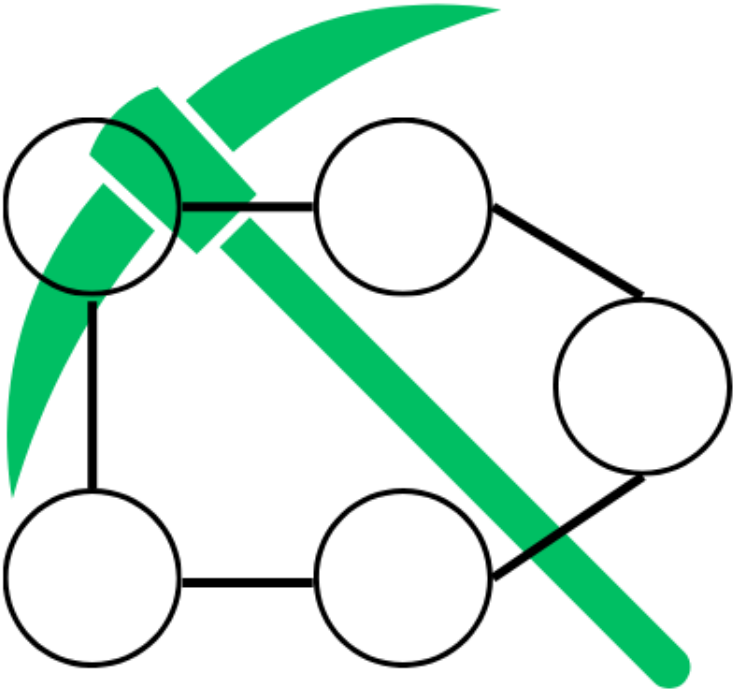
GRAPHMINER

A SOFTWARE TOOL PERFORMING GRAPH MINING AND ENRICHMENT ANALYSIS ON GROUPS OF MOLECULES



Giovi Duivenvoorden
Student number 1117165
Msc Thesis Bioinformatics (BIF80336)
David Meijer, Justin J. J. van der Hooft, Marnix H. Medema
Wageningen University & Research

ABSTRACT

Over the last decades there has been a sharp increase in morbidity and mortality due to microbial infections, as increasingly more microbials are resistant against antimicrobials. This could be reduced by finding new antibiotics to which no resistance is present yet amongst microbials. However, there has been a discovery void of new antibiotics over the last 30 years. This void is partially due to the isolation and intensive laboratory screening which are required to determine whether a compound has antibiotic activity. A computational approach which allows selection before screening could accelerate the process of finding new antibiotics. Proposing that particular substructures within molecules would be the active part of the molecule, makes it interesting to look at substructures which are over enriched in current antibiotics compared to not-antibiotics. These substructures could serve as a first selection on which molecules to screen on antimicrobial activity in phenotypic assays. Previously, multiple tools, such as MoSS, gSpan and GASTON, have been developed which could retrieve the substructures of a molecule and even compare two groups. However, these tools are not encoded in python, are not able to compare more than two groups and do not perform enrichment analysis. The goal of this new tool, GraphMiner, is to mine all substructures of the molecules in each group and perform enrichment analysis for each group, as to show which substructures are over or under enriched. GraphMiner is a command line tool in python, which has proven to be able to mine the substructures of the molecules within each group. Followed by an enrichment analysis for each group to determine which substructures are significantly over or under enriched compared to the total input. Resulting in clustering, which makes the results more interpretable as providing images of the structures characteristic for each cluster within each group. This is shown with a synthetic data set, in which the over enriched substructures were found as expected. Furthermore, a test with (not-)antibacterials and (not-)antivirals was run, which showed that the beta-lactam ring was significantly over or under expressed in groups of not-antibacterials. This structure is known to often have an antibacterial effect, and so proved that the desired substructures are found.

LIST OF DEFINITIONS

Closed substructures – substructures of which no supergraph is present which has the same support as the subgraph.

Heavy Atom – all atoms that are not a hydrogen.

Morgan Fingerprints – “enable mapping of certain structures of the molecule within certain radius of organic molecule bonds” [1].

SMILES – representation of molecules which is linear.

Subgraph Mining – finding the subgraphs present in a graph or graph database.

Supergraph – substructure that contains a subgraph and additional atom(s).

Tanimoto coefficient – “ratio of the number of features common to both molecules to the total number of features” [2].

INTRODUCTION

Drug discovery enabled a sharp decrease in mortality and morbidity caused by infectious diseases, mainly by introducing antimicrobials [3]. However, nowadays drug-resistant pathogens are spreading, including multi- and pan-resistant bacteria, while barely any new antibiotics are discovered. Antimicrobial resistance has thus become one of the top 10 global public health threats according to the WHO, as it puts the success of modern medicine at increased risk [4]. A main strategy against antimicrobial resistance is to discover new antibiotics.

One of the major sources of conventional medicine are natural products, making up for 25% of conventional medicine, although just 15% of plant species have been investigated [5]. Natural products can be found in microbial and animal sources as well [6]. A natural product could be defined as a '*chemical substance produced by living organisms via primary and/or secondary metabolic pathways which usually exhibits pharmacological activities that can be useful in treating various kinds of diseases*' [5]. Natural products are structurally distinct and more complex than synthetic compounds due to more sp3-hybridized atoms and stereocenters. Additionally, natural products often do not fulfill Lipinski's rule of five [7] and are biosynthetically accessible, thus suitable for improvement, innovation and flexibility [8–10]. But the isolation of these compounds is an essential first step for drug discovery. Several strategies to identify natural products include classical, laborious biological screening, analyzing multiple metabolites with computational approaches for identification and molecular networking; organizing MS/MS data to visualize clusters of analytes allowing for better prioritization [9]. Additionally, mass spectrometry can be used to detect metabolites in natural extracts by utilizing MS-based metabolomics [11]. But identification of these natural products can be challenging, just as obtaining sufficient biological material for research.

To increase the efficiency of new drug discovery by simplifying the screening of molecules, multiple computational approaches have been developed [9,10]. Based on natural products, pharmacophore models can be established to illustrate ligand-target binding models [10,12]. Pharmacophores are *'the ensemble of steric and electronic features that is necessary to ensure the optimal supra-molecular interactions with a specific biological target structure and to trigger its biological response'* [12,13]. Pharmacophore modeling focuses on chemical functionalities, enabling target searching with similar biological functionalities. This allows generating a pharmacophore model based on 3D structures of target-bound ligands or single ligands, which could be improved by using molecular dynamic structures. These 3D pharmacophore models can be used for virtual screening through large libraries, to find potential (drug) candidates [12–14]. However, this approach requires a 3D structure of either a ligand or target, thereby focusing on a particular type of drug during screening of the library and requiring previous knowledge.

Most of the research on natural products builds on phenotypic assays, making the straightening of molecular mechanisms and discovering of specific structures, in combination with ever-growing databases, time-consuming [9]. Therefore, the different groups of natural products with similar phenotypic results are explored. The aim is to determine whether there are (pharmacophoric) substructures with significant differences in enrichment between or commonalities within these groups. The substructures are obtained using (Frequent) Subgraph Mining. Subgraph Mining focuses on finding the subgraphs present in a graph or graph database [15]. Multiple mining strategies have been researched, using several algorithms including breadth-first search and depth-first search [16–21]. Breadth-first search starts with checking all subgraphs of a particular size and moves on by increasing the size by one [19]. Depth-first search first extends the first subgraph, until it is below the frequency threshold, and then moves on to the next subgraph [19]. Many graph mining algorithms are based on the Apriori principle; the frequency of a subgraph is at most the frequency of the subgraphs it contains [17–19].

For subgraph mining, multiple tools exist, including state-of-the-art tools GASTON and gSpan which both find frequent substructures above a threshold [22,23]. Another state-of-the-art tool is MoSS, which is based on finding frequent substructures and comparing the frequency of these substructures between two groups, as well as

finding all frequent substructures above a threshold [24]. Therefore, the knowledge gap consists of three parts, as visible in table 1. Firstly, there is not yet a subgraph mining tool in python that focuses on natural products and preserving particular substructures. Secondly, previous tools allow comparison between just two groups, instead of multiple ones. Thirdly, the enrichment analysis and clustering are not yet included in one tool together with the subgraph mining. This tool combines all missing parts and applies new methods to increase the speed of the analysis.

Table 1 Overview of GraphMiner vs. State-of-the-Art tools MoSS, gSPAN and GASTON.

	GraphMiner	MoSS	gSPAN	GASTON
Coding Language	Python	Java	C++	C++
Searching Algorithm	Breadth First Search	Depth First Search, starting at most frequent atom	Depth First Search, lexicographically	Depth First Search, focused on frequent bonds
Number of groups	Any	2	1	1
Performing statistical/enrichment analysis	Yes	No	No	No

GraphMiner is built to facilitate the search for differences in substructures between groups. A case study is searching for new antimicrobials based on structural features found significantly more present in proven antimicrobials and/or structural features found significantly more present in proven not-antimicrobials. Therefore, the tool enables more efficient and easily accessible comparison of different natural product groups. It both presents the frequency of substructures within a group as well as determines which substructures are over or under enriched within a group. The tool presents new possible substructures in natural products. These natural products could be researched further in laboratory experiments as suitable drug candidates. This advances the drug discovery process, which is essential to restrain antimicrobial resistance from limiting modern medicine success. Furthermore, GraphMiner is encoded in python, increasing the interpretability of the tool amongst researchers, as python has become a more popular coding language [25].

METHODS & IMPLEMENTATION

The GraphMiner tool consists of two main parts, which each produce their own output files. The overview of the tool is visible in figure 1. For GraphMiner the following packages are used; python 3.11.3, rdkit 2023.3.1, matplotlib 3.7.1, pandas 2.0.2, ipython 8.14.0, statsmodels 0.14.0, scipy 1.10.1, timer 0.2.2 and numpy 1.24.3.

INPUT FILE

The first step is the input file (figure 1.1). The input file should consist of the SMILES of a molecule and an indication to which group the molecule belongs, this could be either a number or a group name. The number of groups is not limited by the tool. The file should be in a csv file format, but the separator can be determined by the user.

FILTERING

After loading the file, the molecules are split in groups, as indicated in the input file and thus supplied by the user, and for each group the filtering and search algorithm is performed separately. The filtering (figure 1.2) consists of multiple parts. The first selection takes place on the number of heavy atoms that is present in the molecule (see figure 2.A). Heavy atoms are all atoms except hydrogen. The default cut-off is set at 60 heavy atoms, as no molecules above 50 came through the graph mining within a timeout of 2 minutes. This is about 1/8 of the molecules in the dataset used in the test run of GraphMiner. To make sure, there is room for exceptions, the cut-off is set slightly higher. Still, the cut-off enables huge time reduction, while the number of molecules that timed-out is lowered with exactly the number of molecules that are filtered out based on size (appendix 1). The remaining molecules are filtered on whether they contain a dot in their SMILES. If a dot is present in the SMILES, this means the components of the molecule are disconnected, there is no covalent bond between the structures. Therefore, these parts are mined individually, but the results are combined together as one molecule to prevent influence on the statistical analysis later on.

The filtering is continued with a timeout function. This timeout function is built in to cover the second part of the filtering and the graph mining. The default time of the timeout function is 30 seconds, as it is right in the middle of the trade-off between the number of molecules that time out and the time the tool takes (appendix 1).

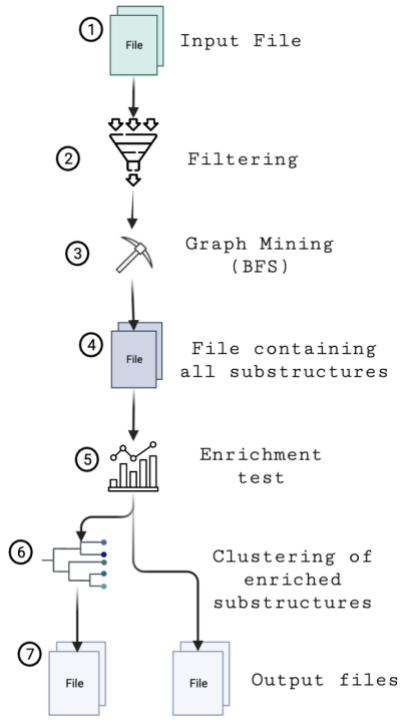


Figure 1 Overview method GraphMiner tool. *1. Input file – the input file should be a csv file containing the SMILES of a molecule and the groupname/-number. 2. Filtering – filtering on size and combination of substructures. 3. Graph Mining – performing breadth first search and mining all subgraphs. 4. File containing all substructures – first output file. 5. Enrichment test – using hypergeometric test on all substructures to find over/under enriched substructures. 6. Clustering of enriched substructures – making a dendrogram of all significantly different expressed substructures to determine different clusters. 7. Output files – files containing substructures, frequencies, p-values, dendrograms and images of significantly different expressed substructures.*

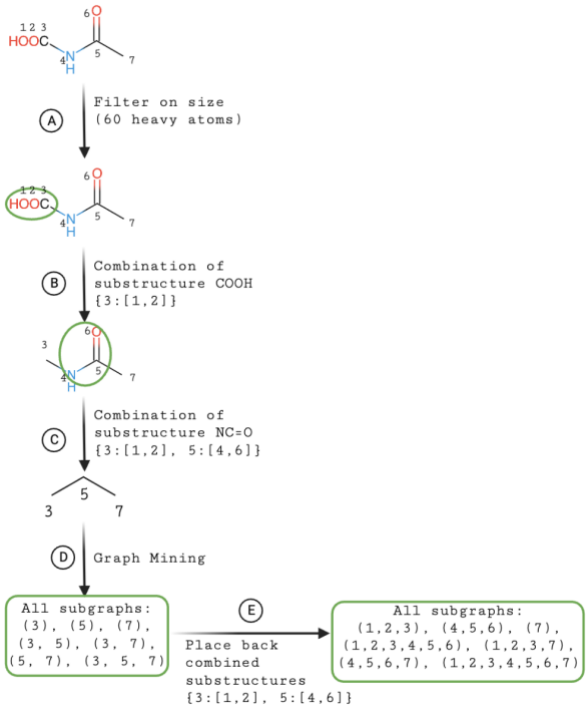


Figure 2 Overview of the filtering and graph mining. *The steps of the filtering, reducing graph complexity and subgraph mining including replacing are shown. These are steps 2 and 3 in figure 1. A. Filter on size – filter on the number of heavy atoms a molecule contains. B. Combination of substructure COOH – the acid group is replaced by a single carbon atom and the corresponding atom map numbers are stored to be replaced back later. C. Combination of substructure NC=O – the peptide bond is replaced by a single carbon atom and the corresponding atom map numbers are stored to be replaced back later. D. Graph Mining – Breadth First Search Algorithm is performed. E. The atom map numbers that were stored are replaced back in the substructures.*

REDUCING GRAPH COMPLEXITY

The last step of the filtering is the representation of a specific set of atoms, a substructure, as one atom, for an overview see figure 2. The representation is performed to reduce the time necessary for the graph mining by reducing the total number of possible substructures that need to be mined. The substructures that are combined are all chemical structures, nine in total. The substructures are an acid or ester group (C(=O)O), phosphoryl group (P(=O)(O)O), phosphate group (P(=O)(O)(O)O), sulfonyl group (S(=O)(=O)), sulfonic acid group (S(=O)(=O)O), peptide bond (NC=O), peptide bond with an oxygen group at the nitrogen (N(O)C(=O)), alcohol group or ether bond (CO) and carbonyl group (C=O). All these chemical groups are combined into a single atom, being either a C, P or S (see figure 2.B and 2.C). Each heavy atom in a molecule has been given a specific and unique atom map number (as visible in figure 2), to enable identifying the specific replaced atoms. The atom map numbers corresponding to the atoms that are removed from the molecule, are stored in a dictionary in a list of values. The key is the atom map number corresponding to the single atom that remains in the molecule. This is visualized in steps B and C in figure 2, where in step B, the two oxygens of the acid group are removed, and their corresponding atom map numbers (1 and 2) are stored as values, with the atom map number of the remaining carbon (3) as the key in the dictionary.

GRAPH MINING

After the filtering, the final molecule, on which the mining is performed, only contains the single atoms of the combined substructures. To perform graph mining, first all neighbours are determined. The neighbours are gathered using RDKit, which allows to retrieve all neighbouring atoms for a specific atom. This function works on each individual atom of the molecule. The atom map number of the atom is stored as the key in a dictionary, with as values a list of the atom map numbers of the neighbours of the specific atom. Based on this dictionary, a breadth first search approach is used to generate all possible substructures within the molecule, with the smallest all the single atom map numbers and the biggest the complete molecule. The breadth first search results in an output dictionary, with as key the length of the substructures and as a value a list of sets, with in each set an individual subgraph of atom map numbers.

The graph mining is encoded with a breadth first search approach (figure 1.3 & 2.D), but the code for a depth first search is available as well.

Due to the combination of substructures during the filtering, the resulting dictionary does not contain all atoms. Therefore, the removed atoms should be placed back in the retrieved substructures. This is encoded, based on the dictionary created during the filtering. The stored atom map numbers are placed back in the substructures where the single atom is present, thereby generating the full substructures (as visible in figure 2.E). In other words, the numbers in the values list are added to all substructures in which the atom map number of the key was present. The substructures are converted to SMILES, based on the atom map numbers.

Of these substructures, each unique substructure within a molecule is stored. When the substructures of all molecules within a group are mined, they are counted and stored. Sometimes, substructures are still generated with dots in their structures. These dots indicate that the substructure contains two separate substructures, and the connection is unclear. Thus, these substructures are left out of the dataset, which is in the test dataset about 3% to 4% of the substructures.

SUBSTRUCTURE FILE

The first output is generated in a csv file (see figure 1.4). This file contains all substructures in SMILES format, followed by columns displaying the frequency of the substructure in each group. Secondly, another output file is generated containing the groupnames. Both these files are essential for the second part of the tool, which is automatically performed as well.

ENRICHMENT TEST

The second part of the tool focuses on the differences in frequencies of substructures between the groups. This is determined using the hypergeometric test of scipy.stats (figure 1.5). The hypergeometric test uses the following function:

$$p(k, M, n, N) = \frac{\binom{n}{k}\binom{M-n}{N-k}}{\binom{M}{N}}$$

In this formula, k = total number of molecules in the specific group that contain the specific substructure, M = total number of molecules of all groups combined, n = total number of molecules that contain a specific substructure in all groups, N = total number of molecules in a specific group. Using this formula, the p-value is calculated for each substructure for each group separately. These p-values are written to an output file, displaying both the frequencies and the p-values.

As many substructures are generated, a multiple testing correction is performed. The default setting is Benjamini-Hochberg multiple correction, as it controls the false discovery rate and therefore is less stringent and finds more true positives [26,27]. But this could easily be swapped out within the tool for multiple other options, including Bonferroni and Holm. After the multiple testing correction, the significantly under or over expressed substructures are retrieved for each group separately. This results in a final list of all substructures that are significantly different expressed for that specific group compared to the other groups. Thus, it could be either over or under expression.

CLUSTERING OF SIGNIFICANTLY DIFFERENT SUBSTRUCTURES

These found substructures are further analysed to generate interpretable results (figure 1.6). A dendrogram is generated based on the pairwise distances between the Morgan fingerprints of the mined substructures, which are calculated using the Tanimoto coefficient. In the dendrogram, groups of substructures are created based on a distance cut-off. The cut-off is set at 1.5 as default, which is based on the results of the test datasets to create feasible groups. The groups created are retrieved and the maximum common substructure within these groups are determined and drawn as a molecular substructure as well as the largest substructure within the group.

OUTPUT FILES

As output, multiple files are generated (see figure 1.7 & table 2). At first, the two files of the first part are generated as explained. During the statistical analysis, multiple files are generated as well. All files will be generated in a folder, which is default named ‘GraphMinerResults’, but can be adjusted. For all files, see the overview below.

Table 2 Output files. *Overview of all output files and the contents therein.*

Output file name	Contents
substrfile.csv	All substructures found in the input file, with all the frequencies per group.
datafile.csv	Contains the list of all group names and the total number of molecules in each group.
pvaloverview.csv	All substructures found in the input file, with all the frequencies per group and the p values per group.
significantsubstr.csv	All substructures that are significantly different expressed sorted per group.
Images/*groupname*_dendrogram.png	The dendrogram of all significantly different expressed substructures.
Images/*groupname*/biggest_group*groupnumber*.png	The biggest substructure of this particular group
Images/*groupname*/mcs_group*groupnumber*.png	The maximum common substructure of this particular group

GRAPHMINER AS COMMAND LINE TOOL

The tool can be used by downloading from GitHub via https://github.com/moltools/GraphMiner. The tool can be installed on the command line using pip and run using multiple features (all visible in ReadMe on GitHub). When using all default features, the command line looks as follows 'GraphMiner -i *name input file*'.

DATA USED FOR TESTING GRAPHMINER

To perform tests on GraphMiner and retrieve the results, a dataset is used containing 250 molecules and 4 groups. The total dataset is retrieved from the Donphan database [28]. The same ratios are kept, making for 83 antibacterials, 34 antivirals, 111 not-antibacterials and 22 not-antivirals, which are completely randomly selected from the total dataset.

The synthetic data set test is performed using a dataset containing 120 molecules. In this dataset 4 groups of 30 molecules are present. The first group contained just carbon and oxygen atoms, the second group contained the same molecules, but with 18 with a sulfonyl group, the third group similar but with 18 with a phosphoryl group and in the last group similar but with 24 with a nitrogen atom.

RESULTS & DISCUSSION

GRAPHMINER HAS MORE FEATURES AND (ANALYSED) RESULTS THAN MOSS

To show that the tool is an improvement over the already present tools with similar function, a comparison is performed to MoSS. MoSS is a tool based on finding frequent substructures and comparing the frequency of these substructures between two classes, as well as finding all frequent substructures above a threshold [24]. The MoSS tool is run on the command line, which opens a new interface to enable the tool. The tool requires an input dataset, which contains three columns; id number, value and description of molecule in SMILES. MoSS splits the input in two groups, based on the value. By default, the split takes place at 0.5, with everything <= 0.5 in the focus group. MoSS only obtains closed substructures. All closed substructures above a certain frequency are obtained for molecules in the focus groups. For these substructures, the frequency in the other group is calculated. The main differences between the tools are displayed in the table below.

Table 3 Comparison GraphMiner & MoSS. *Showing all differences and similarities between GraphMiner and MoSS.*

GraphMiner	MoSS
All substructures	Closed substructures
All substructures	Only substructures above frequency (adjustable)
All substructures of all groups	Only substructures of focus group, and frequencies of other groups
571,861 substructures	6214 substructures
Multiple groups	Two groups
Runtime 3033 sec/50 min	Runtime 906 sec/15 min

As is visible, there are quite some differences between both tools. First of all, the GraphMiner tool results in many more substructures, which provides a bigger dataset for the enrichment analysis. This enables looking at, for example, maximum common substructures as to find the smallest substructure that is over/under enriched in particular groups. Furthermore, the substructures are obtained for each group in GraphMiner, while MoSS only returns frequent subgraphs for the focus group. Additionally, MoSS divides the input in two groups, while GraphMiner uses the groups supplied by the user, which could be any number of groups. On the other hand, the output of MoSS contains information on the absolute and relative presence, while GraphMiner focuses on the absolute presence, although performing enrichment analysis afterwards. Furthermore, MoSS works through all molecules, without requiring a time out function, and is thereby relatively quick. This could be due to a quicker algorithm, or due to the fact that MoSS just mines all substructures of one group, which was only a proportion of the molecules that GraphMiner has to mine through.

Improving GraphMiner is thus certainly possible, compared to MoSS, by reducing the time for graph mining and including relative presence and possibilities to select on minimum frequency of substructures which end up in the results. However, this could influence the enrichment analysis. The usage of closed substructures, as in MoSS, is applied to the list of significantly different substructures, to shorten it and show the biggest substructures.

COMBINING SUBSTRUCTURES AND CHANGING SEARCH ALGORITHM REDUCE TIME REQUIRED

Obtaining all possible substructures of all lengths of a given molecule using graph mining is time intensive. To try and reduce the time used by the graph mining, while still obtaining (almost) all possible substructures, several strategies were applied. First of all, two different graph mining algorithms were written and used in the tool, being Breadth First Search (BFS) and Depth First Search (DFS). Literature research had shown that DFS would result in the quickest graph mining algorithm [22–24]. However, after both have been encoded, the following result was obtained (table 4).

Table 4 Different runs with BFS or DFS and with or without substructure combining. *The input molecules are the number of molecules which are in the input file of the algorithm. The passed molecules are the molecules passing the first filtering steps on size and whether a molecule is present. Substructures is the number of substructures resulting from the graph mining. Timed-out mol is the number of molecules that is timed-out during the graph mining and thus not taken up in the substructures. Time is the amount of time it took to run GraphMiner in total.*

		1	2	3	4
Substructure combining	*YES / NO*	YES	NO	YES	NO
Search Algorithm	*DFS / BFS*	BFS	BFS	DFS	DFS
Time out	*Number of sec*	30	30	30	30
Input molecules	*Number of molecules*	250	250	250	250
Passed molecules	*Number of molecules*	218	218	218	218
Substructures	*Number of substructures*	571,861	567,116	503,522	465,416
Timed-out mol	*Number of molecules*	73	110	93	128
Time	*Sec*	3033	4124	3538	4583
Time	*Min*	51	69	59	76

As is visible in the table above, the BFS algorithm has shown to be quicker. Furthermore, fewer molecules are timed-out and more substructures are found, which shows that the results contain more different molecules, thus more information. Thus, all together this shows that the BFS algorithm works quicker and more efficiently in GraphMiner. However, this could be caused by the way the algorithms are encoded and therefore could be a bug in the program, as previous state-of-the-art tools are all encoded with a DFS algorithm. Thus, this could be solved by encoding a faster DFS algorithm to reduce the time mining takes even more.

Secondly, the reduction of time is mainly caused by the combination of substructures as explained in the Materials & Methods. The results, for both BFS and DFS, are visible in the same table above. As is clearly shown, the combination of the substructures causes a huge decrease in time used for graph mining, as well as an increase in the number of structures that are completely mined within the timeout, and thus obtained in the results.

Combining these two results, the fastest tool encoded is based on a breadth first search mining approach and thereby including the combination of substructures. The combined substructures do however cause a decrease in the number of substructures found, per analyzed molecule. This does not result in much loss of information, as the substructures that are combined, are chemical groups that are often found together in molecular structures.

EXPECTED SUBSTRUCTURES FOUND IN GROUPS OF SYNTHETIC DATASET AFTER ENRICHMENT TESTING

To determine whether the expected results are in the output of GraphMiner, a synthetic dataset is created. This dataset contains 4 groups of 30 molecules. The 30 molecules in the first group contain carbon and oxygen atoms. The second group contained the same 30 molecules, but with 18 with a sulfonyl group, the third group similar but with 18 with a phosphoryl group and in the last group similar but with 24 with a nitrogen atom. It is expected that those groups and substructures containing these added groups/atoms will be found as significantly over enriched in the particular groups. The results of the sulfonyl group are visible in the figures below.

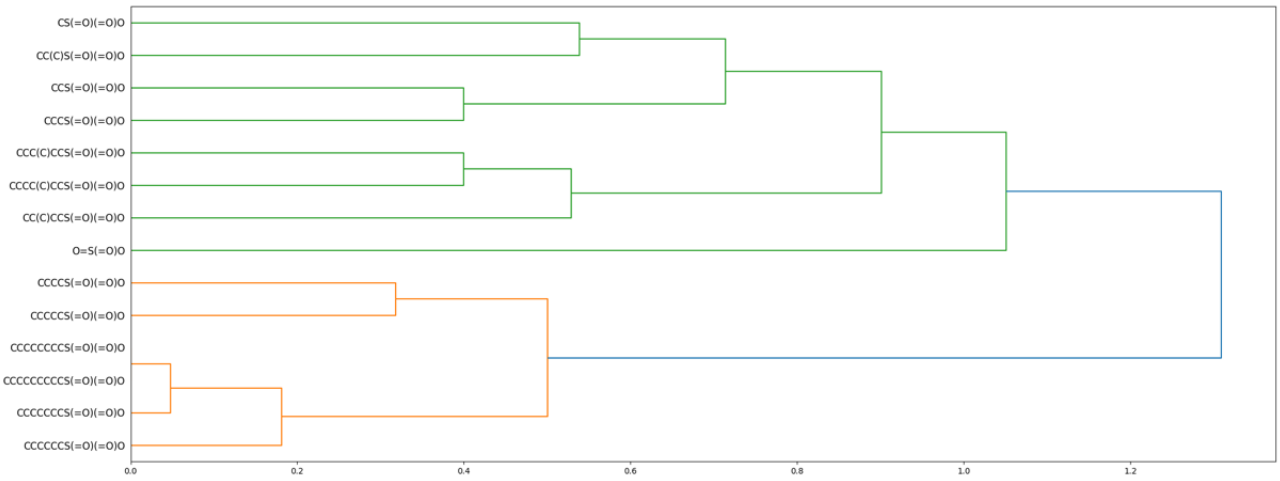


Figure 3 Dendrogram of differently expressed substructures of group with molecules containing sulfonyl group.

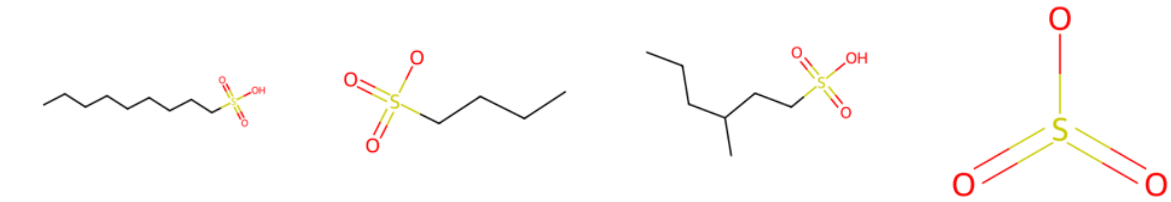


Figure 4 Overview of the results from clustering of group with molecules containing a sulfonyl group. *A. Largest substructure first group from dendrogram, B. Maximum common substructure first group from dendrogram, C. Largest substructure second group from dendrogram, D. Maximum common substructure second group from dendrogram.*

As is clearly visible in figures 3 and 4, only (sub)structures containing a sulfonyl group are found to be statistically over or under enriched, therefore providing some prove for the method of GraphMiner. Furthermore, for the third group only (sub)structures containing a phosphoryl group resulted as significantly over (or under) enriched (see appendix 2). Similarly, for the fourth group only (sub)structures containing a nitrogen are identified as significantly over (or under) enriched (see appendix 2). All in all, this proves to some extent that the tool works as proposed, with the expected resulting substructures. However, it is just one example and therefore not definite proof. Thus, next a case study regarding antibacterials and antivirals is discussed.

UNEXPECTED SUBSTRUCTURE FOUND IN NOT-ANTIBACTERIALS AFTER ENRICHMENT TESTING

After all substructures are mined, the following step was to determine which substructures are significantly over or under enriched within each group. This is calculated using a hypergeometric test, with a default p-value of 0.05. The enrichment is tested based on how many of the molecules contain a particular substructure. But many molecules are included in the testing, making multiple testing correction essential. This resulted in lists of structures which were significantly different expressed within the group tested compared to all molecules in the input dataset (appendix 3). To try and check whether the found substructures are as expected, the test dataset contained antibacterials, antivirals, not-antibacterials and not-antivirals. It is well known that when a beta-lactam ring is present in a molecule, often the molecule is an antibacterial (figure 5) [29], so this would be an expected result to be overexpressed in antibacterials. However, it is not found in the significantly different expressed substructures of the antibacterials (appendix 4.1). But it is shown in the significantly different expressed substructures of the not-antibacterials (figure 7), not as a ring structure, but with the same sequence of atoms.

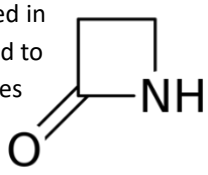


Figure 5 Beta-Lactam ring

However, these results of the hypergeometric test are quite dependent on multiple factors. At first, it is important which groups are put in the comparisons. If the comparison is between different groups of antibacterials, different results will be found, than by comparison between antibacterials and not-antibacterials. Therefore, there should be a clear goal as for which substructures to find, before determining which groups are entered into the tool. Furthermore, it is not for all molecules clear to which group they belong, and therefore this might cause some noise in the analysis. However, this would be a limitation on the dataset the user provided to the tool, not of the tool itself. Additionally, there are differences in the sizes of the group in the dataset, which could influence the results, as larger groups have a bigger impact on the results of the group in total than smaller groups. Lastly, the enrichment analysis is completely based on the results of the subgraph mining earlier in the tool, and thus all bigger molecules that do not come through the filter or the timeout function are not taken into account in this enrichment analysis, which could influence the results when a significant proportion of the molecules are removed. Therefore, both the filter on number of heavy atoms as the number of seconds the time out takes, are adjustable when using the tool to allow the user to make their own consideration.

CLUSTERING PROVIDES INSIGHT INTO FOUND SUBSTRUCTURES

The enrichment testing results in a list of significantly different expressed substructures for each group, which could be up to hundreds of substructures. To make the results interpretable and insightful, the substructures per group are clustered. The clustering is performed in a dendrogram and the clusters are formed using the cut-off of the dendrogram (appendix 4), for the not-anibacterials, the dendrogram is shown in figure 6. Based on the groups in the dendrogram, both the largest substructure found in the group as well as the maximum common substructure are determined. The structures of these are shown as a molecular figure (appendix 4), for not-antibacterials shown in figure 7.

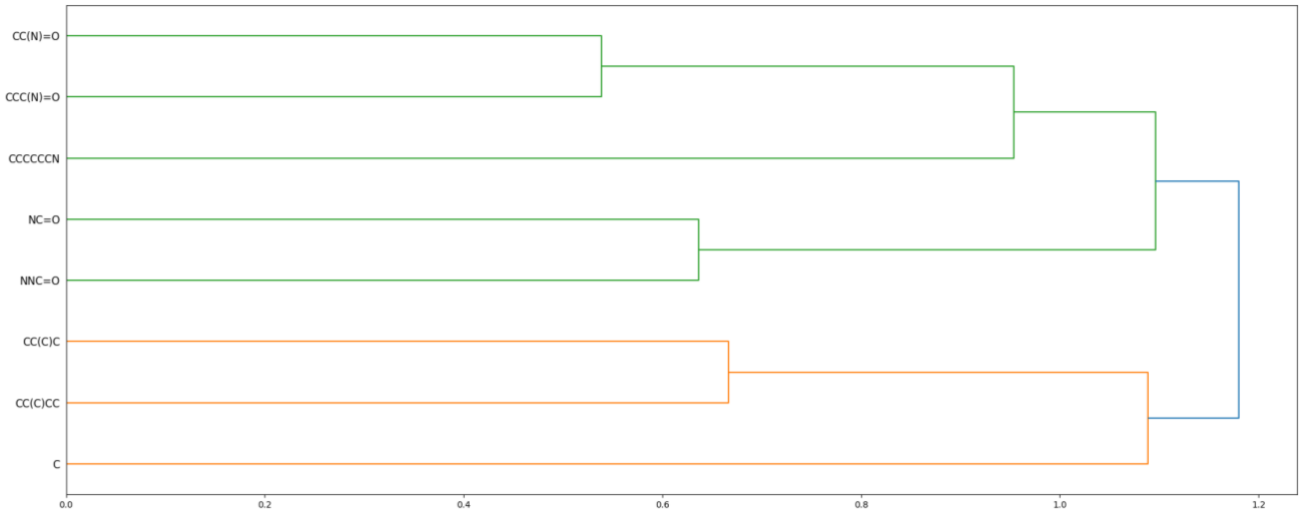


Figure 6 Dendrogram of differently expressed substructures of not-antibacterial molecules.

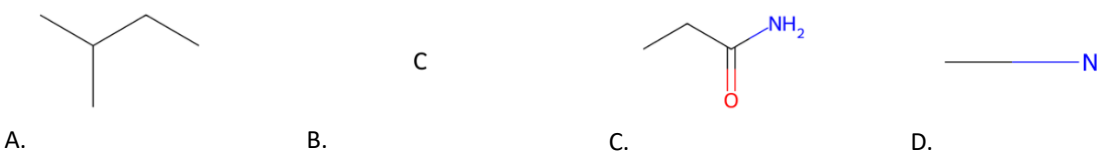


Figure 7 Overview of the results from clustering of not-antibacterial. *A. Largest substructure first group from dendrogram, B. Maximum common substructure first group from dendrogram, C. Largest substructure second group from dendrogram, D. Maximum common substructure first group from dendrogram.*

As is visible in figure 6, two clusters are formed from the significantly different expressed structures found in the not-antibacterial group. For both of these groups, the largest and maximum common substructure are determined and depicted in figure 7. Looking at these structures, it is clear that the maximum common substructure is highly dependent on the clusters formed during the dendrogram, as they are much smaller than the largest substructures. Additionally, when comparing the substructures in the dendrogram, it becomes clear that if the cut-off was placed differently, other maximum common substructures would have been found, which could influence the interpretation of the results. Therefore, the cut-off should be determined for each group separately and based on how many substructures are present, but possibly on how different these are from each other as well. The largest substructure gives more insight in the structures causing a molecule to be antibacterial.

Looking at the first group, the maximum common substructure, just a C (fig 7B), does not give any information on substructures that might be interesting to look at. This individual carbon atom could have ended up as an over or under enriched substructure due to differences in group size, while this should not have influenced the proportion of the molecules in which the carbon is present, as it should be present in all. Another possibility would be that at first aromatic carbon atoms and regular carbon atoms are counted separately and later all carbon atoms are displayed as regular carbon atoms. This could influence the result, when in one group much more or less aromatic carbon atoms were present. On the other hand, the branched carbon atoms of the largest substructure (fig 7A), provides some insight in interesting substructures. Same for the second group, where the maximum common substructure, is just a carbon-nitrogen (fig 7D), while the largest substructure shows the sequence of a beta-lactam ring (fig 7C), which has proven to be an interesting substructure. As previously mentioned, the cyclization within substructures is not always found by GraphMiner. Therefore, it is interesting to look at these substructures with an identical atom sequence, even if there is no ring shown.

The clustering could be influenced by the way the distance is calculated, as it is currently based on the Tanimoto coefficient for the pairwise distances. Using another method could result in different clusters and/or different substructures displayed. How confident the distances are could be tested using bootstrapping, as it estimates the variability of the distance calculated by generating a distribution of estimates [30]. The clearest influence is, as discussed, the cut-off used in the dendrogram, as it determines the numbers and the content of the clusters and thus the resulting structures. Lastly, the clustering results in a summary of the output and thereby concentrates the output, which could cause some loss of insight, albeit necessary if the output is huge.

However, the main set back of this method within GraphMiner is that large dataset (thousands of molecules) cannot yet easily be used within the tool, as too many significant over or under enriched substructures are found. For all these substructures, the pairwise distances have to be calculated and stored in a matrix, which takes up much space and could result in a storage issue. Thus, another possibility could be to combine all substructures based on substructure matches to the biggest substructure in such a group, which should be both less storage and time intensive, but poses some restrictions as there is no clustering and thus if one atom is different, the structure might end up in a different group. On the other side, completely different clustering approaches could be used, such as the CAST method, with approximate clique-finding, Jarvis-Patrick, with k nearest neighbors in common between groups, and more [31].

CONCLUSION AND FUTURE PERSPECTIVES

The goal of GraphMiner was to develop a tool to mine for substructures within groups of molecules and to perform enrichment testing to determine which substructures were significantly different expressed and therefore over or under enriched. All in all, GraphMiner is working as proposed. Firstly, the tool can mine all substructures within a group of molecules and result in a single output file with all substructures and frequencies. Secondly, an enrichment test is performed on the output of the mining, resulting in the substructures that are over or under enriched, which can be performed on multiple groups. Thirdly, these results are clustered per group to gain more insight in them and show substructures that might be interesting to look at during research on new

molecules. Lastly, it is all combined together in a command line tool encoded in python, with default settings for the adjustable parts, requiring just a single input file containing the molecules in SMILES and the group names or numbers.

GraphMiner enables research to gain more insight in what distinguishes different groups of molecules from each other, and thereby forming an idea of which substructures are most likely to (not) be present in new antimicrobials. These substructures could provide an early selection criterium for molecules, to reduce the number of molecules for which laboratory testing for antimicrobial activity should be performed. But, besides searching for antimicrobials, GraphMiner could be used for comparing any two (or more) groups of molecules for their differences. This is quite some improvement over previous graph mining tools, as it is encoded in python, a more universal coding language [25], more than 2 groups can be compared at the same time, a full enrichment and clustering analysis is built in and a unique way of combining substructures to reduce time in graph mining is included in GraphMiner.

GraphMiner could still be improved on a couple of fronts. First of all, it still takes quite some time to perform graph mining, and therefore a timeout function is required to ensure that the running time of the tool is within reason. Thus, trying to encode a faster or different algorithm or determining different methods as to speed up the graph mining, such as has been done with the substructure combining, could improve GraphMiner regarding the time sensitivity. However, a completely different approach could be to use multiple cores within the tool, as such each group can be mined using a separate core, which would improve the speed of the algorithm.

Secondly, the features of GraphMiner could be extended. For example, for some purposes it might be interesting to include a function which could only look at substructure above a certain frequency. But more important is that it will become visible whether a substructure is differently expressed within a certain group due to under- or over enrichment compared to the other groups. As to know, whether the substructure should be there or not, if used for selection.

Thirdly, adding more possibilities to GraphMiner regarding already present features would improve the tool as well. For example, multiple type of distance calculations could be useful for the dendrogram, to determine whether this has an impact or to base the distance calculations on the desired outcome. Similarly, different multiple testing corrections could be encoded, as this could result in different substructures after enrichment testing. Additionally, the possibility could be encoded for the user to supply their own (new) conserved substructures that could be combined before graph mining takes place. This would enable the user to cut out information on substructures that are known to be often found as a whole already and focus on new found substructures.

Lastly, it would be useable, if the two parts of the tool could be used separately. This should still be encoded within GraphMiner and arguments will have to be added to the command line tool to allow for this.

In conclusion, GraphMiner is working as proposed. However improvements could always be made and the tool could be built out further, improved working is shown compared to the current state-of-the-art tools.

CODE AVAILABILITY

GraphMiner is a command line tool, which can be retrieved from GitHub (https://github.com/moltools/GraphMiner) and installed on the command line using pip. For more information, see ReadMe on GitHub.

REFERENCES

1. Data Science for drug discovery research -Morgan fingerprints using Alanine and Testosterone examples in Python. | by Darko Medin | Medium. [cited 6 Jul 2023]. Available: https://darkomedin-datascience.medium.com/data-science-for-drug-discovery-research-morgan-fingerprints-using-alanine-and-testosterone-92a2c69dd765

2. Case: What are the differences between the Tanimoto and Dice similarity coefficients? - The Cambridge Crystallographic Data Centre (CCDC). [cited 6 Jul 2023]. Available: https://www.ccdc.cam.ac.uk/support-and-resources/support/case/?caseid=899a6a77-e379-4981-84f4-07de67f39016

3. Aminov R. History of antimicrobial drug discovery: Major classes and health impact. Biochem Pharmacol. 2017;133: 4–19. doi:10.1016/J.BCP.2016.10.001

4. Antimicrobial resistance. [cited 19 Jan 2023]. Available: https://www.who.int/news-room/fact-sheets/detail/antimicrobial-resistance

5. Süntar I. Importance of ethnopharmacological studies in drug discovery: role of medicinal plants. Phytochemistry Reviews. 2020;19: 1199–1209. doi:10.1007/S11101-019-09629-9/TABLES/1

6. Medema MH, de Rond T, Moore BS. Mining genomes to illuminate the specialized chemistry of life. Nature Reviews Genetics 2021 22:9. 2021;22: 553–571. doi:10.1038/s41576-021-00363-7

7. Lipinski CA, Lombardo F, Dominy BW, Feeney PJ. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. Adv Drug Deliv Rev. 2001;46: 3–26. doi:10.1016/S0169-409X(00)00129-0

8. Ganesan A. The impact of natural products upon modern drug discovery. Curr Opin Chem Biol. 2008;12: 306–317. doi:10.1016/J.CBPA.2008.03.016

9. Atanasov AG, Zotchev SB, Dirsch VM, Orhan IE, Banach M, Rollinger JM, et al. Natural products in drug discovery: advances and opportunities. Nature Reviews Drug Discovery 2021 20:3. 2021;20: 200–216. doi:10.1038/s41573-020-00114-z

10. Seidel T, Wieder O, Garon A, Langer T. Applications of the Pharmacophore Concept in Natural Product inspired Drug Design. Mol Inform. 2020;39: 2000059. doi:10.1002/MINF.202000059

11. Nagana Gowda GA, Djukovic D. Overview of Mass Spectrometry-Based Metabolomics: Opportunities and Challenges. Methods Mol Biol. 2014;1198: 3. doi:10.1007/978-1-4939-1258-2_1

12. Seidel T, Schuetz DA, Garon A, Langer T. The Pharmacophore Concept and Its Applications in Computer-Aided Drug Design. Prog Chem Org Nat Prod. 2019;110: 99–141. doi:10.1007/978-3-030-14632-0_4/TABLES/3

13. Giordano D, Biancaniello C, Argenio MA, Facchiano A. Drug Design by Pharmacophore and Virtual Screening Approach. Pharmaceuticals (Basel). 2022;15. doi:10.3390/PH15050646

14. Schaller D, Šribar D, Noonan T, Deng L, Nguyen TN, Pach S, et al. Next generation 3D pharmacophore modeling. Wiley Interdiscip Rev Comput Mol Sci. 2020;10: e1468. doi:10.1002/WCMS.1468

15. Nguyen LBQ, Zelinka I, Snasel V, Nguyen LTT, Vo B. Subgraph mining in a large graph: A review. Wiley Interdiscip Rev Data Min Knowl Discov. 2022;12: e1454. doi:10.1002/WIDM.1454

16. Wu D, Ren J, Sheng · Long. Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight. Int J Mach Learn & Cyber. 2018;9: 1445–1455. doi:10.1007/s13042-017-0655-y

17. Velampalli S, Jonnalagedda VRM. Frequent subgraph mining algorithms: Framework, classification, analysis, comparisons. Advances in Intelligent Systems and Computing. 2018;542: 327–336. doi:10.1007/978-981-10-3223-3_31/FIGURES/5

18. Albert-Ludwidgs-Universität G-K-A, Gebäude. Frequent Subgraph Miners : Runtimes Don ' t Say Everything. 2006.

19. Mrzic A, Meysman P, Bittremieux W, Moris P, Cule B, Goethals B, et al. Grasping frequent subgraph mining for bioinformatics applications. BioData Min. 2018;11: 20. doi:10.1186/s13040-018-0181-9

20. Takigawa I, Mamitsuka H. Graph mining: procedure, application to drug discovery and recent advances. Drug Discov Today. 2013;18: 50–57. doi:10.1016/J.DRUDIS.2012.07.016

21. Bhavsar SA, Patil VH, Patil AH. Graph partitioning and visualization in graph mining: a survey. [cited 11 Jan 2023]. doi:10.1007/s11042-022-13017-5

22. Nijssen S, Kok JN. The Gaston Tool for Frequent Subgraph Mining. Electron Notes Theor Comput Sci. 2005;127: 77–87. doi:10.1016/J.ENTCS.2004.12.039

23. Yan X, Han J. gSpan: Graph-based substructure pattern mining. Proceedings - IEEE International Conference on Data Mining, ICDM. 2002; 721–724. doi:10.1109/ICDM.2002.1184038

24. Borgelt C, Berthold MR, Patterson DE. Molecular Fragment Mining for Drug Discovery.

25. Why Python keeps growing, explained | The GitHub Blog. [cited 6 Jul 2023]. Available: https://github.blog/2023-03-02-why-python-keeps-growing-explained/

26. Benjamini Y, Hochberg Y. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B (Methodological). 1995;57: 289–300. doi:10.1111/J.2517-6161.1995.TB02031.X

27. Xu CJ, Ciampi A, Greenwood CMT. Exploring the potential benefits of stratified false discovery rates for region-based testing of association with rare genetic variation. Front Genet. 2014;5. doi:10.3389/FGENE.2014.00011/ABSTRACT

28. DONPHAN. [cited 25 Jun 2023]. Available: https://donphan-database.github.io/#/Molecule

29. Elgemeie GH, Azzam RA, Zaghary WA, Aly AA, Metwally NH, Sarhan MO, et al. Synthesis of N-sulfonated azetidines and β-lactemes and their applications. N-Sulfonated-N-Heterocycles. 2022; 89–112. doi:10.1016/B978-0-12-822179-2.00010-0

30. Bootstrapping – Introduction to Machine Learning in Python. [cited 7 Jul 2023]. Available: https://carpentries-incubator.github.io/machine-learning-novice-python/07-bootstrapping/index.html

31. Raymond JW, Blankley CJ, Willett P. Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures. J Mol Graph Model. 2003;21: 421–433. doi:10.1016/S1093-3263(02)00188-2

APPENDIX

APPENDIX 1 TABLES FOR DEFAULT SETTINGS

Table 5 Different runs showing the difference between filtering or no filtering on size.

		1	2	3
Substructure combining	*YES / NO*	YES	NO	YES
Search Algorithm	*DFS / BFS*	BFS	BFS	BFS
Filtering on size	*YES / NO*	YES	YES	NO
Time out	*Number of sec*	30	30	30
Input molecules	*Number of molecules*	250	250	250
Passed molecules	*Number of molecules*	218	218	250
Timed-out mol	*Number of molecules*	73	110	105
Time	*Sec*	3033	4124	5336
Time	*Min*	51	69	89

Table 6 Different runs showing the difference between different time out settings.

		1	2	3	4	5	6	7
Substructure combining	*YES / NO*	YES	YES	YES	YES	YES	YES	YES
Search Algorithm	*DFS / BFS*	BFS	BFS	BFS	BFS	BFS	BFS	BFS
Time out	*Number of sec*	1	5	15	30	60	120	240
Input molecules	*Number of molecules*	250	250	250	250	250	250	250
Passed molecules	*Number of molecules*	250	250	250	250	250	250	250
Timed-out mol	*Number of molecules*	164	132	114	105	95	88	No Data
Time	*Sec*	234	882	2191	5336	7064	19077	82948
Time	*Min*	4	15	37	89	118	318	1382

APPENDIX 2 RESULTS OF CLUSTERING SYNTHETIC DATASET

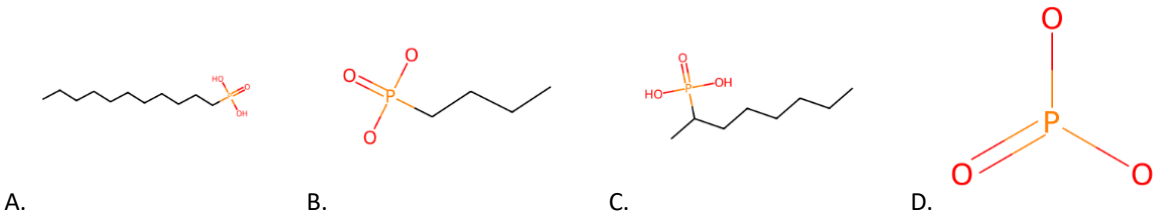


Figure 8 Overview of the results from clustering of group of molecules with phosphoryl groups. *A. Largest substructure first group from dendrogram, B. Maximum common substructure first group from dendrogram, C. Largest substructure second group from dendrogram, D. Maximum common substructure second group from dendrogram.*

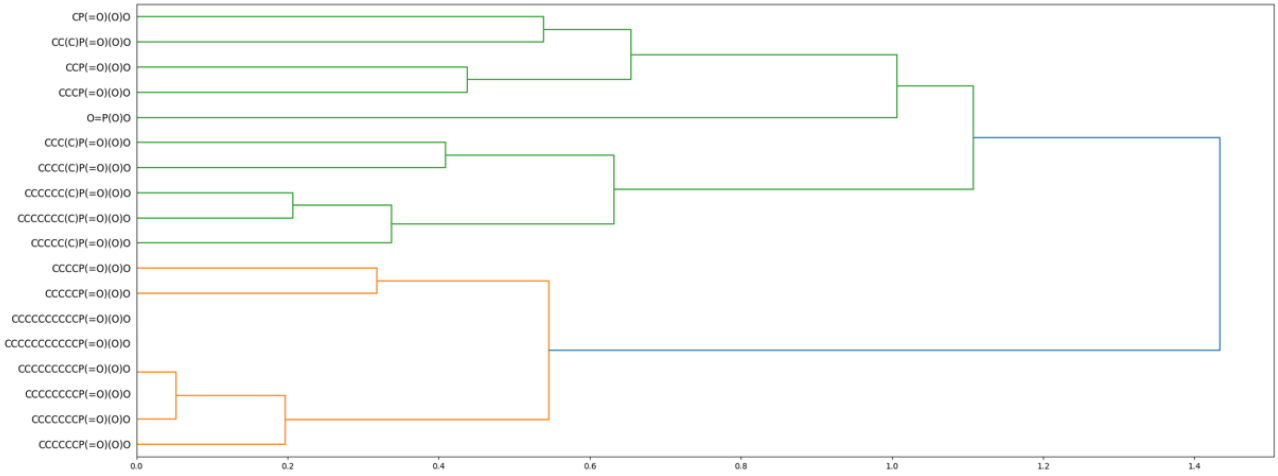


Figure 9 Dendrogram of differently expressed substructures of group of molecules with phosphoryl groups.

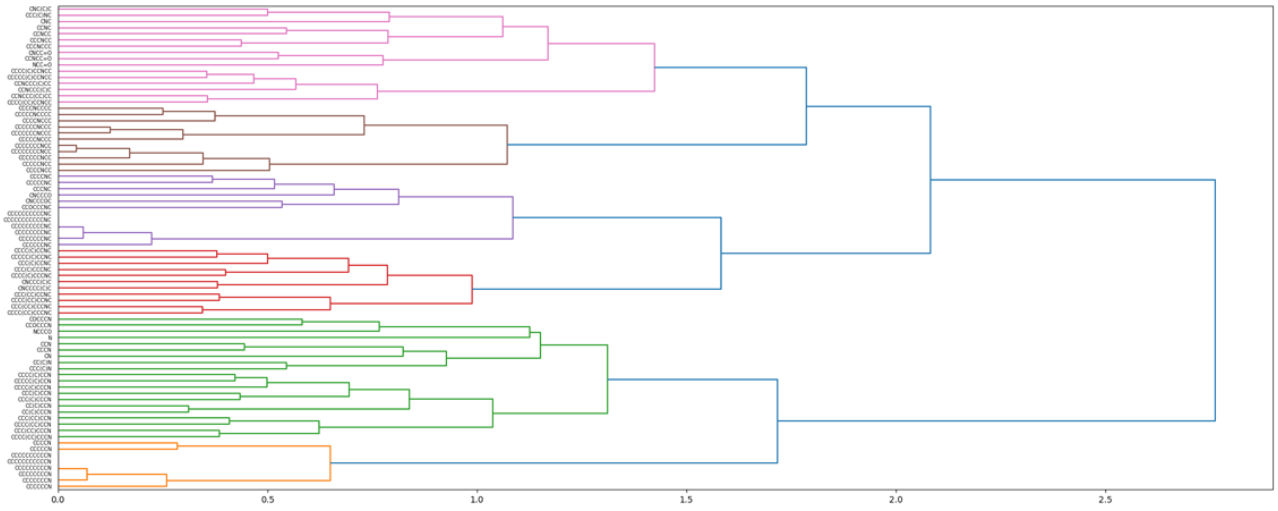


Figure 10 Dendrogram of differently expressed substructures of group of molecules with a nitrogen.

A. B. C. D.

Figure 11 Overview of the results from clustering of group of molecules with nitrogen. *A. Largest substructure first group from dendrogram, B. Maximum common substructure first group from dendrogram, C. Largest substructure fourth group from dendrogram, D. Maximum common substructure fourth group from dendrogram.*

APPENDIX 3 SIGNFICANTLY DIFFERENT EXPRESSED SUBSTRUCTURES

Table 7 Overview of all significantly different expressed substructures found per group.

Antiviral	Antibacterial	Not-Antibacterial
No significantly different expressed substructures present	CCC(C)(C)C, CC(C)(C)C, CCCC(C)CC, cccC(=O)O, CCCC(C)C, CCC(C)CC, CCC(C)C, cocC	ccC(N)=O, ccccccn, Cc(c)cc, cC(N)=O, Cc(c)c, NNC=O, NC=O, C
Not-Antiviral		
CCN(C)C=CN=Nc1ccccc1,Nc1cccc(C(F)(F)F)c1,cccc(c)N=NC=CN(C)CC,ccc(cc)N=NC=CN(C)CC,CN(C)C=CN=Nc1cccc1,cc(N)cc(c)C(F)(F)F,ccc(c)N=NC=CN(C)CC,cccc(c)N=NC=CN(C)C,ccc(cc)N=NC=CN(C)C,cc(c)N=NC=CN(C)CC,ccc(c)N=NC=CN(C)C,ccccN=NC=CN(C)CC,CCNC=CN=Nc1ccccc1,cnc(c(c)Cl)c(c)c,ccc(c)c(n)c(c)Cl,Clc1cccc2ccccnc12,cc(c)N=NC=CN(C)C,Nc1cccc(C(F)F)c1,ccc(N)ccC(F)(F)F,ccc(ccN)C(F)(F)F,FC(F)(F)c1ccccc1,ccccN=NC=CN(C)CC,cccccN=NC=CN(C)C,CNC=CN=Nc1ccccc1,cccc(c)N=NC=CNCC,ccc(cc)N=NC=CNCC,cc(Cl)c(n)c(c)c,ccc(Cl)c1cccn1,cnc1c(Cl)cccc1c,cc(Cl)c1ncccc1c,ccc1cccc(Cl)c1n,cccc1cccc(Cl)c1,cc(N)cc(c)C(F)F,cc(N)ccC(F)(F)F,cc(ccN)C(F)(F)F,cccc(c)C(F)(F)F,ccc(cc)C(F)(F)F,cccN=NC=CN(C)CC,ccccN=NC=CN(C)C,cccc(c)N=NC=CNC,ccc(cc)N=NC=CNC,ccc(c)N=NC=CNCC,NC=CN=Nc1ccccc1,ccccnc1ccccc1,ccc1cccc(Cl)c1,ccnc(c)c(Cl)cc,ccc(Cl)c(cc)nc,cc(Cl)c1ccccn1,ccnc(cc)c(c)Cl,cccc(n)c(Cl)cc,cc1cccc(Cl)c1n,cccc(c)cc(c)Cl,ccc(c)c(cCl)nc,ccc(cc)c(n)cCl,ccc(c)C(F)(F)F,ccN=NC=CN(C)CC,cccN=NC=CN(C)C,ccc(c)N=NC=CNC,cc(c)N=NC=CNCC,ccc(cc)N=NC=CN,cccc(c)N=NC=CN,C=CN=Nc1ccccc1,NcccccC(F)(F)F,cccccN=NC=CNCC,Cn1ccc(N)nc1=O,Nc1ccnc(=O)n1,ccc(nc)c(c)Cl,ccc(Cl)c(c)nc,ccccnc(cc)cc,ccccnc(c)ccc,cccnc1ccccc1,cc1cccc(Cl)c1,ccc(c)cc(c)Cl,ccnc(c)c(c)Cl,ccc(Cl)c(n)cc,cccc(n)c(c)Cl,cnc(cCl)c(c)c,ccc(c)c(n)cCl,ccnc1ccccc1Cl,cccncc(Cl)ccc,ccc1cccnc1cCl,cc(c)C(F)(F)F,cN=NC=CN(C)CC,ccN=NC=CN(C)C,cc(c)N=NC=CNC,ccc(c)N=NC=CN,C=CN=Nc(cc)cc,C=CN=Nc(c)ccc,Nc1cccc(CF)c1,ccc(N)ccC(F)F,ccc(ccN)C(F)F,FC(F)c1ccccc1,cccccC(F)(F)F,cccccN=NC=CNC,ccccN=NC=CNCC,cc(N)nc(=O)nC,cn(C)c(=O)ncN,cc(N)nc(n)=O,cnc(c)c(c)Cl,ccc(n)c(c)Cl,ccc(Cl)c(c)n,cnc1ccccc1Cl,ccccnc(c)cc,cccnc(c)ccc,cccnc(cc)cc,ccnc1ccccc1,cc(Cl)cc(c)c,Cc1ccc(N)c1,cc(c)c(n)cCl,cccncc(Cl)cc,cccc(Cl)cncc,cccccc(Cl)cc,cc1cccnc1cCl,cccc(cCl)ncc,cccc(cc)ccCl,CCN(C)C=CN=N,cN=NC=CN(C)C,cc(c)N=NC=CN,C=CN=Nc(c)cc,cc(N)cc(c)CF,cc(N)ccC(F)F,cc(ccN)C(F)F,NcccC(F)(F)F,cccc(c)C(F)F,ccccC(F)(F)F,ccc(cc)C(F)F,CN=Nc1ccccc1,ccccN=NC=CNC,cccN=NC=CNCC,cccccN=NC=CN,cnc(=O)n(c)C,ccn(C)c(n)=O,cc(Cl)c(c)n,cccc(cCl)nc,cccc(Cl)cnc,nc1ccccc1Cl,ccccnc(c)c,ccccnc(c)cc,ccnc(c)ccc,ccnc(cc)cc,ccccnccccc,ccc(cc)ccCl,ccccc(Cl)cc,cccccc(c)Cl,cc(C)cc(c)N,ccc(n)c(c)n,ccncc(Cl)cc,cccncc(c)Cl,Clcc1ccccn1,ccnc(cCl)cc,cccc(c)ccCl,nccccccccCl,CN(C)C=CN=N,C=CN=Nc(c)c,ccc(c)C(F)F,cccC(F)(F)F,CN=Nc(cc)cc,CN=Nc(c)ccc,N=Nc1ccccc1,cccN=NC=CNC,ccN=NC=CNCC,ccccN=NC=CN,C=CN=Ncccccc,NccccccC(F)F,cn(C)c(n)=O,Cn(c=O)cccN,Cn1ccnc1=O,O=c1ncccn1,cnc(=O)ncN,ccc(N)nc=O,cnc1ccccc1,ccc(cCl)nc,ccc(Cl)cnc,cccc(n)cCl,cccc(Cl)cn,cccnc(c)c,ccnc(c)cc,ccccncccc,cccncccccc,ccc(c)ccCl,cccc(Cl)cc,cc(n)c(c)n,ccncc(c)Cl,ccnc(c)cCl,nc1ccccc1n,ccccccccCl,cc(c)C(F)F,ccC(F)(F)F,CCN(C)C=CN,CN=Nc(c)cc,ccc(cc)N=N,cccc(c)N=N,ccN=NC=CNC,cN=NC=CNCC,cccN=NC=CN,C=CN=Ncccc,ccc(N)ccCF,ccc(ccN)CF,FCc1ccccc1,cccccC(F)F,Cnc(=O)ncN,ccnc(=O)nC,cccn(C)c=O,ccnc(n)=O,cnc(=O)nc,Ncnc(n)=O,cc(N)nc=O,cccc(c)nc,ccc(cc)nc,nc1ccccc1,cnc(c)cCl,cncc(c)Cl,ccc(n)cCl,ccc(Cl)cn,CN=Ncccc,Cnccc(n)N,ccnc(c)c,ccccnccc,cccncccc,ccncccccc,cc(c)ccCl,ccc(C)ccN,ccc(N)ccC,cccc(n)cn,cC(F)(F)F,CN(C)C=CN,C=CN(C)CC,CCC(N)=CN,CN=Nc(c)c,ccc(c)N=N,cN=NC=CNC,CCNC=CN=N,ccN=NC=CN,C=CN=Nccc,cc(N)ccCF,cc(ccN)CF,NcccC(F)F,cccc(c)CF,ccccC(F)F,ccc(cc)CF,cnc(=O)nC,ccn(C)c=O,ccc(c)nc,cnc(n)=O,Ncccnc=O,ccc(n)cc,ccc(c)n,cc(n)cCl,cc(Cl)cn,CN=Ncccc,cccccN=N,cn(C)c=O,cccnccc,ccncccc,cc(N)ccC,ccc(n)cn,ccnc[nH],cccnccCl,C=CN(C)C,CC(N)=CN,C=C(N)CC,cc(c)N=N,CNC=CN=N,cN=NC=CN,C=CN=Ncc,ccc(c)CF,cccC(F)F,NccccccCF,Cnc(n)=O,cnc(c)c,nc(n)=O,CN(C)CC,cccnc=O,ccccccnc,ccccN=N,CN=Nccc,cc(n)cn,ccnccc,CccccccN,cnc[nH],ccnccCl,C=C(C)N,N=NC=CN,cN=NC=C,cc(c)CF,ccC(F)F,CCNC=CN,cccccCF,CCncccn,cc(c)n,Ncnc=O,ccccnc,cccccn,cnccCl,cccN=N,CN=Ncc,ccnCCO,Cncccn,cncccn,FC(F)F,CCnccn,nc[nH],C=CN=N,cC(F)F,CNC=CN,C=CNCC,CCC=CN,NcccCF,ccccCF,c[nH],ncccn,nccCl,ccN=N,Cnccn,cnCCO,C=CNC,NC=CN,CC=CN,CccBr,cccCF,nccn,cccn,cccN,CNCC,ccnC,C=CN,cN=N,CN=N,ccCF,ncN,ccn,ccc,ccN,CCn,cnC,N=N,FCF,cCF,cN,cn,cc,Cn,CF,N,c,n		

APPENDIX 4 RESULTS OF CLUSTERING ANTIBACTERIAL/ANTIVIRAL

APPENDIX 4.1 ANTIBACTERIAL

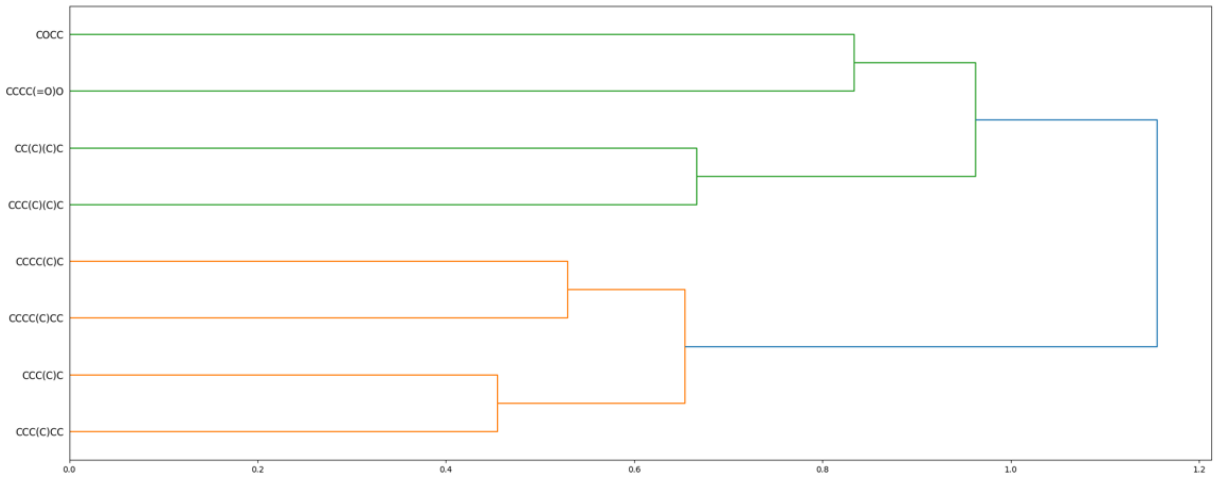


Figure 12 Dendrogram of differently expressed substructures of antibacterial molecules.

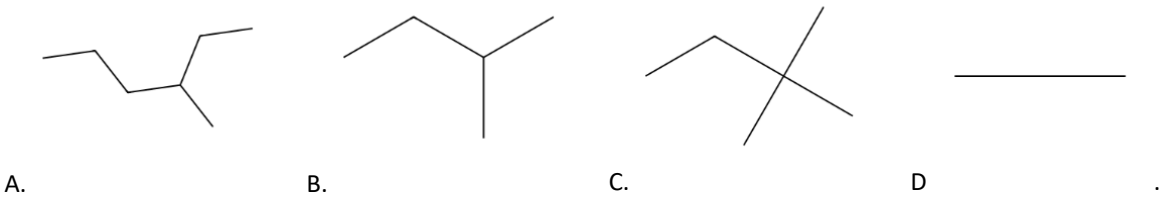


Figure 13 Overview of the results from clustering of antibacterial. *A. Largest substructure first group from dendrogram, B. Maximum common substructure first group from dendrogram, C. Largest substructure second group from dendrogram, D. Maximum common substructure second group from dendrogram.*

APPENDIX 4.2 NOT-ANTIVIRAL

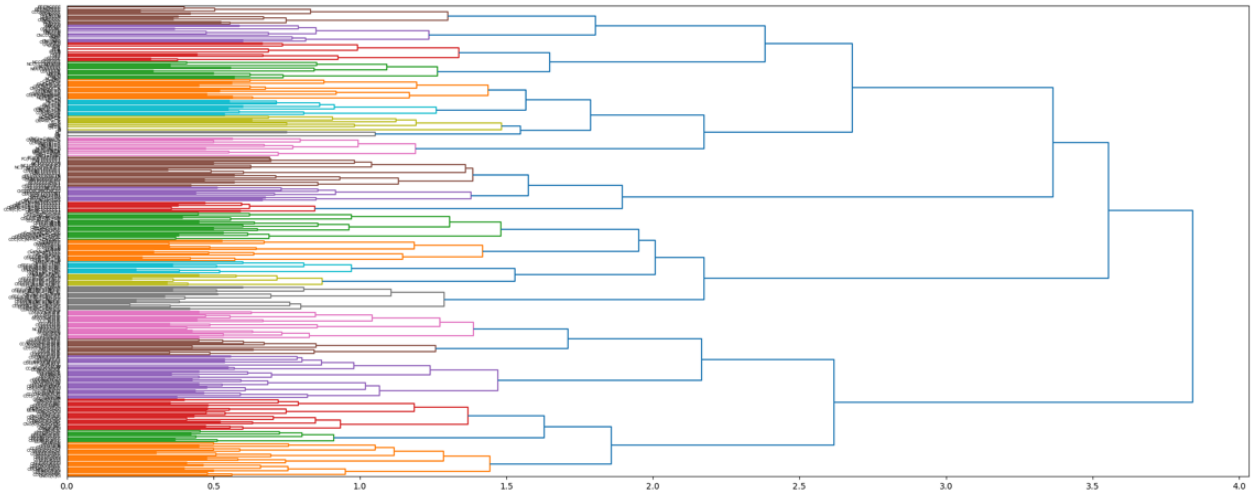


Figure 14 Dendrogram of differently expressed substructures of antibacterial molecules.

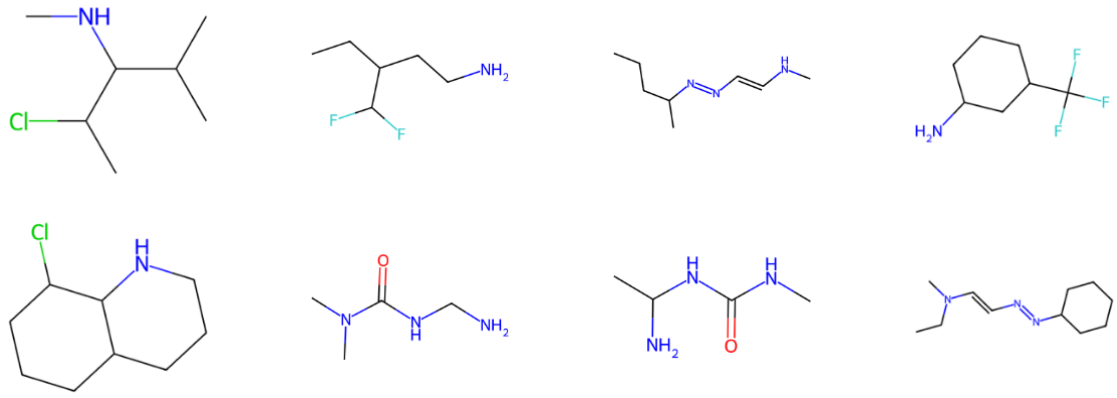


Figure 15 Couple of the largest substructures retrieved from the groups from the dendrogram of not-antiviral. *Same groups and order as the figure below.*

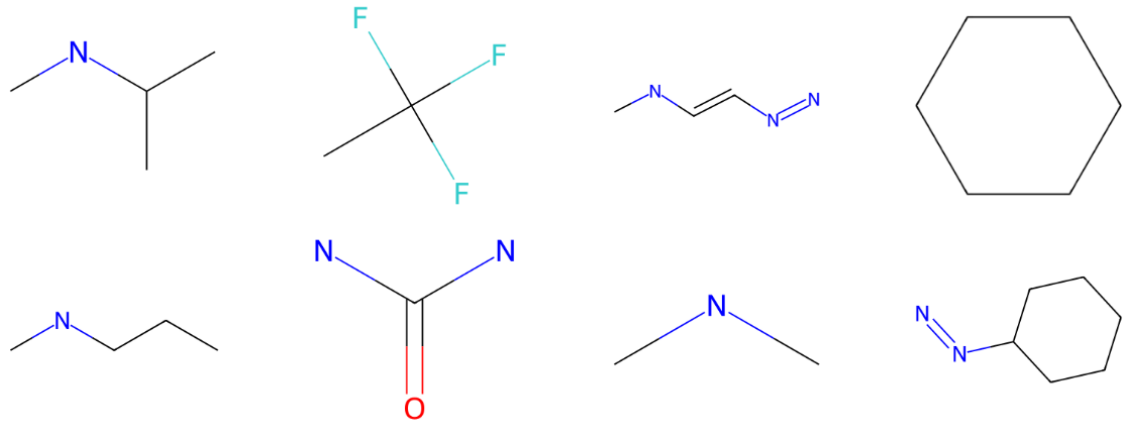


Figure 16 Couple of the maximum common substructures retrieved from the groups from the dendrogram of not-antiviral. *Same groups and order as the figure above.*