problem, the algorithm of Lautemann [7] is an XP-algorithm for all bounded degree graphs and the respecting NP-hardness proof does not carry over. The other two hardness proofs also hold for the graph matching problem.

$\delta_R(K)$ to the original nodes of K. Our rules can be interpreted as a restricted double-pushout method, in which L is a star, K are all nodes except the center of the star, and R is a connected graph. Furthermore each node of K has the same number of edges with the same labels in L and R.

In a node replacement grammar, a single node u with a given label in a graph G is replaced by a graph D, as in our rule. The difference is how the embedding into the former neighborhood is done. Whereas in our case, the edges are explicitly assigned new endpoints, the embedding in node replacement grammars is done purely by the node labels. More specifically, a set of tuples of node-labels (μ, δ) is given and edges added between each node of label μ in G without u and each node of label δ in D.

The definition which is in some sense closest to ours are hyperedge replacement grammars. A hypergraph is a graph where N is not necessarily a set of cardinality two, but of arbitrary cardinality. A rule of a hyperedge replacement grammar specifies a label of an hyperedge e which is to be replaced by a hypergraph whose nodes are a superset of the nodes $N(e)$ incident to e. Notice that due to the fact that we remove an edge (and no nodes), the embedding is quite simple.

For a hypergraph H its dual hypergraph is the graph which has a node for each hyperedge of H and a hyperedge for each node of H whose incident nodes are those corresponding to the hyperedges of H incident to the node. Our rules can be viewed as hyperedge replacements in the dual hypergraph with the additional condition that the degree of each node is exactly two (i.e. that the dual of the hypergraph is a graph).

Notice that we can consider our algorithm as an extension of the membership algorithm of Lautemann [7] to the subgraph matching problem. We will not elaborate on this due to space limitation.

4 Normal Form of the Rules

In order to simplify the description of the algorithm, we will assume next replacement graphs consists of exactly two nodes (and an arbitrary number of edges between them). If all replacement graphs have this property, we call the rules in *n*ormal form. In the following, we show that general replacement rules as defined above can always be reduced to a set of rules in normal form. The maximal degree of a rule with replacement graph G increases at most by $\zeta(G) \cdot \Delta(G)$. As discussed before, we can bound $\Delta(G)$ by the maximum of the degrees of the rules and the query graph.

First, we argue that we do not need rules in which the new graph has exactly one node. Such rules can only change the label of the node. We can remove such rules exactly as in the case of context-free (string) grammars when constructing the Chomsky normal form.

Let $(L, L_1, \ldots, L_d) \to ((V, E, N, L_V, L_E), n_1, \ldots, n_d)$ be a rule in general form with $V = \{v_1, \ldots, v_n\}$, $n \geq 3$ and $E = \{e_1, \ldots, e_m\}$. We assume that the edges are ordered such that there is $1 \leq i \leq d$ with $n_1, \ldots, n_i = v_1$ and $n_{i+1}, \ldots, n_d \neq v_1$. If we only consider rules with connected graphs, we assume that the nodes are numbered by descending depth-first-search (DFS) completion numbers and let $\{v_1, \ldots, v_n\}$ be the nodes in this order. This ensures that $G[\{v_2, \ldots, v_n\}]$ is connected. We show how we can replace this rule by one rule in normal form and one rule where the replacement graph has $n - 1$ nodes. Iterating this replacement, we can ensure that all rules have the normal form.

The idea is that the first rule generated the node v_1 and a node with a new label ℓ which will be replaced by the remainder of the graph later. The number of edges between v_1 and ℓ corresponds to the number of edges of the graph between v_1 and the remainder of the graph, i.e. $|\delta(v_1)|$. In order to ensure that all edges get the correct endpoints, we introduce new edge-labels of all these edges.

More formally, we introduce a new node-label ℓ and new edge-labels $\ell_{e_1}, \ldots, \ell_{e_n}$. In the first rule, the graph of the rule consists of the node v_1 with its label and a new node with label ℓ. This node ℓ will later be replaced by the graph $G[\{v_2, \ldots, v_n\}]$. For each edge in $e \in \delta_E(v_1)$, we create an edge between the two nodes with label ℓ_e. The new endpoint for the edges $e_1, \ldots, e_i$ is v_1 and for the edges $e_{i+1}, \ldots, e_d$ it is the node with label ℓ.

The second rule will replace a node with label ℓ whose incident edges have labels $(L_{i+1}, \ldots, L_d, (\ell_e)_{e \in \delta_G(v_1)})$ by the graph $G[\{v_2, \ldots, v_n\}]$. The new endpoints of the edges with labels $L_{i+1}, \ldots, L_d$ are those of the original rule and the endpoints of the edges with labels ℓ_e for $e \in \delta_G(v_1)$ are the endpoints different from v_1 in G. See Figure 3 for a pictorial description.

As the second rule will be the only rule for the node label ℓ, we can only apply the two rules in common and hence they exactly replace the original rule.

We now argue that we can bound the degree of all rules created by the sum of the original degree of the rule and $\Delta(G) \cdot \zeta(G)$ for G being the replacement graph. We distinguish between the contribution of the edges defining the original degree and additional the edges due to the replacement. The original edges were partitioned into the two created rules and hence their number can not increase in any rule. For the additional edges, we argue as follows. Any constructed rule either creates a node of G or a subgraph $G[\{v_i, \ldots, v_n\}]$. In the first case, the number of additional edges is bounded by the degree of the node. In the second case, the the nodes $\{v_i, \ldots, v_n\}$ are the nodes of a subtree of the DFS-tree that is cut at v_i, as we numbered the nodes according to descending DFS completion numbers and v_i is the root of the tree. We have to bound the number of edges of G with exactly one endpoint in $v_i, \ldots, v_n$. Consider now the DFS-tree as an undirected tree rooted at v_i. The nodes of each subtree of v_i that is not contained in $v_{i+1}, \ldots, v_n$ form a minimal cut and con contribute at most $\zeta(G)$ to the degree of the rule. Hence the degree of the rule is bounded by $\Delta(G) \cdot \zeta(G)$.

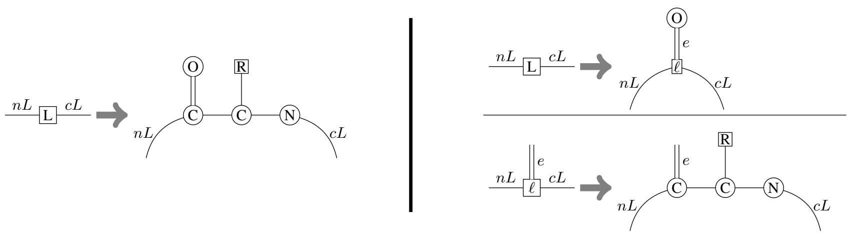

Fig. 3: The rule on the left-hand side is replaced by the two rules on the right-hand side. In the first rule, only the node with label O is created and a node with label ℓ is added as a placeholder for the remaining graph. The edge between these two nodes gets label e in order to ensure that it gets the correct endpoint in the end. In the second rule, the node with label ℓ is replaced by the remaining graph and the edges with labels nL, cL and e get the correct endpoints.

5 A Matching Algorithm

We start with an exponential-time algorithm that simply enumerates backward substitutions in all possible ways until either the starting label is found or no further backward substitutions are possible. We store all subgraphs we used to generate a label in a table in order to avoid multiple enumerations of the same structures. Afterwards, we characterize the graphs on which the algorithm runs in polynomial time. We assume that the rules are in normal form in the following.

If G can be obtained from the graph H by a single application of a rule, we say that H is a possible predecessor of G. Notice that if the rules and the input graph have bounded degree, given a graph G, we can simply enumerate all possible predecessors in polynomial time by iterating over all rules and trying to apply them backward on all pairs of nodes.

5.1 An Exponential Algorithm

Let (S, P) be a graph rewriting system and $G = (V, E, N, L_V, L_E)$ be a query graph. For $L \in L_V$ and $L_1, \ldots, L_d \in L_E$, let $G(L, L_1, \ldots, L_d)$ be the star with d edges with labels $L_1, \ldots, L_d$, whose center has label L (the labels of the other nodes are not specified).

Our exponential algorithm stores the following dynamic programming table:

$$DP(L, U, e_1, \ldots, e_d) \in \{\text{true}, \text{false}\},$$

where L is a node label, U is a subset of the nodes, d is the degree of a rule replacing node-label L, and $e_1, \ldots, e_d$ are edges of the graph with one endpoint in U, which is true, if it is possible to obtain a subgraph of $G[U]$ unioned with the edges $e_1, \ldots, e_d$ from $G(L, L_E(e_1), \ldots, L_E(e_d))$. Notice we have derived from $G(L, L_E(e_1), \ldots, L_E(e_d))$ and not from a subgraph of it. Furthermore, the labels of the endpoints of $e_1, \ldots, e_d$ that are not in U do not matter for this derivation. A subgraph of G is in $\mathcal{G}(S, P)$, if there is a table entry $DP(S, U, e_1, \ldots, e_d)$ set to true for the starting symbol S and arbitrary U and $e_1, \ldots, e_d$.

As our problem asks whether there is a subgraph of G in $\mathcal{G}(S, P)$, we only have to consider minimal subsets U, i.e. if $DP(L, U, e_1, \ldots, e_d)$ is true, then $DP(L, U', e_1, \ldots, e_d)$ should be true for each $U' \supseteq U$. We will not set these to true, as every entry of the table that can be set to true after some steps from $DP(L, U', e_1, \ldots, e_d)$ can also be set to true from $DP(L, U, e_1, \ldots, e_d)$.

In the beginning, we set $DP(L_V(v), \{v\}, e_1, \ldots, e_i) = \mathrm{true}$ for all nodes v, $1 \leq i \leq |\delta(v)|$, and $e_1, \ldots, e_i$ being a subset of cardinality i of the edges incident to v. All other values are set to false. Notice that intuitively, we select the subgraph (V', E') by deriving true for the table entry with node set V' and the empty set of edges from the entries of DP for the node set $\{v\}$ and the edge set $\delta(v) \cap E'$ for all $v \in V'$.

Then we iterate over all pairs of entries $(L^1, U^1, e^1_1, \ldots, e^1_{d^1})$ and $(L^2, U^2, e^2_1, \ldots, e^2_{d^2})$ labeled true with $U^1 \cap U^2 = \emptyset$ and try to mark further entries with true by computing all possible predecessors when we apply an arbitrary rule $(L, L_1, \ldots, L_d) \to (H, n_1, \ldots, n_d)$ with $V^H = \{u_1, u_2\}$ and $H = (\{u_1, u_2\}, E_H)$ of our grammar. Let $\bar{E} = \{e^1_1, \ldots e^1_{d^1}\} \cap \{e^2_1, \ldots, e^2_{d^2}\}$ and $E^i = \{e^i_1, \ldots, e^i_{d^i}\} \setminus \bar{E}$. We can label $(L, U^1 \cup U^2, e_1, \ldots, e_d)$ to true using such a rule, if

- the two nodes of H have labels L^1 and L^2, respectively. Let u_i be the node with label L^i $(i = 1, 2)$,
- there is a one-to-one correspondence $m : \bar{E} \to E_H$ of the edges in $\bar{E}$ and the edges of H respecting such that $L_E(e) = L_{E^H}(m(e))$ for all edges $e \in \bar{E}$,
- there is a one-to-one correspondence $m' : E^1 \cup E^2 \to \{1, \ldots, d\}$ between the edges $E^1 \cup E^2$ and the indices $1, \ldots, d$ such that for an edge e of E^i, its label is $L_{m'(e)}$ and $n_{m'(e)} = u_i$,

Notice that if all rules have bounded degree d, the running time of the algorithm is polynomial in the size of the table, which is $|L^V| \cdot 2^{|V|} \cdot |E|^d$.

5.2 Reducing the Number of Considered Subsets

In the following, we only consider rules whose graphs are connected graphs. In this case, the sets U always induce connected subsets of the graph.

Let $(L, U, e_1, \ldots, e_d)$ be an entry with value true in the dynamic programming table. Remove the edges $\delta(U) \setminus \{e_1, \ldots, e_d\}$ from G. If this graph has more than one connected component, we will not be able to apply a rule which contains nodes from different components. Hence, we can add all nodes not reachable from the component containing U to U. In the following, we restrict to entries $P(L, U, e_1, \ldots, e_d)$ such that U can not be enlarged in this manner.

If the input graph G is a tree, for each subset $\{e_1, \ldots, e_d\}$ of edges there is at most one set U of nodes to be considered, namely the maximal subtree containing $e_1, \ldots, e_d$ as edges leading to leaves, if such a tree exists (with the exception that we have two possible sets, if the set of edges has cardinality 1). Hence our algorithm runs in polynomial time on trees if the degrees of the rules are bounded. In the next section, we show that the problem is NP-complete if the rules and the query graph can have arbitrary degree.

More generally, we can characterize a set of nodes U by the edges of its cut $\delta(U)$ (again, each set of edges can characterize two sets of nodes - the two sides of the cut). As we are only interested in subsets that can not be enlarged, we can characterize subset U uniquely by the edges $\{e_1, \ldots, e_d\}$ plus the edges on $\delta(U)$ that are reachable in $G[V \setminus U]$ from one of the endpoints of $\{e_1, \ldots, e_d\}$ that are not in U, where we only need to report one edge for parallel edges.

We now argue that we only have a polynomial number of subsets U to be considered if $\varDelta(G) \cdot \zeta(G)$ are bounded by constants by showing that at most $\zeta(G) \cdot \varDelta(G)$ edges are needed to describe the cut. Assume that we need more than $\zeta(G) \cdot \varDelta(G)$ edges to describe the set U. This means that for at least one edge e_i, we have more than $\zeta(G)$ edges with different endpoints in $V \setminus U$ that are reachable from the endpoint u of e_i not in U in the graph $G[V \setminus U]$. Take the set U' of nodes reachable from u in $G[V \setminus U]$ as one set and U as the other set. Both are connected sets and we have $\delta(U, U') > \zeta(G)$, a contradiction.

6 NP-completeness for Rules of Unbounded Degree on Stars

In this section, we show that the graph and subgraph matching problem become NP-complete, if we do not bound the degree of the rules and the query graph, even if the input graphs are stars. A star is a tree with only one internal node, called the center of the star.

The proof uses a reduction of 1in3SAT to our problem. Recall that in the 1in3SAT problem, we are given a formula $\phi = c_1 \wedge \cdots \wedge c_m$ in conjunctive normal form such that each clause c_i consists of three literals, i.e. $c_i = l_1^i \vee l_2^i \vee l_3^i$ over a set $\{x_1, \ldots, x_n\}$ of variables, i.e $l_j^i = x_k$ or $l_j^i = \neg x_k$. We are interested whether there is a truth-assignment of the variables such that exactly one literal is true for each clause. This problem was proven to be NP-complete by Schäfer [10] and is already listed by Garey and Johnson [6] as NP-complete problem LO4.

Let $\phi = c_1 \wedge \cdots \wedge c_m$ with $c_i = l_1^i \vee l_2^i \vee l_3^i$ be a formula over the variables $\{x_1, \ldots, x_n\}$. The set of node-labels will be $\{c_1, \ldots, c_m\} \cup \{l_j^i \mid 0 \leq i \leq n \text{ and } 0 \leq j \leq m\}$ and all edges will have the same label l_e. The query graph is the m-star whose center has label l_m^n and the other nodes have labels $c_1, \ldots, c_m$. The starting label is l_0^0 (see Figure 5 for an example of the construction). The intuition is that the label l_0^0 can be replaced by two rules, one corresponding to setting x_1 to true, the other to setting x_1 to false. Applying the rule corresponding to setting $x_1 = \mathrm{true}$ will add the neighbors c_i

to the center for all clauses having literal x_1. We will change the label of the center to the label l^1_j, where j is used to count the number of edges incident to the center, i.e. the number of clauses that are satisfied by the truth-assignment in the first step. Such a label can be replaced by the rule corresponding to setting x_1 to a truth value. Hence the upper index of the label of the center indicates which variables already have a truth-value. The lower index is used to count the number of edges we already added to the star, i.e. if there are j clauses having literal x_i, the new label of the center is l^1_j. If we derive the l^n_m, we created a star with m incident edges. This graph matches the query graph if each label c_i was created exactly once and hence, for each clause exactly one literal was true.

Formally, we define the rules as follows. Let C^i_t be the indices of all clauses with literal x_i and C^i_f the indices of all clauses with literal $\neg x_i$. For each $0 \le i < n$ and each $0 \le j \le m$, we have the following two rules:

– $(l^i_j, \underbrace{l_e, \dots, l_e}_{j\text{-times}}) \to (G, \underbrace{c, \dots, c}_{j\text{-times}})$, where G is the graph with center c having label $l^{i+1}_{j+|C^i_t|}$ and $|C^i_t|$ further nodes, one with label c_k for each $k \in C^i_t$. We call this rule $r^{i,t}_j$.
– $(l^i_j, \underbrace{l_e, \dots, l_e}_{j\text{-times}}) \to (G, \underbrace{c, \dots, c}_{j\text{-times}})$, where G is the graph with center c having label $l^{i+1}_{j+|C^i_f|}$ and $|C^i_f|$ further nodes, one with label c_k for each $k \in C^i_f$. We call this rule $r^{i,f}_j$.

The construction can clearly be performed in polynomial time. We now argue that the 1in3SAT-problem has a solution if and only if there is a subgraph of the input star that can be constructed by the graph rewriting system.

We first show that if the 1in3SAT problem has a solution then our problem has one. For this, let $\pi : \{x_1, \dots, x_n\} \to \{\text{true}, \text{false}\}$ be an assignment such that each clause has exactly one literal that is satisfied. In the following, we will identify $\mathtt{true}$ with t and false with f. Starting with l^0_0, we use the rules $r^{0,\pi(x_1)}_0, r^{1,\pi(x_2)}_{|C^i_{\pi(x_1)}|}, \dots, r^{n-1,\pi(x_n)}_{\sum_{i=1}^{n-1} |C^i_{\pi(x_i))}|}$ and get the input star at the end.

On the other hand, assume that a subgraph of the star can be constructed from l^0_0 using the rules. Notice that from the construction, it is clear that we always have a star whose center has a label l^i_j and only this node (and its incident edges) can be replaced. Hence, we will use exactly n rules, one for each variable and we can interpret the rules as a truth-assignment. Furthermore, the lower index of the label of the center will always be equal to the number of edges of the star. As the final label has to be l^n_m, we will have constructed the complete input star at the end (and not just a subgraph of it) and the truth-assignment is such that each clause has exactly one literal that is satisfied.

Notice that we use the lower indices of the labels l^i_j to count the number of nodes that are created to eventually ensure that the complete star is constructed. Hence, this construction can be used for the subgraph matching problem as well as for the graph matching problem. For the graph machting problem, the lower indices can be dropped.

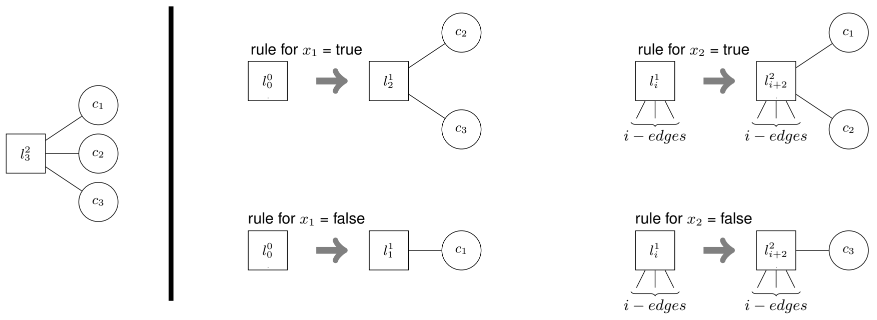

Fig. 4: The construction for the 2-SAT formula $c_1 \wedge c_2 \wedge c_3$ with $c_1 \equiv \neg x_1 \vee x_2$, $c_2 \equiv x_1 \vee x_2$ and $c_3 \equiv x_1 \vee \neg x_2$. The query graph is depicted on the left, the rules on the right hand side. With the rule corresponding to setting x_1 to true, we can add nodes with label c_2 and c_3 to the star and similarly for the other truth assignments. In this example it is not possible to select rules corresponding to set the truth-value of x_1 and x_2 and construct the star with labels c_1, c_2, and c_3 for the leaves and l_3^2 for the center.

7 W[1]-hardness in the Degree for Trees

In this section, we show that the graph matching problem is W[1]-hard in the degree even for trees. This is a strong hint that we will not find a FPT algorithm in the degree even for trees. We show this by a reduction from the longest common subsequence problem, which is shown to be W[1]-hard in the number of sequences, even for fixed sized alphabets, in [8].

Recall that in the longest common subsequence problem, we are given d sequences $s_1, \ldots, s_d$ over an alphabet Σ and a number k. The question is whether there is a sequence s of length k which is a subsequence of each of the d strings $s_1, \ldots, s_d$.

We have labels $\{c, t, h, e\} \cup \Sigma$. We construct a tree and rules as follows. The tree consists of a center node with label c, from which d paths evolve with labels corresponding to the strings, terminated by a node with label e and a further path of k nodes of label h followed by one of label t. See Figure 5 for a sketch of the construction.

We have rules that, when considered in a backward direction, allow to shrink a node with a label from Σ into the center and rules which shrink a star of d nodes of the same label from Σ and a node with label h to the center. Finally, we have the terminating rule which shrinks the center with d nodes of label e and a node of label t to the starting symbol.

We now argue that the query graph can be constructed, if there is a common substring of length k. We do it again by showing how the query graph can be reduced to the starting label by backward substitutions. Starting form the query graph, we shrink all nodes into the center that are not part of the common substring of length k until the first remaining node of each string is of the common substring (in arbitrary order). Then we shrink the d nodes corresponding to the first letter of the common substring and one node of label h to the center. We continue this way until all nodes of the strings but the last ones are

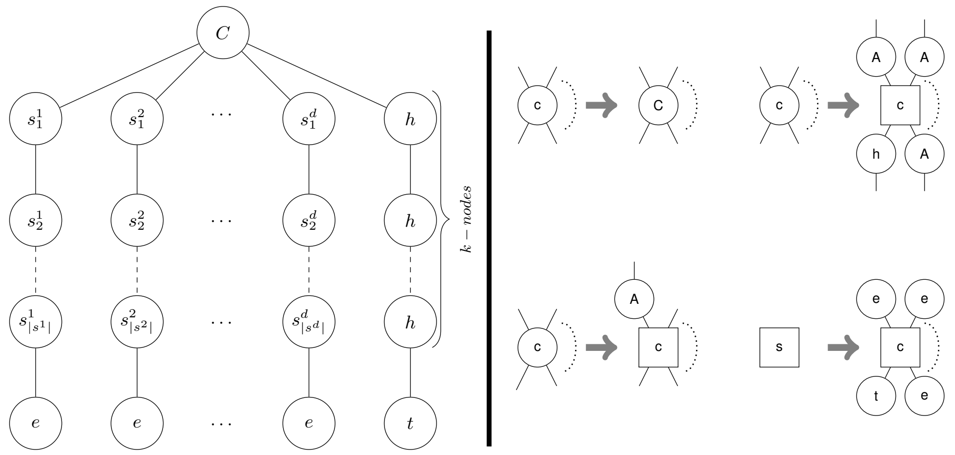

Fig. 5: The sketch of the input graph on the left and the rules on the right. The rules have to be created for all $A \in \Sigma$. The rule at the top left replaces an arbitrary string by an arbitrary symbol. The rule at the bottom left extends an arbitrary string by an arbitrary symbol. The rule at the top right extends all strings by the same symbol and the counter for the number of matching symbols increases by one. The rule at the bottom right generates the starting configuration.

shrunken into the center. Finally, we can use the rule that creates the starting label from this graph.

Finally, we argue that there is a common substring of length k if a (sub)graph of the query graph can be constructed. Here, we use the forward direction when arguing. We can only start with the rule that creates the star with d labels e and one label t for the leaves and label c for the center. By the construction, it is only possible to emerge new nodes from the node with label c and there will be only one such node. Hence, we can only match the query graph and not a subgraph of it and we have to create exactly the given strings, either by adding a single letter to a string or by adding the same letter to all strings and one node of label h. Notice, that we have to add the node with label h always to the part terminated by t as otherwise, we will not be able to construct (a subgraph of) the query graph. During the creation of the strings, we have to create k nodes of label h and hence, the corresponding substrings build a common substring of length k.

Notice that the rules only depend on d and the alphabet Σ, i.e. the hardness result holds even for a fixed graph grammar. Furthermore, we do not need edge-labels in the normal form of the rules by giving each string a unique copy of the alphabet.

The construction can be used to show the hardness of the graph matching problem and the subgraph matching problem.

8 Estimating the Size of the DP-Table in a Large Database

We showed in Section 5.2 that the running time becomes polynomial for a class of graphs with bounded ζ-value. More precisely, we can estimate the size of the dynamic programming table for a graph $G = (V, E)$ as $|\mathcal{L}^V| \cdot |E|^{\Delta(G) \cdot \zeta(G)}$. We calculated this value over all purchasable structures in the Zinc-database[2] and report on the statistics in this section. The database contains about 22 million structures and is often scanned through in computational drug discovery. For the vast majority of structures, the parameter value is at most 6, but can become as large as 22. In Table 1, we give the histogram on the distribution of this value.

Table 1: Histogram of $\zeta(G)$ over the all purchasable structures G in the Zinc-Database.

Number	1	2	3	4	5	6	7	8	9	10
ζ	105329	13429106	4027109	3450440	541304	308973	62943	25141	5649	1698

Number	11	12	13	14	15	16	17	18	19	20	21	22
ζ	518	268	152	72	29	35	11	10	1	2	0	1

Furthermore, we report on the number s of subsets U of the nodes such that $G[U]$ and $G[V \setminus U]$ are connected. If u is this number, the size of the dynamic programming table can also be bounded by $|\mathcal{L}^V|s^{\Delta(G)}$ and hence, if it is polynomially bounded in the input size and the maximum degree is bounded as well, we obtain a polynomial algorithm, too. For most of the instances, this value is between 16 and 2048, only about $2‰$ have a larger value, and only 209 instances have a value exceeding $100,000$. Two instances have more than 2 million components. See Figure 6 for a plot of the data. Notice that it is a worst-case estimation and in practice the dynamic programming table will be much smaller.

The results clearly indicate that these numbers are generally small in this database and hence, we are confident that an implementation of this algorithm will be very efficient.

On the other hand, there are known structures, such as fullerenes, where the ζ-value becomes very large and hence, the algorithm proposed will not be efficient. These structures typically are difficult as well for other query languages such as SMARTS, which often rely on precomputation of properties such as the smallest set of smallest rings [11] (known as the minimum cycle basis in computer science).

In the future, we will evaluate these values over the database pubchem [3] consisting of more than 120 million structures.

9 Conclusion

In this work, we present an algorithm for graph rewriting-based molecular structure search that is polynomial if the degree of the rules of the graph rewriting system and the cut size between each pair of connected components are bounded. We further showed empirically that this assumption is reasonable.

Our algorithm is a generalization of the graph-matching algorithm of Lautemann for hyperedge replacement grammars. It only works on hypergraphs whose duals are graphs,

[2] `http://zinc.docking.org/`

[3] `http://pubchem.ncbi.nlm.nih.gov`

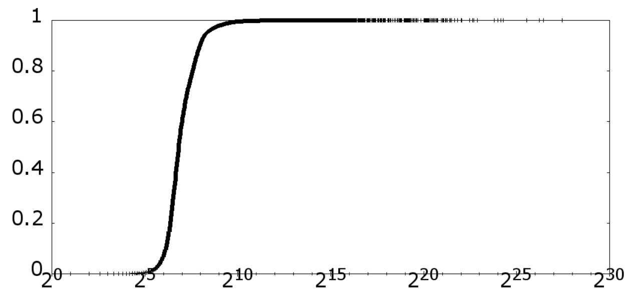

Fig. 6: The graph shows the percentage (y-axis) of the structures G for which the number of subsets U of nodes such that $G[U]$ and $G[V \setminus U]$ are connected is below the value on the x-axis. For more than 2‰ of the structures the value lies between 16 and 2048.

i.e. every node in the hypergraph has exactly two incident edges at any time during the derivation. We will investigate whether it is possible to extend our algorithm to general hypergraphs.

The algorithm is not a fixed-parameter algorithm and we showed that the problem is W[1]-hard even on trees. Nevertheless, it is possible that there is an algorithm that is exponential in the degree, but polynomial in the maximal size of a minimal cut or a related graph parameter. We will investigate this further in future work.

Furthermore, we will implement our algorithm. In order to become efficient enough to scan databases, suitable algorithm-engineering techniques have to be developed and applied.

References

1. Sheila Ash, Malcolm A. Cline, R. Webster Homer, Tad Hurst, and Gregory B. Smith. Sybyl line notation (sln): A versatile language for chemical structure representation. *Journal of Chemical Information and Computer Sciences*, 37(1):71–79, 1997.
2. AK Dehof, HP Lenhof, and A Hildebrandt. Predicting protein NMR chemical shifts in the presence of ligands and ions using force field-based features. In *Proceedings of the German Conference on Bioinformatics 2011*, 2011.
3. Anna Katharina Dehof, Alexander Rurainski, Quang Bao Anh Bui, Sebastian Böcker, Hans-Peter Lenhof, and Andreas Hildebrandt. Automated bond order assignment as an optimization problem. *Bioinformatics*, 27(5):619–625, 2011.
4. Matthias Dietzen, Elena Zotenko, Andreas Hildebrandt, and Thomas Lengauer. Correction to on the applicability of elastic network normal modes in small-molecule docking. *Journal of Chemical Information and Modeling*, 54(12):3453, 2014.
5. Hans-Christian Ehrlich and Matthias Rarey. Systematic benchmark of substructure search in molecular graphs - from ullmann to VF2. *J. Cheminformatics*, 4:13, 2012.
6. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
7. Clemens Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Inf.*, 27(5):399–421, 1990.
8. Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003.

9. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.
10. Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978.
11. Antonio Zamora. An Algorithm for Finding the Smallest Set of Smallest Rings. *Journal of Chemical Information and Computer Sciences*, 16:40–43, 1976.