MASARYK UNIVERSITY
FACULTY OF INFORMATICS


# Annotation and validation of ligand molecules

MASTER'S THESIS

**Bc. Vladimír Horský**

Brno, spring 2014

# Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Vladimír Horský

**Advisor:** RNDr. Radka Svobodová Vařeková, Ph.D.

# Acknowledgement

I would like to thank my supervisor, Radka Svobodová Vařeková, for her patience, support, advice and guidance. Then, I would like to express sincere gratitude to my parents for their unwavering support. Finally, I would like to thank my dear friends for cheering me up and for reminding me where the nearest open bar was.

# Abstract

Ligand molecules are a large group of molecules with diverse functions in living organisms. Because of their importance, they are often in the spotlight of various research fields, for example (and not limited to) cheminformatics or pharmacology. Ligands can be found in chemical databases, where they are often incorrectly annotated or stored in an invalid state. The objective of this thesis was to design and implement algorithm for validation and annotation of ligand molecules. Firstly, it was necessary to understand the chemical background of ligand diversity and similarity. A suitable way of representing ligand molecules in computer that will facilitate effective discovery of similarity between pairs of ligands has been found (in the form of the modified molecular graph) and implemented. Application LiCMP (Ligand CoMParator) has been designed and implemented as the result of this thesis. The program has then been used to annotate 10 247 ligands extracted from the Protein Data Bank against 1 865 reference ligands downloaded from the LigandExpo. 47.74 % of input ligands have an issue with their structure. LiCMP was then successfully cross-validated with SwCMP using a set of 34 208 saccharides to be annotated and 924 saccharides from LigandExpo as the reference set.

# Keywords

ligand, ligand motive, ligand molecule, Protein Data Bank, PDB, LigandExpo, annotation, molecular graph, molecule comparison, isomorphism, parallelization, C++, Qt

# Contents

# 1 Introduction

Ligands are rather small molecules that bind to biomacromolecules in vast array of living organisms and form complexes with them [1]. They perform many significant chemical functions such as activation of biomolecules and transmitting signals between biochemical structures. Because of their functional variety, ligands are often studied by pharmacology with the ambition of discovering new and better drugs.

Cheminformatics is a new interdisciplinary science field that has been established to discover and implement effective ways of processing chemical data that are stored in today's large chemical databases [2]. It utilizes mathematics, informatics, and information technology approaches to work in silico with promising molecules, usually drugs. Cheminformatics can, in certain areas, provide better results than experimental chemistry while potentially using less time and money.

Ligands stored in various databases have been often extracted from their complexes with biomacromolecules using automated methods, and therefore are predicted to either be incorrectly annotated, or to have problems in their structure. The objective of this thesis is to design and implement algorithm for validation and annotation of ligand molecules. A challenge for design of such algorithm is the potentially unlimited amount of distinct ligand molecules.

Firstly, it is fundamental to understand the chemical background of ligand diversity and similarity. A suitable way of representing ligand molecules in computer that will facilitate effective discovery of similarity between pairs of ligands has to be found, studied, and implemented. Validation and annotation algorithm will then be designed and implemented.

Realized algorithm will be used to validate and annotate a set of ligand molecules that have been extracted from the Protein Data Bank [3] as a part of the metalloprotein research project at the National Centre for Biomolecular Research (NCBR) [4]. A set of ligands with confident annotations that have been downloaded from the LigandExpo [5] will be used as reference ligands in the validation and annotation process.

# 2 Theory

## 2.1 Chemical background

### 2.1.1 Ligand

There exist two types of ligands. In coordination chemistry, ligands are molecules (or functional groups) that binds to a central metal atom [6]. In biochemistry, ligands are (usually small) molecules that bind to a biomolecule to provide specific functionality in organism [1]. Main focus of this thesis lies in ligands from the view of biochemistry.

Target biomolecules for ligand binding are usually proteins and DNA molecules. Roles of ligands include, and are not limited to, substrates (molecules upon which chemical reactions, catalyzed by enzymes, are conducted), neurotransmitters (molecules that transmit signals between neurons), activators (molecules that increase activity of target enzymes) and inhibitors (molecules that decrease activity of target enzymes) [1]. Therefore, due to their extensive functionality options, ligands are often studied by pharmacological disciplines.

Despite being usually classified as small molecules, ligands can range from single ion [7] to a protein [8]. Because of the way they make their functionality happen, they are not in most cases bound to the target receptor molecule via covalent bond(s). Instead, they realize their bonding using intermolecular forces (e.g. van der Waals forces and hydrogen bonds). They do, however, change their conformation when such bonding occurs. This observation is important for the design of algorithms that are to compare ligands in their default conformation to ligands that have been extracted from some larger structure in their bound conformation.

### 2.1.2 Bond properties

A chemical bond [6] is a force interaction that binds atoms together. Bound atoms are in an energetically more favorable state then they were in before establishment of the bond. Bonds can be divided into two groups [6]. Intramolecular bonds (also known as strong chem-

ical bonds) are bonds that hold atoms together in molecules. They are based on the electrostatic attraction between nuclei of bound atoms and shared valence electrons from overlapping valence diameters in the case of covalent bonds [9], or on electrostatic attraction between two differently charged atoms in the case of ionic bonding. Intermolecular bonds (also known as weak interactions) repulse or attract whole molecules. They are weaker than forces of covalent bonds. Subtypes of intermolecular bonds include dipole-dipole interactions and Van der Waals forces.

The covalent radius is a measure of the size of an atom that participates in a covalent bond with another atom. The sum of covalent radii of two atoms in a bond should be equal to the length of the bond. Reference valence diameters are measured mainly using X-ray diffraction (or much less commonly using rotational spectroscopy and neutron diffraction of crystals) and tabulated for multiple bond orders of atoms from the periodic table.

Reference covalent radii have been tabelated and published [10], while covalent radii for single bonds [11], double bonds [12] and triple bonds [13] have been tabelated as well. It is important to note however that above mentioned values are not universal, because covalent radius of an atom depends on the chemical specifics of environment it exists in. Bonds between atoms of significantly dissimilar electronegativity tend to be shorter than bonds between two atoms of same element [6]. Rather than exact values, above mentioned tabulated radii should therefore be taken as average or idealized values.

### 2.1.3 Chirality in chemistry

If an object is chiral (derived from the Greek $\chi\varepsilon\iota\rho$ - "hand"), it cannot be superposed to its mirror image [14]. Enantiomorphs (or enantiomers, as they are called in chemistry) is a term for a chiral object and its mirror image that cannot be superposed to it. The concept of chirality can be found in mathematics, physics and chemistry.

Most of chirality in chemistry revolves around stereogenic centers [15, 16] (also called stereocenters). Stereogenic center is an atom with connected groups such that swapping relative positions in two of them in space leads to creation of a stereoisomer [15]. The most common stereogenic center is a carbon atom in tetrahedral configu-

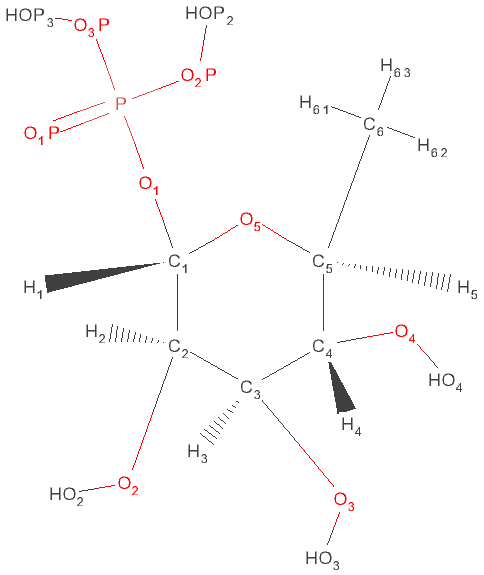



Figure 2.1: 6-deoxy-mannose-1'-phosphate, an example of a molecule with five chiral carbon atoms ($C_1$, $C_2$, $C_3$, $C_4$ and $C_5$). Source: LigandExpo.

ration with four different ligands attached to it [17]. Chiral molecule can theoretically have $2^n$ enantiomers[1] (where $n$ is the number of chiral centers), even though not every enantiomer exists naturally due to steric restrictions. A concrete example of a chiral molecule with five chiral carbon centers is depicted in figure 2.1. All naturally occurring amino acids, as well as saccharides, are chiral. Certain enzymes distinguish between possible enantiomers of its ligands.

Aside from carbon atoms, that are very common, nitrogen, phosphorous and several kinds of metal atoms in tetrahedral conformation can be chiral by the same rules as those that apply to carbon atoms.

To differentiate chiral molecules from achiral molecules, as well as enantiomers from each other, three naming conventions have been developed:

- **R/S system** [18] is the most common and most general naming convention of chiral molecules. It does not depend on any reference molecule and has no connection to the other two

1. Le Bel-van't Hoff rule (1874)

presented naming conventions. In this convention, each stereogenic center is given either R- or S- preposition. To assign a R- of S- preposition to a stereogenic center, two steps have to be taken. First, substituents that are bound to a stereogenic center are each given a priority based on their atomic number in accordance to the Cahn-Ingold-Prelog priority rules [19]. Then, it is necessary to view the center in a way that places the lowest-priority substituent farthest away from the viewer. If the priority of the remaining three substituents decreases in counterclockwise direction, it is a S- center. Otherwise, it is a R- center [18].

- **(+)/(-) system** [20] classifies enantiomers by the way they affect polarized light. If an enantiomer rotates the plane of polarized light clockwise (when the light travels straight to the viewer), it is a (+) enantiomer. Otherwise, it is a (-) enantiomer.

- **D/L system** [21] gives D- or L- prefixes to enantiomers based on their configuration similarity to glyceraldehyde. In the case of amino acids, configuration prefix can be obtained using the "CORN" rule. First, it is necessary to view the chiral carbon atom in a way that places the hydrogen atom farthest away from the viewer. The, if the arrangement of groups COOH, R and $NH_2$ is clockwise, it is a D- enantiomer. Otherwise, it is a L- enantiomer [22]. Nearly all naturally occurring amino acids are L- enantiomers [23].

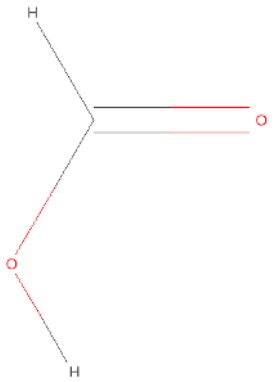
## 2.2 Mathematical background

### 2.2.1 Molecular graph

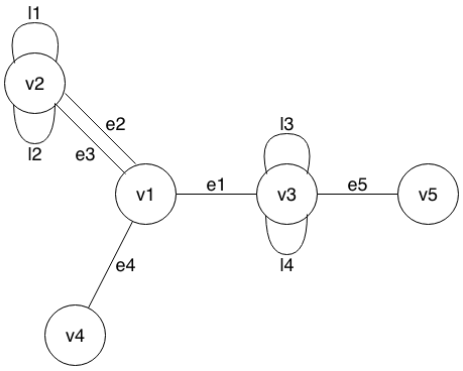
Molecular graph [2, 24, 25] is a formalized mathematical representation of a molecule. Rather that describing absolute spatial configuration of molecule in space, it characterizes the topology of a molecule.

Molecular graph is an ordered quintuple $G = (V, E, L, \varphi, \beta)$, where $V = \{v_1, v_2, ..., v_m\}$ is a non-empty set of vertices, $E = \{e_1, e_2, ..., e_n\}$ is a multiset of edges, $L = \{l_1, l_2, ..., l_o\}$ is a multiset of loops, $\beta$ is a finite set of atom symbols and $\varphi$ is a map $V \to \beta$.

Vertices represent individual atoms, edges represent bonds between atoms, loops represent valence electron pairs that are not shared with any other atom (lone electron pairs) and $\beta$ is a set that contains symbols of elements which atoms are present in the graph. $\varphi$ is a relation that gives an element symbol to every atom in the graph. Bonds between atoms are represented as edges of the graph, i.e. by unordered pairs of vertices (e.g. $\{v_1, v_2\}$ represents a bond between atoms $v_1$ and $v_2$) in the multiset $E$. Lone electron pairs are also represented by an unordered pair of vertices that the atom forms with oneself (e.g. $\{v_1, v_1\}$). Bonds of higher order are represented by multiple copies of a bond in $E$ in a similar way as multiple lone electron pairs are represented. An example of a molecule represented by a molecular graph is depicted in figure 2.2.




(a) Structural formula.
Source: LigandExpo.




(b) Molecular graph visualization.

$G = (V, E, L, \varphi, \beta)$,
$V = \{v_1, v_2, v_3, v_4, v_5\}$,
$E = \{\{v_1, v_3\}, \{v_1, v_2\}, \{v_1, v_2\}, \{v_1, v_4\}, \{v_3, v_5\}\}$,
$L = \{\{v_2, v_2\}, \{v_2, v_2\}, \{v_3, v_3\}, \{v_3, v_3\}\}$,
$\beta = \{C, O, H\}$,
$\varphi(v_1) = C, \varphi(v_2) = \varphi(v_3) = O, \varphi(v_4) = \varphi(v_5) = H$

(c) Formal description of the molecular graph.

Figure 2.2: Two representations of formic acid molecule - via structural formula and via molecular graph.

Bonds of a graph can alternatively be represented by an incidence matrix [25], which is a square shaped symmetric matrix with $|V|$ rows. At each position $a_{ij}$ there is either the order of bond between atoms $v_i$ and $v_j$, or 0 of no such bond exists. At position $a_{jj}$ there is the number of lone electron pairs present at atom $v_j$. Since information contained in multisets $E$ and $L$ can be fully inferred from the incidence matrix, it is possible to define molecular graph as an ordered quadruple $G = (V, A, \varphi, \beta)$, where where $V = \{v_1, v_2, ..., v_m\}$ is a non-empty set of vertices, $A$ is the incidence matrix defined earlier, $\beta$ is a finite set of atom symbols and $\varphi$ is a map $V \to \beta$.

### 2.2.2 Graph isomorphism problem

Graph isomorphism [26] is a graph theory term that deals with identity of two graphs, and is defined as follows: Two graphs $G = (V, E)$ and $G' = (V', E')$ are considered isomorphic if a bijection $f : V \longrightarrow V'$ exists such that $x, y \in E$ if and only if $f(x), f(y) \in E'$ holds for all $x, y \in V$, $x \neq y$. Such an $f$ is called an isomorphism of graphs $G$ and $G'$ [26]. Trivially, finding an isomorphism of two graphs can be viewed as finding a way of renaming vertices of the first graph so that the result looks like the second graph. The isomorphism relation is an equivalence relation, therefore it partitions the class of all possible graphs into equivalence classes [27]. Members of each class are isomorphic to the rest of graphs of said class.

The graph isomorphism problem (i.e. determination whether two finite graphs are in isomorphic relation with each other) is a difficult problem. It is still uncertain if this problem can be solved in polynomial time for every possible graph. Practical algorithms still exhibit exponential time complexity on their worst case scenario input graphs [28]. While certainly useful, determining only whether two molecular graphs are isomorphic or not does not provide enough granularity to be used as a molecule similarity decision approach. Suppose there are two ligand molecules that only differ in the presence of a single oxygen atom that is missing in one of the mentioned molecules because of its mistreatment by the automated extraction method used to extract the ligand from some larger macromolecular structure. A graph isomerism problem solver would just return "no" and completely miss the glaring similarity amongst said lig-

ands. Therefore, it is necessary to operate with smaller units than whole graphs.

A subgraph is defined as follows: Let $G$ and $G'$ be graphs. We say that $G$ is a subgraph of $G'$ if $V(G) \subseteq V(G')$ and $E(G) \subseteq E(G')$ [26]. The subgraph isomorphism problem then aims to determine if $G$ contains a subgraph that is isomorphic to $H$ when graphs $G$ and $H$ are given. This problem is NP-complete (as has been proven [29]) and can be solved via the use of brute force depth-first tree-search enumeration algorithm, even though there exist solutions that enable solving the subgraph isomorphism problem for certain specific classes of graphs in polynomial time [30].

## 2.3 Computer representation of molecule structure

### 2.3.1 1D structure

One-dimensional structure [2] (1D structure) carries information about element types of atoms that form a molecule. The chemical formula is defined as a sequence of element symbols $(A_i)_{x_i}$ where $A_i$ is an element type and $x_i$ is number of atoms of element type $A_i$ that are present in the molecule. To give an example, formula $C_{12}H_{22}O_{11}$ represents maltose molecule, which contains 12 carbon atoms, 22 hydrogen atoms and 11 oxygen atoms. Element symbols can occur multiple times in the formula when it is required to show that the structure of a molecule consists of several significant parts (e.g. the formula of butyric acid $C_4H_8O_2$ can be also written as $CH_3(CH_2)_2COOH$ to emphasize that it is a carboxylic acid) .

To load a molecule in 1D representation to a computer, it is sufficient to input the count of each element. No other information is represented in such structure. Also, many different molecules have the same chemical formula (e.g. $C_{12}H_{22}O_{11}$ is the formula of maltose as well as galactobiose). Therefore, 1D structure representation has limited usage besides some physical attribute computation (e.g. molecular weight [31]) and statistical usage.

OC[CH]1O[CH](O[CH]2[CH](O)[CH](O)[CH](O)O[CH]2CO)
[CH](O)[CH](O)[CH]1O

Figure 2.3: SMILES representation of 2D structure of maltose (single string).

### 2.3.2 2D structure

Two-dimensional structure [2] (2D structure) describes the topology of a molecule, i.e. which atoms are connected to each other and what is the order of each bond. Coordinates of atoms in space (and information that can be inferred from them) are not present in this representation. The most human-friendly representation of 2D structure is via structure formula, i.e. an image with defined notation for element symbols and bonds. In structure formula, atoms are represented by their element symbols. Bonds are represented by a line between two atoms (or more lines closely together in parallel if described bond has higher order than one). An example of a structural formula is depicted in figure 4.1a. Such representation, albeit human-friendly, is not suited for use in computations. Instead, two computer-friendly representations of 2D structure are being commonly used.

Line notation [2] is a molecule specification that represents 2D structure as an ASCII string. Several line notations have been developed and used over time, e.g. InChI[2], WLN[3], SMILES[4], SMARTS[5] and SLN[6]. Main advantages of this representation are its significant storage efficiency and a degree of human readability. Algorithms to convert 2D structure representation to a line notation representation vary between line notation types. In the case of SMILES, five rules and depth-first traversal (after breaking rings) through a molecule is employed in the conversion process [32]. Reverse order of conversion is possible as well, although converting line notation formula to a 2D structure formula representation as well as to a 3D structure representation can lead to ambiguous results [33]. To give an example, SMILES representation of maltose is showed in figure 2.3.

2. International Chemical Identifier
3. Wiswesser Line Notation
4. Simplified Molecular Input Line Entry Specification
5. SMILES Arbitrary Target Specification
6. SYBYL Line Notation

Molecular graph representation looks at molecules as multigraphs - structures studied by graph theory discipline. Description of this representation method can be found in subsection 2.2.1. Molecular graphs can be extended to contain more information about bonds and atoms, such as charge or true 3D structure of molecule in the form of Cartesian coordinates of atoms (see section 4.1). Molecular graphs can be physically represented in computer memory in three different ways:

- atoms as objects and bonds as pointers between them
- indexed array of atoms and array of index pairs as bonds
- an incidence matrix along with an array of atoms

Each physical representation has its advantages and its disadvantages, depending on the algorithms that will work with represented molecules (e.g. efficient memory complexity requirement versus usability for GPGPU[7] computations requirement).

### 2.3.3 3D structure

Three-dimensional structure [2] (3D structure) describes the absolute configuration of atoms in space using a set of coordinates. It contains complete information about structure of a molecule. There are three types of commonly used coordinates: Cartesian coordinates, distance matrix and internal coordinates. Cartesian coordinate system is a widely used coordinate system that defines position of each point in space using three signed distances to three fixed perpendicular lines.

Internal coordinates [34] (also known as Z-matrix) define 3D structure of a molecule as a set of distances between atoms (in Ångstroms[8]), angles of pairs of bonds (in degrees) and torsion angles (in degrees, also known as dihedral angles, i.e. angles between planes defined by three bonds). Internal coordinates are relative to other, already defined atoms in the molecule. The first defined atom has no coordinates. The second atom is defined only by its distance

7. General-purpose computing on graphics processing units
8. $1\text{Å} = 10^{-10}m$

```
ATOM      1  C   FMT A   1      -0.095  -0.420  -0.001  1.00 10.00           C
ATOM      2  O1  FMT A   1      -1.124   0.213   0.000  1.00 10.00           O
ATOM      3  O2  FMT A   1       1.085   0.218   0.000  1.00 10.00           O
ATOM      4  H   FMT A   1      -0.126  -1.500   0.003  1.00 10.00           H
ATOM      5  HO2 FMT A   1       1.881  -0.331  -0.000  1.00 10.00           H
CONECT    1    2    3    4
CONECT    2    1
CONECT    3    1    5
CONECT    4    1
CONECT    5    3
END
```

Figure 2.4: An example of a connection table in the form of a PDB file FMT_ideal.pdb that contains the formic acid molecule. Source: LigandExpo.

to the first atom. The third atom is defined by its distance to either the first or the second atom (to which it is connected) and by bond angle between its bond and the other already existing bond. From the fourth atom onwards, all four coordinates are used. Conversion from internal coordinates to Cartesian coordinates is widely used (e.g. in protein structure modeling), albeit computationally intensive [35].

Distance matrix [36] describes 3D structure of a molecule in a coordinate-independent manner by a matrix of distances between every atom pair. It is a square matrix with $n$ rows where $n$ is the number of atoms in the described molecule. Distance matrix 3D structure description is used for NMR and X-ray data representation.

Connection table molecule representation method [2] is often used in various file formats designed for 3D chemical structure storage and exchange (e.g. the PDB file format, described in 3.2). It consists of two lists: A list of atoms and list of bonds between atoms. Each item of the list of atoms usually carries more information than just 3D coordinates (e.g. charge, name and residue membership), while each item of the list of bonds can also carry addition information, such as order of described bond. An example of a connection bond is enclosed in figure 2.4.

Since hydrogen atoms can usually be inferred from the rest of provided information, some connection table representations omit them entirely and are therefore called hydrogen-suppressed connection tables.

# 3 Tools and methods

## 3.1 Protein Data Bank

The Protein Data Bank [3] is a highly curated and annotated repository of experimentally determined three-dimensional structural data of mostly large biological macromolecules. PDB is maintained by the Worldwide Protein Data Bank (wwPDB) organization. wwPDB formalizes the repository and ensures its transparency to depositors from any part of the world [37]. The repository itself is physically managed by RCSB PDB[1], that is one of the founding members of wwPDB. Other members of wwPDB include PDBe[2], PDBj[3] and BMRB[4].

Data in the PDB are stored in PDB file format as well as in mmCIF[5] file format. Each entry has its own unique PDB ID that consists of four alphanumeric characters. As of $3^{rd}$ May 2014, there are 99 775 structures present in the archive. New data are added to PDB each Wednesday on a weekly basis.

PDB database can be viewed as a primary source of data. Many derived databases use data from PDB and categorize them in different manner. An example of such derived database is the SCOP[6] database that categorizes and sorts entries by type of their structure and assumed evolutionary relations [38].

## 3.2 PDB file format

PDB file format defines structure of files present in the PDB repository [39]. A PDB file holds information mainly about atoms that comprise a structure described by the PDB file, although it can contain additional information about described structure, such as crystallography data, citations, database references and various annotation of described molecule.

---

1. Research Collaboratory for Structural Bioinformatics Protein Data Bank (USA)
2. Protein Data Bank in Europe
3. Protein Data Bank Japan
4. Biological Magnetic Resonance Data Bank
5. macromolecular Crystallographic Information File
6. Structural Classification of Proteins

PDB file is a pure text file with fixed column positions and record sizes. Each line contains at most 80 characters due to legacy width constraint of a terminal screen. Each line starts with up to six character long record name (e.g. `ATOM`, `HETATM`, `REMARK`) that specifies type of information present on such line and ends with line feed character. Usable character set is restricted to non-control ASCII characters. Furthermore, the use of punctuation characters is deprecated.

## 3.3 Ligand Expo

As mentioned earlier, the Protein Data Bank contains mostly information about structures of proteins, nucleic acids and complex chemical structures. References to ligands, found within such structures, are stored in the Chemical Component Dictionary [40] that is maintained by the wwPDB and updated each week. Each ligand has its own, up to three-character long, alphanumeric ID (also known as residue code).

Ligand Expo [5] serves as a web interface to the Chemical Component Dictionary. It allows user to search the Dictionary in several ways, e.g. by ID, by chemical name or formula, by supplying a file in one of supported file formats, or by sketching a ligand to search for. Search results can be filtered, as well as refined - Ligand Expo can either search for exact match, or apply relaxation to the query (e.g. only match a subset of atoms from the query).

## 3.4 Visual Molecular Dynamics

Visual Molecular Dynamics [41] (VMD) is a software for complex biological structure analysis. It is not limited to static analysis and rendering, but can also visualize trajectories of molecular dynamics computations and render a video of them. VMD is multi-platform, supports plug-ins and is limited in import size only by the amount of system memory.

VMD can also be used for visualization and analysis of general molecules of varying sizes, as it supports import of molecule data from PDB files. VMD supports several visualization styles, such as (and not limited to) points and lines, bonds or space-filling models.

User can measure bond lengths as well as bond angles and torsion angles. Graphical user interface (GUI) as well as command line interface (CLI) are both available, while CLI supports scripting

VMD is available free of charge for non-commercial usage, but its usage should be cited in papers whenever it has been used in preparation of such paper[7]. It can be downloaded, after registration, from the website of the Theoretical and Computational Biophysics Group at the Beckman Institute for Advanced Science and Technology of the University of Illinois at Urbana-Champaign[8].

## 3.5 Programming language

The implementation part of this thesis has been written in the C++ programming language in its most current standardized version, C++11[9]. It is an intermediate level compiled programming language that is widely used for creation of application software of varying sizes as well as highly optimized software (drivers, industrial programs) [42]. C++ is a multi-paradigm language as it contains functional, object-oriented and generic programming capability, as well as procedural programming capability inherited from its predecessor, the C programming language [43].

While the standard library of C++ (STL) contains an array of useful objects and algorithms, it lacks platform-specific functionality [43]. Therefore, other libraries are often used to extend the capabilities of C++ while maintaining its advantages.

## 3.6 Qt Software development kit

Qt Software development kit (Qt SDK) is a multi-platform application development framework [44]. It is written in C++ programming language and supports programming in C++ and QML languages. Qt SDK extends functionality of the C++ programming language by providing features such as thread management, network support,

7. http://www.ks.uiuc.edu/Research/vmd/current/LICENSE.html
8. http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=VMD
9. Standardized as ISO/IEC 14882:2011

additional containers or file handling application programming interface (API). All of the added features are platform-independent.

Qt SDK also contains a multi-platform IDE, Qt Creator. Qt SDK can be downloaded[10] as a source code, or in compiled binary form with support for GNU Compiler Collection (GCC) and Microsoft Visual Studio compilers in various versions. It can be used under GPLv3[11], LGPLv2.1[12] or commercial license.

## 3.7 Software design patterns

A software design pattern is a reusable generalized solution to a certain type of problems [45]. Rather than pieces of code that can be pasted into existing software projects, they represent formalized description of best practices used for problem solving. Adherence to such practices decreases the amount of design issues that take roots early, but cause major problems later during the software development cycle, when refactoring is costly and difficult. In LiCMP, variants of two creational patterns have been implemented: Singleton and iterator.

Singleton pattern [45] is used when there should be only a single instance of an object that has to be globally visible and accessible. Its main advantage over the usage of global variables is less cluttering of global name space, because not only there can always be just a single instance of a singleton class, but several instances of global classes can be refactored into much less singleton classes (preferably only one). In C++, singleton classes can be trivially implemented by declaring attributes and methods of a class as static. While easy to implement, such solution lacks polymorphic flexibility regarding inheritance (static methods cannot be virtual in C++ since they are not associated with a class instance and therefore lack type information [43]) and provides no implicit protection against accidental instantiation of two singleton class objects (that will, unintentionally, share their static attribute values).

10. `http://qt-project.org/downloads`
11. GNU General Public License Version 3
12. GNU Lesser General Public License Version 2.1

Iterator pattern [45] specifies design of object used to access elements of other objects without exposing their inner structure. It should enable various traversal logics without excessively enlarging interface of accessed objects. Iterators should also allow more than one reading traversals of the same container object at the same time. Finally, containers themselves should be responsible for instantiating their own iterators. In the C++ STL[13], as well as in the Qt SDK[14], iterator patterns are implemented as template functions that have their own specialization for each container type that can be iterated over.

## 3.8 SwCMP

SwCMP is an application designed for batch sugar molecule annotation [46]. It compares input sugar molecules to reference molecules (valid sugars obtained from a credible source such as Ligand Expo) and decides of each input sugar on the basis of comparison results. SwCMP has also implemented isomer groups finder function which compares all input molecules with each other and classifies them into isomer groups (groups of sugars that are identical from the viewpoint of the program). Regarding limitations of SwCMP, it can only annotate sugar molecules with pyranose heterocycles, and lacks chirality comparison functionality and methods to deal with highly problematic input molecules.

13. C++ STL iterator overview: `http://www.cplusplus.com/reference/iterator/`
14. Qt SDK iterator overview: `http://qt-project.org/doc/qt-5/containers.html#stl-style-iterators`

# 4 Implementation

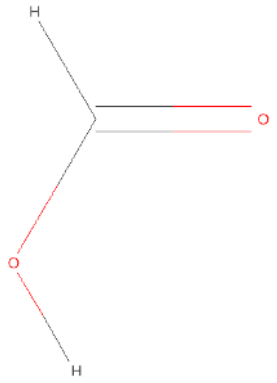
## 4.1 Representation of molecules

The definition of molecular graph (defined in section 2.2.1) can be modified to suit its required use case. For the purpose of chirality determination, adding three-dimensional structure information about the molecule in the form of three Cartesian coordinates for each atom is necessary. In practice, atoms constitute molecules which, when parsed from a source file, have either a name, or some kind of identification (e.g. PDB ID, residue code). Mapping such ID to each atom of said molecule can be useful for sorting out parsed atoms in case there was more than one molecule present in a source file.

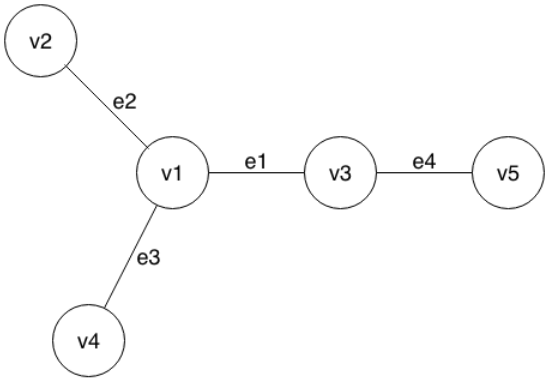
Rather than represent bonds of higher bond orders with several copies of those bonds in $E$, it is more practical to map its bond order to each bond individually. Such mapping can be implemented into algorithms more efficiently that the multiset $E$. Finally, since information about lone electron pairs is of little usability for this use case scenario, it is safe to omit the multiset $L$.

After taking proposed modifications into account, the modified molecular graph is defined as an ordered octuple $G = (V, E, S, I, \delta, \varepsilon, \zeta, \alpha)$ where $V = \{v_1, v_2, ..., v_m\}$ is a non-empty set of vertices, $E = \{e_1, e_2, ..., e_n\}$ is a set of edges, $S$ is a finite set of symbols of elements that are represented among atoms of this graph, $I$ is a finite set of molecule IDs that are represented among atoms in this graph, $\delta : V \to S$ is a relation that gives an element symbol to every atom in this graph, $\varepsilon : V \to I$ is a relation that gives a relevant molecule ID to every atom in this graph, $\zeta : V \to \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ is a relation that gives an ordered triple which represents three Cartesian coordinates to every atom in this graph, and $\alpha : V \times V \to \mathbb{N}$ is a relation that gives bond order to every bond in this graph. An example of a molecule represented by a modified molecular graph is depicted in figure 4.1.

A simpler version of the modified molecular graph has been used in my bachelor's thesis.




(a) Structural formula. Source: LigandExpo.




(b) Modified molecular graph visualization.

$G = (V, E, S, I, \delta, \varepsilon, \zeta, \alpha)$,
$V = \{v_1, v_2, v_3, v_4, v_5\}$,
$E = \{\{v_1, v_3\}, \{v_1, v_2\}, \{v_1, v_4\}, \{v_3, v_5\}\}$,
$S = \{C, O, H\}$,
$I = \{FMT\}$,
$\delta(v_1) = C, \delta(v_2) = \delta(v_3) = O, \delta(v_4) = \delta(v_5) = H$
$\varepsilon(v_1) = \varepsilon(v_2) = \varepsilon(v_3) = \varepsilon(v_4) = \varepsilon(v_5) = FMT$
$\zeta(v_1) = (-0.095, -0.45, -0.001), \zeta(v_2) = (-1.124, 0.213, 0)$,
$\zeta(v_3) = (1.085, 0.218, 0), \zeta(v_4) = (-0.126, -1.5, 0.003)$,
$\zeta(v_5) = (1.881, -0.331, 0)$,
$\alpha(\{v_1, v_2\}) = 2, \alpha(\{v_1, v_3\}) = \alpha(\{v_1, v_4\}) = \alpha(\{v_3, v_5\}) = 1$

(c) Formal description of the modified molecular graph. Source of data: LigandExpo.

Figure 4.1: Modified molecular graph representation of formic acid molecule.

## 4.2 Overview of LiCMP

The implementation part of my master's thesis have resulted in the assembly of program LiCMP. Its name is an abbreviation of Ligand CoMParator.

### 4.2.1 Functionality

LiCMP is an application designed for batch identification of ligands (referred to as ligands to be identified) by comparing them to a set

of reference ligands. It is a CLI[1] program with support for multi-threaded processing. Ligand(s) to be identified, as well as reference ligands, are supplied to LiCMP in the form of PDB files. Input PDB files are checked for accessibility and validity, i.e. whether they adhere to the standardized PDB file format and whether they do not lack any vital information. Full check of the PDB format is not performed though, because LiCMP only needs valid `ATOM` and `HETATM` lines for its computations. If an input file fails any of those checks, application terminates.

Molecular graphs are then constructed as the next step. Bond orders determination and chirality atom recognition is performed as a part of the molecular graph construction process. Extra effort is taken while constructing possible molecular graphs for each ligand to be identified since poor quality of such input ligands is assumed. After constructing molecular graphs, each ligand to be identified is compared to each reference ligand. Pairwise comparison jobs are dynamically dispatched among available cores to split the work efficiently.

Comparison of two ligands is carried out on the level of their molecular graphs. After a trivial check of feasibility, depth first comparison with backtracking that starts at chosen pairs of atoms is computed. Best comparison result is stored and next pairwise comparison with another reference ligand is started.

After a ligand to be identified has been compared with all relevant reference ligands, best result among results obtained so far is chosen and written out into the logfile, that represents the main output method of LiCMP, as the identification of said ligand to be identified. Several other highly similar comparison results are also written out as other likely identification alternatives.

### 4.2.2 Command line parameters

Each batch run of LiCMP can be configured with the use of several command line parameters. Parameters have to adhere to defined order of occurence: `licmp [options] [logfile] [source of reference ligands] [source of ligands to be identified]`.

1. command line interface

Each batch run of LiCMP can be configured with the use of several command line parameters:

- **-i chem_sym** defines a chemical symbol (or more than one, separated by a space) of an element that will be ignored. Atoms of ignored elements are not loaded from source PDB files. When this parameter is not provided, hydrogen atoms are ignored by default.

- **-ex extension** defines a filename extension that will be expected from all source files retrieved through the use of parameters **-r**, **-di** and their recursive variants. Regardless of provided extension, however, all source files are supposed to have inner structure that adheres to the PDB file format.

- **-scTout seconds** defines maximum amount of time that single comparison between a reference ligand and a variant of compared ligand can take. After said amount of time passes, comparison is forcibly canceled and a note about such event is written to the logfile. Default value is 10 minutes.

- **-idTout seconds** defines maximum amount of time that the identification process of a ligand can take. After said amount of time passes, identification is forcibly canceled and a note of unsuccessful identification of such ligand is written to the logfile. Default value is 30 minutes.

- **-thrd number** defines number of computation threads that will be spawned by LiCMP. Default value is the number of logical cores present in user's system.

- **-s** prohibits the program from writing out anything to the standard output and the standard error output, save for messages about unrecoverable fatal errors that prohibit LiCMP from carrying out its tasks.

- **-laf** makes the program write all filenames of source files to the logfile separately from identification results.

- **-roc number** defines how many other identification possibilities should be listed in the logfile for every ligand to be identified. Default number of listed additional identification possibilities is 5.

- **-l name** provides a name of file that will be used by LiCMP as the logfile. This parameter has no default value and is compulsory, as the program does not have any other significant output functionality.

- **-r directory** provides the program with a name of directory from where source files of reference ligands will be taken.

- **-rR directory** has the same meaning as **-r**, but also makes the program look recursively for source files in subfolders of the specified folder.

- **-ch name** provides LiCMP with filename(s) of source files of ligand(s) to be identified.

- **-di directory** provides the program with a name of directory from where source files of ligands to be identified will be taken.

- **-diR directory** has the same meaning as **-di**, but also makes the program look recursively for source files in subfolders of the specified folder.

Command line parameters that specify logfile file name, a source of reference ligands and a source of ligands to be identified are compulsory, the rest is optional. Each occurrence of parameters **-ch**, **-di** and **-diR** spawns one separate ligand identification job. Minimum number of supplied parameters is three (e.g. `licmp -l logfile.txt -r reference -di toBeIdentified`).

### 4.2.3 Program input specification

LiCMP accepts PDB files as the input source of ligands to process. An example of such file is shown in figure 4.2. If an input file contains lines of other types than `ATOM` and `HETATM`, they are ignored.

```
HETATM 3986 C2 BGC A 551 32.131 32.328 40.183 1.00 35.02 C
HETATM 3987 C3 BGC A 551 31.669 31.742 38.863 1.00 38.06 C
HETATM 3988 C4 BGC A 551 32.708 32.144 37.786 1.00 34.53 C
HETATM 3989 C5 BGC A 551 34.127 31.654 38.186 1.00 31.53 C
HETATM 3990 C6 BGC A 551 35.341 31.747 37.253 1.00 27.42 C
HETATM 3991 C1 BGC A 551 33.581 31.957 40.484 1.00 37.10 C
HETATM 3992 O1 BGC A 551 33.955 32.619 41.651 1.00 45.39 O
HETATM 3993 O2 BGC A 551 31.329 31.812 41.198 1.00 32.05 O
HETATM 3994 O3 BGC A 551 30.351 32.162 38.526 1.00 37.26 O
HETATM 3995 O4 BGC A 551 32.278 31.480 36.619 1.00 36.75 O
HETATM 3996 O5 BGC A 551 34.456 32.322 39.403 1.00 37.45 O
HETATM 3997 O6 BGC A 551 35.815 33.065 37.023 1.00 23.98 O
```

Figure 4.2: Example of an input PDB file that contains molecule $\beta$-d-glucose

`CONNECT` lines are also ignored, because there is no guarantee for reliable connection data to be present in source files of ligands to be identified. All input PDB files are checked for validity against the PDB file format standard [39].

Ligands to be identified can have PDB ID of their parent structure associated with them. LiCMP parses this ID from their source file name if it contains the ID between two underscores (ex. file M_1b1y_3996.pdb contains ligand that has been extracted from parent structure with PDB ID 1B1Y).

### 4.2.4 Program output specification

Results of ligand identification process are written to the logfile. For every input ligand to be identified, an identification record is written to the logfile. Each record contains main identification result (aka comparison result with highest score that has been achieved) and several other identification possibilities (five in default, their count is changeable by user). An example of a record can be found in figure 4.3. First line of a result holds summary of encountered differences between ligand to be identified and reference ligand that it had been identified with. The rest of lines contain description of encountered differences.

On the summary line, there is this information (from left to right):

- filename of the ligand to be identified

```
C:/LiCMP/compsw/1/M_1her_6317.pdb; 1cjp  ; MUG ; MUG ; 100 %;   0 ;   0 ;   0 ;   0 ;   0 ;   0 ;   0 ;
C:/LiCMP/compsw/1/M_1her_6317.pdb; 1cjp  ; MUG ; B7G ;  87 %;   0 ;   5 ;   0 ;   0 ;   0 ;   2 ;   1 ;
   - excessive atoms:
                          CM4  C5   C6   O2   O1
   - bond grade mismatches:
                            C3 --   C4 is a double bond in compared molecule, while   C3 --   C4 is a single bond
                            in reference molecule.
                            C2 --   C3 is a double bond in compared molecule, while   C2 --   C3 is a single bond
                            in reference molecule.
   - chirality mismatches:
                          Comp. atom: C1' with ligand O5' has + chirality vs. ref. atom: C1 with ligand O5 has -
                          chirality
C:/LiCMP/compsw/1/M_1her_6317.pdb; 1cjp  ; MUG ; END ;  86 %;   3 ;   3 ;   0 ;   1 ;   3 ;   0 ;   0 ;
   - missing atoms:
                          CAT  OAC  OAO
   - excessive atoms:
                          O2   O2'  O3'
   - missing bonds:
                           CAU --  CAJ
   - excessive bonds:
                           C4A --   C4
                            C5 --  C4A
                           O5' --  C1'
```

Figure 4.3: An example of a identification result record

- PDB ID of parent structure of the ligand to be identified
- three-character alphanumeric ID of the reference ligand
- three-character alphanumeric ID of the ligand to be identified
- similarity percentage of both ligands
- number of atoms that are present in the ligand to be identified but are missing from the reference ligand
- number of atoms that are present in the reference ligand but are missing from the ligand to be compared
- number of atoms that could not have been connected to any variant of the ligand to be identified
- number of bonds that exist in the reference ligand but are missing from the ligand to be compared
- number of bonds that exist in the ligand to be identified but are missing from the reference ligand
- number of bonds in the ligand to be identified that differ in their order from paired chiral atoms in reference ligand

- number of chiral atoms in the ligand to be identified that differ in their chiral configuration from paired chiral atoms in the reference ligand

## 4.3 Data structures

In this section, the most important data structures used in LiCMP are discussed together with their benefits, drawbacks and reasoning for their utilization. The STL of C++ as well as Qt SDK have several container types each. Functionality of many of them overlaps, but computational complexity of operations differs among container types. Choosing the right container for a usage scenario by analyzing computation complexity of its most used operations can reduce overall computation complexity of whole algorithms.

### 4.3.1 QList container

The most commonly used container type in LiCMP is QList[2] from the Qt SDK. It is an amalgamation of vector and linked list container types. Internally, QList is represented by an array of pointers to objects that have been allocated on the heap. Because of such inner representation, direct access to an element by its index is done in $O(1)$ [47] time. The inner pointer array is allocated during container construction as an adjacent portion of memory with a little extra space at the beginning and at the end of the array. Therefore, adding and deleting items from the beginning and the end of the container is done in amortized $O(1)$ [47] time as well.

Complexity of inserting a new element in the middle is at worst on par with inserting to an array type container filled with void pointer objects. Because of this disadvantage, no inserting in the middle of a QList container is ever performed in LiCMP.

2. QList documentation: `http://qt-project.org/doc/qt-5/QList.html`

### 4.3.2 QMap and QMultiMap containers

QMap[3] is an associative container that stores information in pairs of a key and a value. Since the main focus of this container is fast retrieval of values using their associated keys, inner structure of QMap is based on a red-black tree data structure. Furthermore, items in QMap container appear as sorted in ascending order by their keys when being iterated through. Each key in QMap has to be unique (unlike in QMultimap[4], where multiple pairs of keys and values with the same key are allowed). Asymptotic complexity of insertion and lookup of elements is $O(log\ n)$ [47] where $n$ stands for number element pairs that are already present in the container.

In LiCMP, QMap and QMultiMap fill two main roles:

- storage containers for atoms that have been parsed from source files (in this scenario, key is element symbol of paired atom, which facilitates retrieval of atoms of required elements)
- auxiliary containers that serve as search structures as well as automatically sorted structures - they pair pointers to objects existing elsewhere (as values) with keys by which the container should be sorted by and with which it will be queried

### 4.3.3 QSet containers

Cetrain algoritms require retrieval of a subset of given multiset in its mathematical meaning - all elements in retrieved set have to be unique. Set containers (set from STL and QSet[5] from Qt SDK) are suitable for this use case because only one occurrence of each member inside them is allowed, no matter how many times is the same value inserted into the container. Main difference between both set containers is that STL set is always sorted in ascending order, while QSet is kept unsorted.

QSet is internally represented by hash table. Asymptotic complexity of insertion and lookup of elements is $O(n)$ where $n$ stands for the number of elements that are already present in the container.

---

3. QMap documentation: `http://qt-project.org/doc/qt-5/QMap.html`
4. QMultiMap documentation: `http://qt-project.org/doc/qt-5/QMultiMap.html`
5. QSet documentation: `http://qt-project.org/doc/qt-5/QSet.html`

QSet, as well as QList, allocates more memory for its hash table than needed as the time of last allocation (or reallocation) to decrease the number of necessary reallocations when the container grows.

## 4.4 Class diagram

Brief class diagram of classes and their relationships is depicted in figure 4.4. Full class diagram can be found on the attached CD. LiCMP consists of 21 classes. In total, there are 113 attributes (106 private and 7 protected) and 236 methods (43 private, 2 protected and 191 public).

## 4.5 Significant classes

### 4.5.1 `GraphAtom` class

Class `GraphAtom` is a direct implementation of the modified molecular graph that has been designed in section 4.1. It is responsible for the core functionality of LiCMP: Building molecular graphs and exploring possible matchings between common subgraphs of two molecular graphs. It represents single atom that has already been placed into a molecular graph. Therefore, it contains all chemical information about such atom (parsed from source PDB file and represented via `Atom` container class instance), as well as pointers to adjacent connected atoms in the graph. Since the above mentioned core functionality has been implemented using recursive algorithms, each `GraphAtom` class instance has to also hold its state relative and relevant to the current algorithm that iterates over the graph.

Method `placeConnectedAtom` is an implementation of the first key recursive algorithm that is called during the life of a `GraphAtom` class instance. It checks if an atom, represented by an `Atom` class instance and supplied as a parameter, can be connected to this existing `GraphAtom`. If yes, it is transformed into a newly dynamically allocated `GraphAtom` class instance (if it has not been transformed already) and a new `Bond` class object is created after deciding the order of the new bond in method `determineBondGrade` by comparing its length to a list of predefined lengths of bonds between atoms of

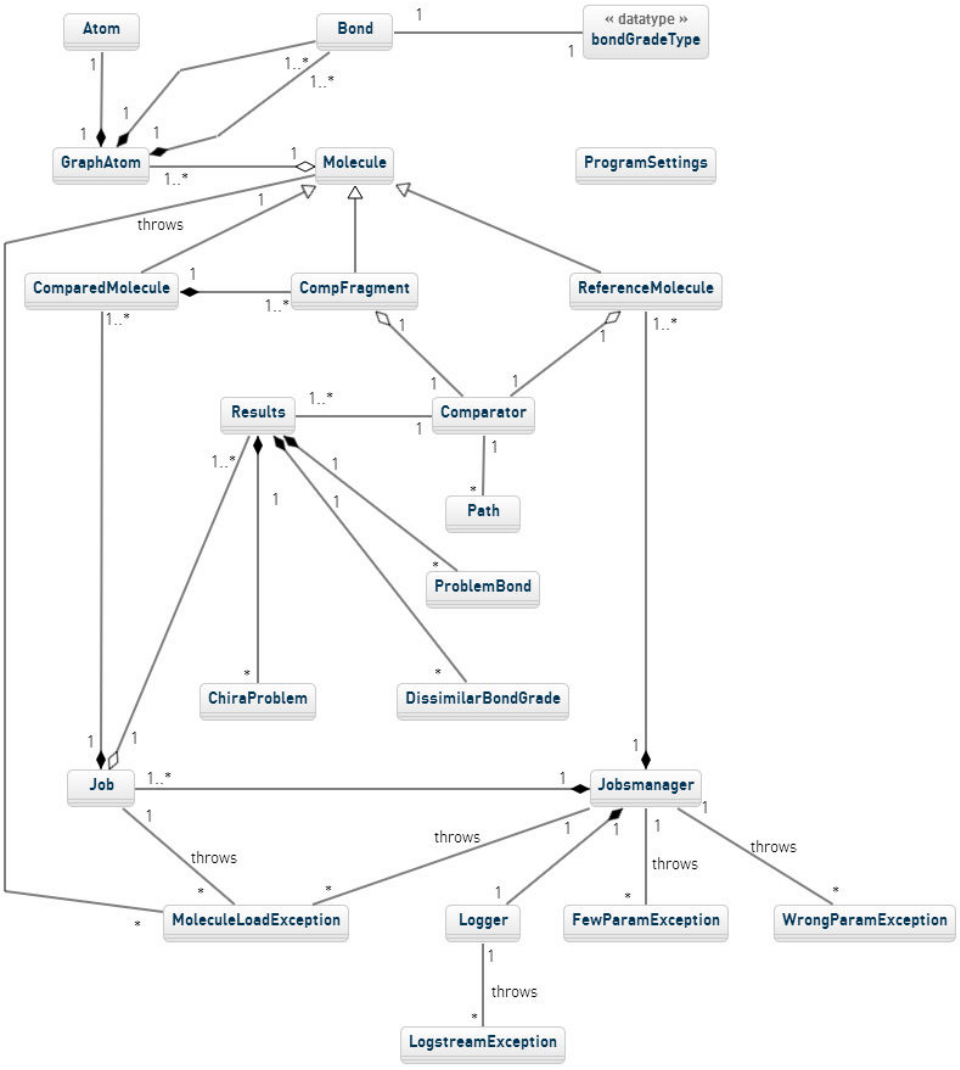



Figure 4.4: Brief class diagram of program LiCMP

supported elements (currently supported elements are carbon, nitrogen and oxygen). The bond order itself is represented by custom enu-

merative type `BondGradeType`. Method `placeConnectedAtom` is then called on all connected atoms that have not worked with the new placement candidate yet.

### 4.5.2 `Molecule` class and its children

Each molecule, that has been loaded from a source PDB file, is represented in LiCMP by a class from the `Molecule` class family. Aside from their common predecessor, class `Molecule`, there are three distinct classes that represent three kinds of molecules that play a role during the process of ligand identification: Reference ligand molecule, molecule of ligand to be identified and fragment molecule of ligand to be identified.

The common predecessor, class `Molecule`, holds attributes and methods that are common for all of its children. Such attributes include list of atoms that belong to this molecule (along with support containers that facilitate operations with the atom list) and pointer to a `GraphAtom` class instance that is a part of molecular graph of this molecule. Method `loadPdbData` (used in child classes `ReferenceMolecule` and `ComparedMolecule`) takes a reference to an input stream, connected to a source PDB file, and extracts all information about atoms of the input molecule which it then stores in appropriate container attributes. Method `buildMolGraph` (used in `ReferenceMolecule` and `CompFragment` classes) iteratively builds molecular graph from the list of atoms of this molecule, one at a time. Code of molecular graph building algorithm is split between this method and recursive method `placeConnectedAtom` of class `GraphAtom`.

Reference ligands are represented in LiCMP by class `ReferenceMolecule`. Construction of molecular graphs for reference ligands is much easier than construction of molecular graphs for ligands to be identified, because reference ligands are expected to be downloaded from a credible source (like LigandExpo) and therefore to be valid. On the contrary, input ligands to be identified (represented in LiCMP by class `ComparedMolecule`) can be defective in several major ways, one of them being a situation where two (or more) ligands overlap and occupy the same space. When ligands overlap, LiCMP tries to separate overlapping molecules via method

`findFragments` of the `ComparedMolecule` class and creates new `CompFragment` class for each harvested ligand (more detailed information about treatment of input ligands to be identified can be found in section 4.8). Each extracted ligand is then guaranteed to have similar level of validity as a reference ligand, and therefore can have its molecular graph constructed using default implementation of method `buildMolGraph`.

### 4.5.3 `Job` class

In LiCMP, a job is a list of ligands to be identified that have been supplied by user, along with a reference to a list of reference ligands that will be used to identify supplied ligands. Class `Job` manages processing of a single job from loading ligands to be compared, through delegation pairwise comparisons to worker threads, to evaluating identification results and writing them to the logfile. After receiving list of ligands to be identified through the constructor, method `loadComparedDatabase` is called to load them from their source PDB files and create `ComparedMolecule` class instances for them.

To start the pairwise comparison process, worker threads and associated `Comparator` class instances have to be allocated by the `processJob` method. To communicate with `Comparator` class instances, class `Job` uses the Qt signal/slot system [44]. Every time a pairwise comparison of two molecules is finished, `Comparator` instance emits signal `pushCompResult` that is caught by slot `receiveResult` in `Job` class instance. This slot method records comparison results and stores them for identification results analysis, and either dispatches new comparison, or waits until all threads have completed their assigned comparisons. Then, it writes suitable identification results to the logfile and calls slot method `startNextComparison` to start the identification process of next ligand to be identified.

### 4.5.4 `Comparator` class

The ligand identification process in LiCMP is threaded, i.e. multiple pairwise comparisons between `ReferenceMolecule` and `CompFramgent` class instances are carried out on worker threads at

any given time. Pairwise comparisons are conducted by class `Comparator`. It contains necessary class instances to perform the comparison (local instance of class `CompFragment` and a pointer to an instance of `ReferenceMolecule` class owned by class `Joblist` - only one instance of each reference ligand exists in LiCMP at a time), that have been loaded to the Comparator class instance by calling methods `receiveCompMol` and `receiveRefMol`. Slot method `compare` launches the comparison itself after receiving signal `kickoffComparison` from a `Job` class instance with supplied ID that either matches ID of this `Comparator` class instance, or is relevant to all `Comparator` class instances if the supplied ID is -1.

### 4.5.5 `Path` class

Class `Path` is the implementation of container class that contains the pairing itself as well as attributes relevant to the pairing that is being currently constructed, but after it is completed, they are no longer relevant. A completed pairing is either a pairing that cannot be expanded anymore, or a pairing that has been superseded by successors when a branchpoint is solved. Attributes relevant to the pairing, that is being currently constructed, include list of startpoints and list of branchpoints. Both new terms (startpoint and branchpoint) are explained in section 4.6. Each pairing is uniquely identified by the path code - a string of characters. Length of the path code is equal to the number branchpoints minus one that have been solved before this Path class construction. Methods of class `Path` are, with the exception of method `createChild` that creates a child instance of this `Path` class instance, various getter and setter method.

### 4.5.6 `Results` class

Each pairwise comparison of two ligand molecules in LiCMP produces single Results class instance that contains all information that suffice for it to be able to assembly a string containing identification results that can then be written out directly to the logfile. The constructor of this class requires not only pointer to the pair of compared ligands, but also pointer to a `Path` class instance. That is because class Results contains functionality for detecting missing bonds, ex-

cessive bonds (method `crawlCheckMissingBonds`) and paired atoms with dissimilar chirality (method `crawlCheckChirality`). Detection of aforementioned problems is best done after a pairing is composed, and after it is completed, similarity percentage of compared ligands is computed using metric described in section 4.9.

### 4.5.7 Container classes

Class `Atom` represents single atom that has been extracted from a parsed source PDB file. It effectively contains information of a whole line of the PDB file, sorted into appropriate attributes.

Classes `ProblemBond`, `DissimilarBondGrade` and `ChiraProblem` each represent a problem that has been discovered while searching through computed pairing for dissimilarities. Such problems are finally stored in a `Result` class instance, counted and (if the result is good enough) written out. Class `ProblemBond` represents a single bond and is defined by names and elements of two atoms that it connects. In LiCMP, it is used to specify a missing or an excessive bond. Class `DissimilarBondGrade` serve as a container for a single occasion when two paired bonds differ in their bond orders. It is defined by two bonds (via `ProblemBond` class instances) and their bond orders. Class `ChiraProblem` is constructed when there is a need to describe dissimilar spatial configuration of two paired chiral atoms and their paired ligands. It is defined by the two pairs of bound atoms in each molecule (again via `ProblemBond` class instance) and their chirality.

### 4.5.8 Singleton classes

Class `ProgramSettings` is a singleton class (for information about the singleton software design pattern, see section 3.7), implemented as a static class, that holds useful information which greatly influence behavior of LiCMP. Some information is loaded into its instance at the beginning of LiCMP execution (information such as table of default element valence diameters, table of valence diameters of possible bond orders of supported elements, or table of scores for selected most common elements that are a part of metric described in section 4.9), while the rest is comprised of settings supplied to the program

by the user via command line parameters along with default values of such settings, so that the program behaves in a consistent manner when the user uses only a minimum set of command line parameters.

Class `Logger` is a wrapper class for the output stream that is connected to a logfile. While not implemented as a singleton class in the strictest meaning of the term, only one instance of class `Logger` is needed and used throughout the program, since LiCMP uses only one logfile to output identification result. From extensibility viewpoint, functionality of class `Logger` can be enriched (e.g. sorting identification results into different output logfiles) when the demand arises.

## 4.6 Comparison algorithm

The comparison algorithm of LiCMP is based on finding the best isomorphism between largest possible subgraphs of molecular graphs of compared molecules by extending already existing isomorphism with suitable atom pairs found during synchronized depth-first search (DFS) traversals of both graphs.

First step in finding isomorphism between two graphs is to determine starting pairs of atoms from where the isomorphism will grow. Without any prior knowledge about which atom from the reference ligand should be surely paired with one concrete atom from the ligand to be identified, blind testing of all atom pairs, made of atoms of same element type, is required. Such algorithm is, however, computationally unfeasible, since the number of possible pairings to explore would rise in quadratical proportion to the number of atoms in both graphs. To reduce the graveness of above mentioned limitation, only atoms of the least represented element type among both graphs are taken into consideration as starting pairs for new isomorphisms (algorithm 1). If two (or more) element types are represented in the graph by the lowest number of atoms, one element type is chosen randomly, as no specific element-based distinction is required. This solution is feasible, since larger ligands tend to have less common elements in their graphs in very low numbers, while pairing for small ligands is computed in very short time.

**Algorithm 1:** determineStartingElement

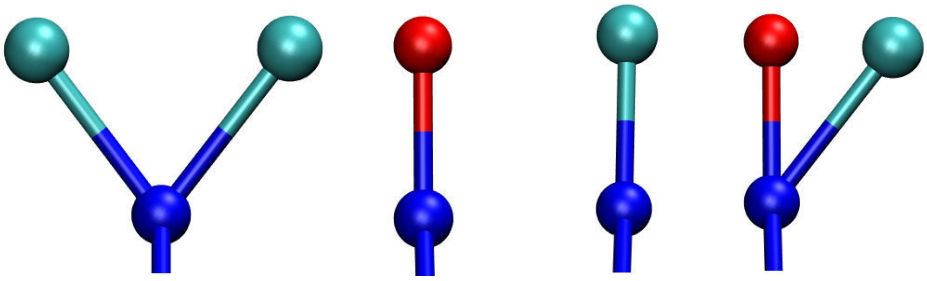
**Data**: Pointers to reference ligand and identified ligand
**Result**: Element which atoms will serve as first pairings' members

```
1  refElems ⟵ Distinct element symbols of atoms of reference ligand ;
2  compElems ⟵ Distinct element symbols of atoms of identified ligand ;
3  chosenElem ;
4  chosenElemCount ⟵ ∞ ;
5  foreach elem ⟵ compElems do
6      if refElems ∈ elem then
7          refElemCount ⟵ number of atoms of element elem in reference
           ligand ;
8          compElemCount ⟵ number of atoms of element elem in
           identified ligand ;
9          if chosenElemCount < (refElemCount + compElemCount) then
10             chosenElemCount ⟵ (refElemCount + compElemCount) ;
11             chosenElems ⟵ elem ;
12         end
13     end
14 end
```

Algorithm 1 also plays key role in deciding whether the isomorphism discovery process will be carried out, since if there are no atoms of the same element type in both graphs, no chemically meaningful isomorphism can be created. Aside from that, there are no other preprocessing rules in the comparison algorithm.

Each proposed isomorphism is then extended using new pairings made of usable atoms from each graph that are connected to atoms of an already established pairing (algorithm 2). During each extension step, feasible atoms are grouped by their element types to be processed together since atoms of dissimilar element type can not be paired together. Then, possible new pairings are assessed. Usable atoms are atoms that are not yet a part of a pairing of the currently extended isomorphism. Based on the count of usable atoms, there are three possible situations that can appear while choosing next pair of atoms to extend the currently constructed isomorphism, as illustrated in figure 4.5.

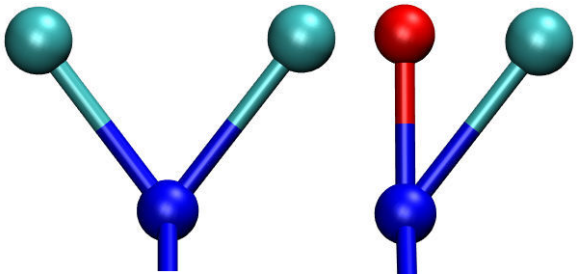
When there are no available atoms connected to either atom of established pairing, no new pairings are created (situation 0). When there is exactly one atom connected to each atom of established pairing (situation 1), new pairing is constructed, using those two mentioned atoms, and is then immediately explored for next new pairing possibilities before creating all possible new pairings connected to the established pairing (DFS principle). When there is more than one usable atom connected to one of the atoms in the established pairing while there is at least one usable atom connected to the other atom of the established pairing, no new pairing is established, since it is impossible to decide the optimal pairing amongst those usable atoms. The established pairing is marked to be solved later.



(a) Situation 0.



(b) Situation 1.



(c) Situation N.

Figure 4.5: Three possible situations that can appear while choosing next pair of atoms to extend the currently constructed isomorphism Blue atoms are atoms of already established pairing, while red and green atoms are candidate atoms for new pairing formation. Atoms of the same color are of the same element type.

**Algorithm 2:** extendIsomorphismDFS

**Data**: Pointers to atoms of established pairing - atom from reference molecule ($RA$), atom from identified ligand ($IA$), pointer to the currently extended isomorphism $ISO$

```
1  elemToProc ;
2  foreach elem ⟵ element symbols of usable atoms connected to RA do
3      if IA has at least one usable atom of element elem connected then
4          elemToProc ⟵ elem ;
5      end
6  end
7  foreach elem ⟵ elemToProc do
8      if RA or IA has more than one usable atom of element elem then
9          mark the pairing of RA and IA for solveBranching algorithm
           usage ;
10         continue;
11     end
12     newRefAtom ⟵ pointer to the only usable atom of element elem
       connected to RA ;
13     newCompAtom ⟵ pointer to the only usable atom of element elem
       connected to IA ;
14     newPair ⟵ (newRefAtom, newCompAtom) ;
15     add newPair to ISO ;
16     extendIsomorphismDFS(newRefAtom, newCompAtom, ISO) ;
17 end
```

After growing current isomorphism as much as possible by solving situations 0 and 1, it is necessary to solve situations 2 to further enlarge the isomorphism. First, an established pairing with the smallest number of connected atoms to pair (or to reject and leave unpaired - this happens when one atom of processed pairing has more usable atoms of an element than the other) is selected. Then, Cartesian product of usable atoms connected to both atoms in the established pairing is computed. The product is defined as follows: Suppose there exist sets of usable atoms $A$ and $B$ from which atoms that will constitute new pairings will be chosen. The result of this Cartesian product is a set $\chi$ that is defined as

$$\chi = \{X_{1...n} | p = min(|A|, |B|), q = max(|A|, |B|), n = \prod_{i=0}^{i<p} (q - i)\} \quad (4.1)$$

where $X$ are sets defined as

$$X = \{(a_x, b_y) | 0 \leq x \leq |A|, 0 \leq y \leq |B|\} \quad (4.2a)$$

$$|X| = min(|A|, |B|) \quad (4.2b)$$

$$X \in (a_x, b_y), X \notin (a_x, b_m), m \neq y \quad (4.2c)$$

$$X \in (a_x, b_y), X \notin (a_n, b_y), n \neq x \quad (4.2d)$$

New isomorphisms are then created from the currently processed one. They have parent-child relationship. Each child differs from its parent at the time of its creation only by the inclusion of new pairings from a set $X$. For each set $X$ a child isomorphism is created. LiCMP then takes steps to enlarge each child isomorphism in the same way as described above, while discontinuing the parent isomorphism.

After all established isomorphisms have been extended as much as possible, they are analyzed for deficiencies. Both compared molecular graphs are traversed and unpaired atoms are discovered and noted. Each isomorphism is then traversed, pairing by pairing, and bond discrepancies (missing bonds in each graph as well as dissimilar bond orders of paired bonds) as well as chirality problems are

discovered and recorded. Similarity percentages are computed for each isomorphism using metric described in section 4.9 and the best one is preserved and returned.

## 4.7 Parallelization of molecule comparison

To identify a ligand, LiCMP has to compare it to each relevant member of some reference ligands set that can span hundreds of molecules, which means hundreds of pairwise comparisons just to identify a single ligand. In practice, thousands of ligands have to be identified and validated as a part of a single batch task. The total number of comparisons can therefore be in millions, thus making the batch identification process a very lengthy and impractical for real world usage.

A significant increase in performance of batch ligand identification can be achieved by parallelization of pairwise comparisons. Each pairwise comparison is independent of another one, as long as no ligand is compared with more that one other ligand at the same time. This requirement can be fulfilled very easily. In LiCMP, the currently compared molecule is copied for each thread (represented by a `Comparator` class instance) to have its own copy. Then, unique reference ligands are assigned to each thread and the comparison process is started. When a thread completes its task, it sends the comparison result to the coordinator object (`Job` class instance) and in exchange receives another reference ligand which it then starts comparing with its designated copy of ligand to be identified. After all required molecule comparisons are carried out, copies of next ligand to be identified are distributed amongst threads and the process starts anew.

## 4.8 Problematic input molecule processing

Certain percentage of ligands to be identified are stored in their source files in an invalid, chemically nonsensical, state. Such issue stems from the usage of automatic methods that extract standalone ligands from large complexes where they are in bound state with a biomacromolecule. Most common problems of said ligands is the

presence of excessive unrelated atoms (that belonged to either different ligand, or to the biomacromolecule itself) in their files, loss of required atoms, and the presence of two (or more) ligands that overlap the same space (that represent several versions of the same ligand, but the automatic method deemed them to be standalone molecules). Two examples of such malformed ligands are pictured in figure 4.6. If the naive molecular graph construction algorithm was applied to ligand that suffers from one of the above mentioned problems (the third in particular), constructed graph would be of unsatisfactory quality. Therefore, it is necessary to treat input ligands to be identified differently.

First, the advanced molecular graph construction algorithm divides atoms from input PDB file by the residue name (represented by the residue code) that they belong to (not by residue number - the assumption, that there should be only one residue per one input PDB file, stands). Then, the graph growth process starts from the first atom of each discovered residue, one at a time (not in parallel). Newly created graph grows by one atom from currently processed residue. To determine whether an atom is in bonding distance of another atom, covalent radii (see section 2.1.2 for associated theory) of both atoms are summed and a tolerance length of 0.4 Å is added to the sum. If the sum is lesser then the distance between said two atoms, their potential bond is taken into consideration.

If there are no more suitable atoms, the algorithm tries to grow the graph using atoms from other residues. After connecting one such atom to the graph, it checks whether there are any other atoms from the same residue that can be connected to the graph via the newly connected foreign atom. Each candidate for addition to the graph is checked whether it is not too close (i.e. closer than the 0.7 Å threshold) to any atom that comprise the currently constructed graph. If it fails this test, it will never be connected to the current graph.

When no more atoms can be connected to the newly created molecular graph, it is established as a standalone compared molecule fragment that will later be pairwise compared to each reference ligand. Input ligand type of fragment (represented via its residue code) is determined by majority rule: The type is established as a type of ligand which most atoms of this new fragment belong to.

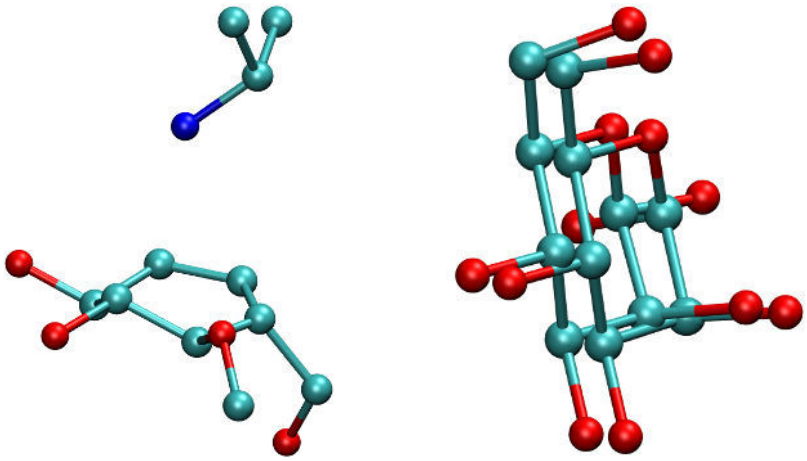


Figure 4.6: Two examples of malformed ligands: A ligand with unrelated atoms (left) and two ligands that overlap each other (right).

If there are any atoms of the currently processed residue that have not been placed in a graph yet, construction of new fragment begins again. If there are no usable atoms of current residue, LiCMP starts processing atoms of another residue as long as there exist atoms that have not yet been placed into any graph of a fragment.

Bond order determination is a straightforward task. When a connection is being established between two atoms of supported element types[6], their distance is computed and compared to a set of reference fonds with predefined bond lengths. Bond order of a reference bond which length is the closest to the length of the newly created bond is taken as the determined bond order.

To decrease the amount of false bond order dissimilarities, the concept of fuzzy bonds is proposed. Along expected bond orders (single, double, triple), additional artificial bond orders have been placed in the reference fist of bond orders between the already existing ones (fuzzy single bond order between single and double bond orders, and fuzzy triple bond order between double and triple bond orders). When a bond of fuzzy order has its order compared to another bond, it acts as a bond of both orders that its fuzzy order has been designed to bridge. To give an example, a double bond is compared with fuzzy single bond. These two bonds are deemed similar,

6. carbon, nitrogen and oxygen

because for the purposes of this comparison, the fuzzy single bond acts like a bond of double bond order.

## 4.9 Molecule similarity metric

To assess the similarity of two ligands, a similarity metric is required. Such metric should take into consideration sizes of compared molecules (expressed in counts of their elements) as well as element types of elements that compose each molecule. Each type of problem (e.g. missing atom, excessive bond), discovered during the pairwise comparison process, should carry different weight in the enumeration of graveness of comparison problems.

With accord to specifications listed above, a new molecule metric for the LiCMP program has been developed. It is a metric that sums all problems discovered during the comparison process, computes their integer representation and converts it to a similarity percentage with respect to sizes of reference ligand and ligand to be identified. The similarity percentage of two ligands is computed using formula 4.3.

$$S = 100 - \frac{S_{res}}{S_{ref} + S_{in}} * 100\% \tag{4.3}$$

where:

- $S$ stands for the similarity percentage
- $S_{res}$ stands for the enumerated problem score
- $S_{ref}$ stands for the score of the reference ligand
- $S_{in}$ stands for the score of the ligand to be identified

After the construction of molecular graph is completed, newly created ligand is assigned its score that will be used in formula 4.3 as $S_{ref}$ or $S_{id}$. Score for each ligand is computed as sum of scoring values of all individual atoms that are part of the newly assembled molecular graph. Values for atoms depend on their element types, and are as follows:

- 1 for hydrogen atoms
- 2 for oxygen atoms
- 3 for nitrogen atoms
- 4 for carbon atoms
- 6 for every atom of element not mentioned above atoms

After the completion of evaluation of a pairwise comparison, sets of problems are returned as comparison results. Returned sets need to have their contents enumerated, so that formula 4.3 can be applied and similarity percentage of two ligand can be returned to the user. During the enumeration process, each discovered problem is given a value. All values of all problems are then summed. Problem types and their values are:

- 1 for single discovered spatial configuration discrepancy of paired chiral atoms
- 1 for each couple of paired bonds that have dissimilar orders
- 1 for every bond that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand
- 1 for every hydrogen atom that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand
- 2 for every oxygen atom that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand
- 3 for every nitrogen atom that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand
- 4 for every carbon atom that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand

- 6 for every atom of element not mentioned above that is either missing from the ligand to be identified, or is excessive there, with regards to the reference ligand

Unconnected atoms are not summed into the problem score, because if they are relevant, they are already represented in either missing, or excessive atoms set.

# 5 Results and discussion

## 5.1 Ligand dataset

The input set of ligands to be validated and annotated consists of 10 247 molecules. These ligands have been extracted from various proteins from PDB as recurring motives as a part of the metalloprotein research project at the National Centre for Biomolecular Research [4]. All input PDB files were deemed valid by LiCMP and a valid ligand to be identified has been extracted from each of them using algorithm in section 4.8. A short table of the most common ligands along with their count present in this input set as well as in the result set is in figure 5.1.

| ligand code | count |
|---|---|
| FMN | 916 |
| ATP | 690 |
| ANP | 426 |
| HOH | 402 |
| ADP | 393 |
| BOG | 287 |
| GTP | 255 |
| ASP | 221 |
| MG | 193 |
| DMU | 178 |

| ligand code | count |
|---|---|
| FMN | 919 |
| ATP | 694 |
| O | 443 |
| ANP | 425 |
| ADP | 391 |
| HOH | 383 |
| BOG | 284 |
| GTP | 259 |
| TPP | 202 |
| DMU | 177 |

Figure 5.1: Ten most numerous ligand types present in the input set of ligands to be identified (left) and ten most numerous ligand types present among computed annotations of said ligands.

Input ligands have been compared to reference set that consisted of 1 865 ligands. Said reference ligands have been extracted from LigandExpo on the $9^{th}$ May 2014 in their ideal forms. As expected, all reference ligands have been deemed valid by LiCMP.

## 5.2 Identification results

Out of the input set of ligands to be identified, 29 ligands timed out completely and LiCMP was not able to annotate them (more information about limitations of LiCMP is in section 5.5). 10 218 ligands remained, and out of this set, 5 346 ligands have been annotated with total confidence (i.e. have achieved 100 % as their similarity percentage with at least one molecule from the reference set). 2 153 ligands have been annotated with greater similarity percentage than 94 %. Therefore, 2 719 ligands have been annotated with lower final similarity percentage than 94 %. 19 ligands have lower similarity percentage than 94 % because their comparison with some of the reference ligands have timed out (more information about local time outs is in section 5.5).

After removing all timed out entries from the set of annotation results, 10 153 annotations of ligand molecules have remained in the result set. Out of this group, 4 847 ligands (47.74 %) have been annotated with less that 100 % similarity percentage (as seen in figure 5.2), while 2 701 input ligands to be identified have been annotated with lower that 94 % similarity score. Figure 5.3 shows the distribution of similarity scores of annotation results. Such result indicates that the input ligand set is of relatively low quality with input ligands that suffer from various problems. 8 684 ligands have had their annotations confirmed. This statistic shows that input ligands are correctly annotated in most cases, however many of them are not stored in their source files in valid states.

## 5.3 Differing patterns

Out of the resulting set of 10 218 identified ligands, 3 547 specimens have in issue in the number of atoms that are present in their source files. The most common problem was the presence of an excessive magnesium atom. An example of this issue, that arose because of wrong association of the magnesium atom to the ligand by the automatic ligand extraction application, is depicted in figure 5.4.

Other atom-related issues of minor occurrence counts include excessive $\alpha$-C carbon presence (figure 5.5), other excessive metal at-

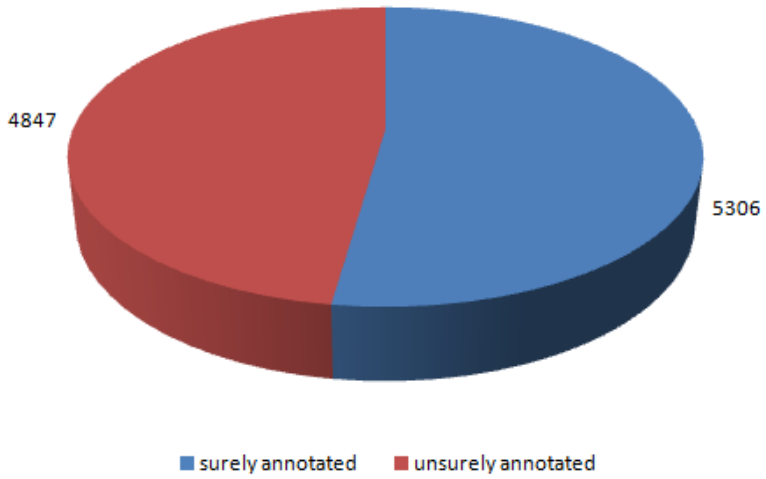



Figure 5.2: Pie chart of ligand annotation results.

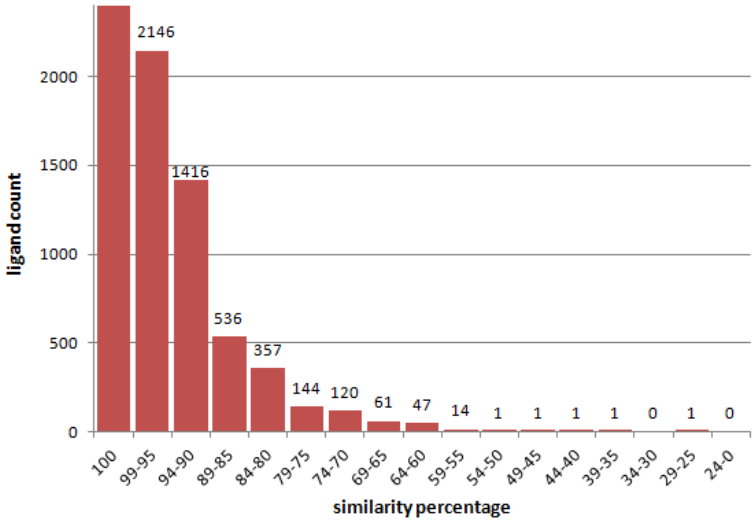



Figure 5.3: Distribution of similarity scores of ligand annotation results.

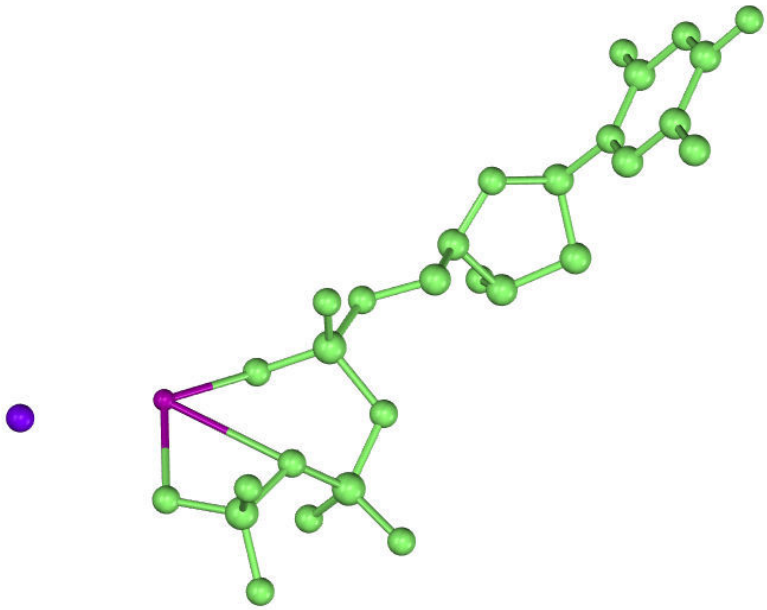


Figure 5.4: Example of a ligand that has an excessive magnesium atom (in magenta), as well an unconnected oxygen atom (in dark violet), in its source file.

oms presence, the inclusion of fragments of ligands and amino acids adjacent in the source biomacromolecule and single missing oxygen atom; and the lack of various atom groups ranging from a single oxygen to an entire cyclohexane ring (figure 5.6) to more than half of the original ligand (figure 5.7).

Regarding chirality issues of input ligands to be identified, they can be divided into two groups. First group contains ligands that have wrong chirality configuration on at least one carbon stereogenic center. An example of such such ligand is depicted in figure 5.8). Ligands to be identified from the second group have some connected atoms at some of their stereogenic centers on the level of the chiral carbon and its immediate surrounding and the chirality comparison algorithm therefore cannot discern if such atom is above or below the plane of the center and its surroundings. An example of this occurrence is depicted in figure 5.9).

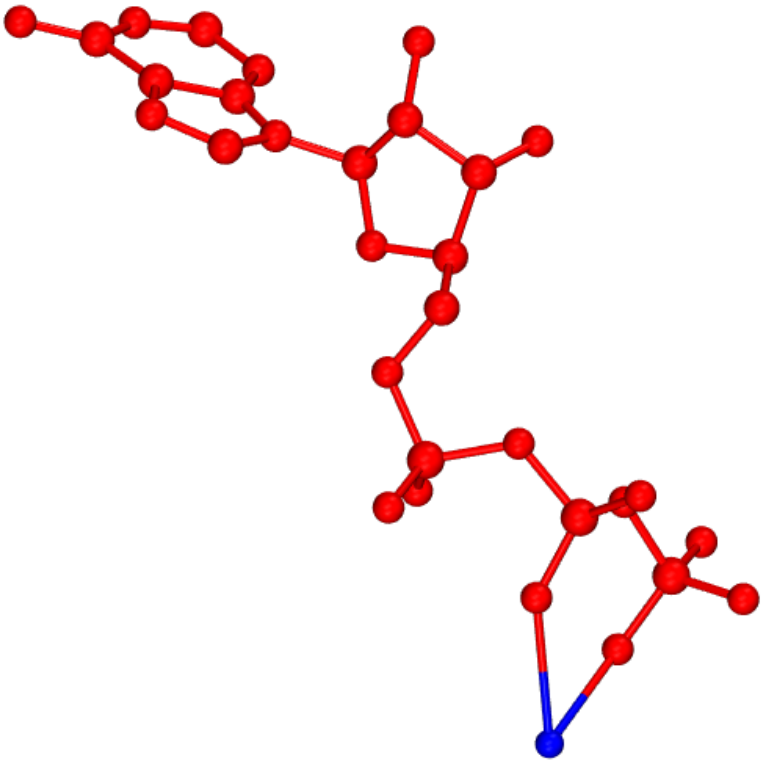


Figure 5.5: Example of a ligand that has an excessive $\alpha$-C carbon atom (in blue).

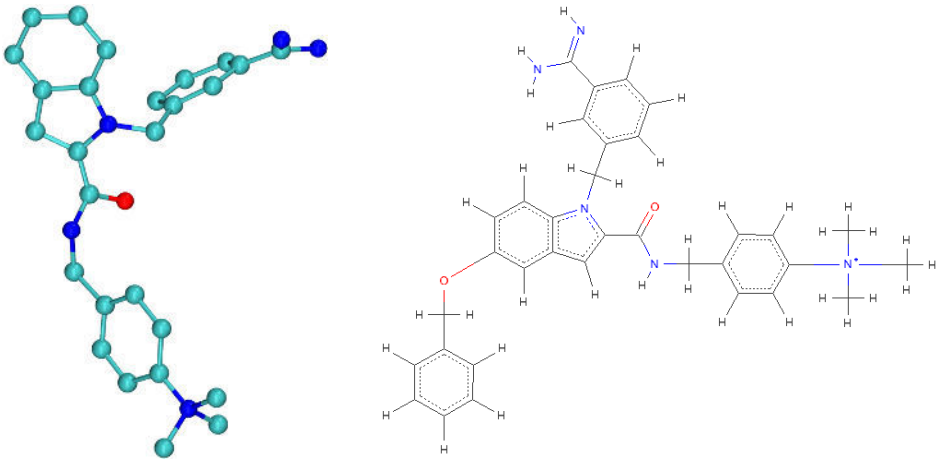


Figure 5.6: Example of a ligand that lacks an entire cyclohexane ring (left) and its counterpart from the reference set (right, source: LigandExpo).

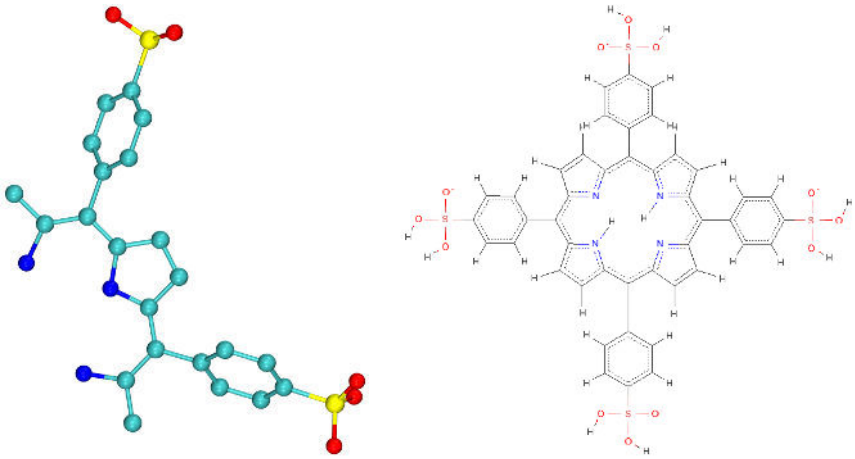


Figure 5.7: Example of a ligand that lacks more than half of atoms that it is expected to have (left) and its counterpart from the reference set (right, source: LigandExpo).

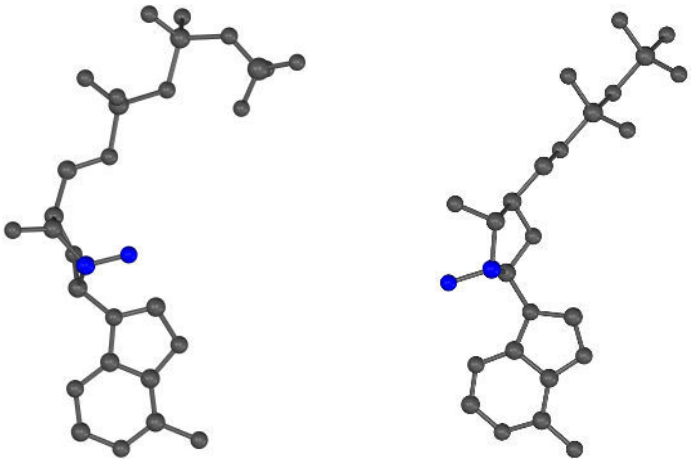


Figure 5.8: Example of a ligand that has wrong chiral configuration on a carbon stereogenic center (left, problematic center and its connected atom in the wrong position are in blue) and its counterpart from the reference set (right, paired stereogenic center and its connected atom are in blue).

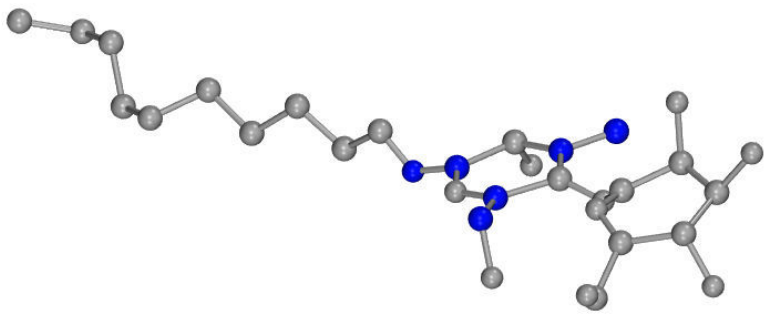


Figure 5.9: Example of a ligand that has three chiral stereogenic center with three connected atoms (in blue) that are considered to be on the level of the chiral carbon and its immediate surrounding.

## 5.4 Comparison with SwCMP

To assess implementation differences of LiCMP from the implementation of SwCMP, both tools have been tested on the same set of saccharide molecules.

### 5.4.1 Sugar dataset

The input set of saccharide molecules has been first introduced in my bachelor's thesis [46]. It is a set of 34 208 molecules that have been extracted as frequently occurring motives from the PDB by software tools developed by Matúš Uhliar (mainly by program Sugar6 analyzer) in his master's thesis [4]. A short table of the most common saccharides, along with their count, present in this input set is in figure 5.10, while a tabular comparison of the most common saccharides in results set of SwCMP and LiCMP is in figure 5.11.

Input saccharides have been compared to the reference set that contained 924 saccharides in the case of LiCMP and 777 saccharides in the case of SwCMP. The reason for this discrepancy lies in the greater implemented functionality of LiCMP: While SwCMP in its version v0.5 was not able to distinguish 147 reference saccharides from some of the rest of 777 reference saccharides that have been used as its reference molecule set, LiCMP is fully capable of distinguishing all 924 saccharides that have been used as its reference molecule set.

| ligand code | count |
|---|---|
| NAG | 13 353 |
| MAN | 3 350 |
| GLC | 1 666 |
| BMA | 1 498 |
| BGC | 1 015 |
| NDG | 966 |
| GAL | 964 |
| FUC | 818 |
| BOG | 581 |
| XYP | 402 |

Figure 5.10: Ten most numerous saccharide types present in the set of input saccharides.

### 5.4.2 Identification results and deviations

Out of the input saccharide set, only one specimen timed out completely and LiCMP was not able to annotate it (more information about limitations of LiCMP is in section 5.5). Out of the remaining 34 207 molecules, 9 336 saccharides have not been annotated with absolute confidence (i.e. have not achieved 100 % as their similarity percentage with any molecules from the reference set). However, 9 074 saccharides have achieved similarity percentage between 95 and 99 %. Therefore, only 262 input saccharide molecules have been annotated with lower similarity percentage than 95 %. Substantial part of this deviating set are 76 saccharides that suffered from local comparison timeout (explained in section 5.5).

After cleaning the annotation result of entries that have had their quality compromised because of timeouts, 34 056 molecules have remained in the resulting successfully annotated set. Out of this group, 9 233 saccharides have been annotated with smaller similarity percentage than 100 %, while 24 824 saccharides have had their annotation validated with absolute confidence (i.e. have achieved 100 % as their similarity percentage with at least one molecule from the reference set). For comparison, only 1 135 saccharides have been annotated with smaller similarity percentage than 100 %. This discrep-

| ligand code | count |
|---|---|
| 5AX | 13 646 |
| ASO | 6 838 |
| ALL | 1 809 |
| FU4 | 1 002 |
| A2G | 910 |
| ARA | 624 |
| BOG | 536 |
| B2G | 516 |
| LMT | 446 |
| SUC | 262 |

| ligand code | count |
|---|---|
| 5AX | 13 660 |
| ASO | 6 827 |
| FU4 | 922 |
| BGC | 692 |
| BOG | 518 |
| NAG | 486 |
| XYP | 426 |
| GLC | 422 |
| MAL | 298 |
| NDG | 273 |

Figure 5.11: Ten most numerous saccharide types present among computed annotations of said saccharides by SwCMP (left) and LiCMP (right).

ancy is related with higher sensitivity of LiCMP and its slightly different molecule similarity metric that differs from the original metric in SwCMP. Summary of the annotation process results from LiCMP in comparison with results from SwCMP is depicted in figure 5.12). As expected, only 185 saccharides achieved lower similarity percentage than 95 %. Figure 5.13 shows the distribution of similarity scores of annotation results by LiCMP in comparison with annotation results by SwCMP.

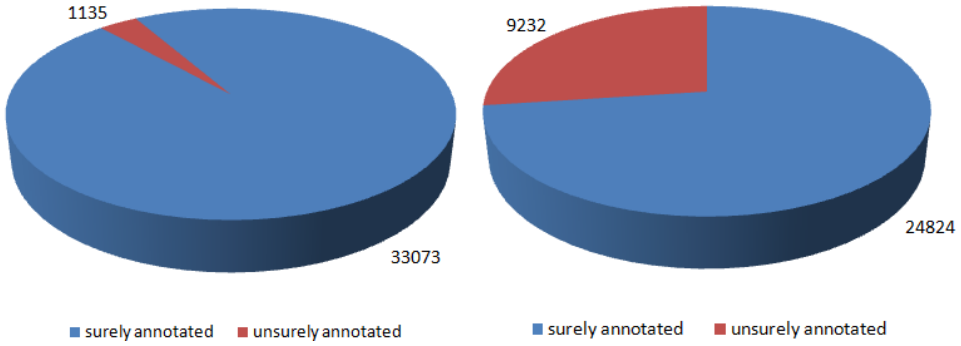



Figure 5.12: Pie charts of saccharide annotation results by SwCMP (left) and LiCMP (right)

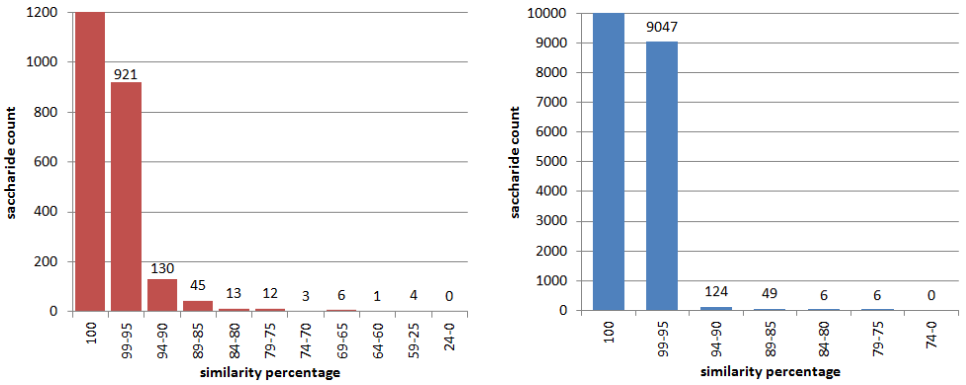



Figure 5.13: Distribution of similarity scores of saccharide annotation results by SwCMP (left) and LiCMP (right).

Distribution of input and output saccharide code count varies significantly at the first glance, there is, however, a simple reason for this difference. Output saccharides that are present in the results of LiCMP and are lacking from the results of SwCMP (e.g. BGC, NAG, XYP) are molecules that have been sorted into an isomer group by SwCMP, i.e. a group which atoms are completely identical from the viewpoint of the program. The main reason for SwCMP to use isomer groups was its lack of chirality related functionality (most saccharides differ from the rest of their group only by the configuration of atoms bound to their chiral centers). Since LiCMP supports chirality recognition and comparison, no isomer groups are needed and output ligand names truly belong to the reference molecules that participated in the computation of the highest similarity percentage result.

An important similarity between annotation validation results of SwCMP and LiCMP exists in the number of similarly annotated saccharides (i.e. saccharides that have had their annotation validated and confirmed). SwCMP claims that only 11 363 saccharides already have the right annotation, and LiCMP reports that only 10 844 molecules are correctly annotated (as shown for both programs in figure 5.14).

6 657 molecules have an error in their chirality. Vast majority of them (6 252) have only one chiral center with malformed spatial configuration of its bound atoms. In figure 5.15, an example of input sac-

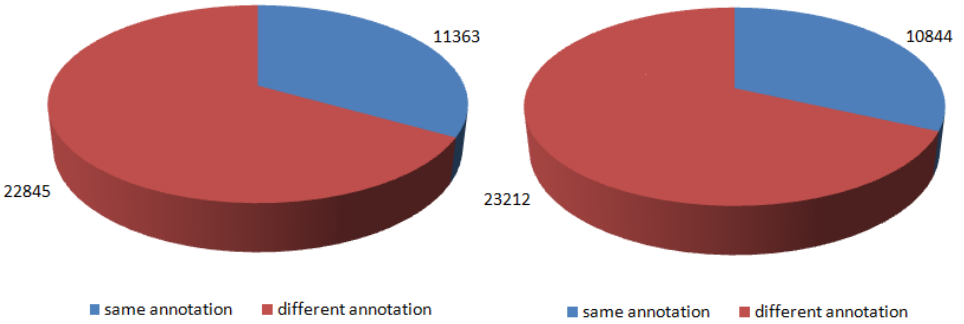



Figure 5.14: Pie charts of the number of saccharides that have had their annotations confirmed by SwCMP (left) and LiCMP (right).

charide that had 100 % similarity percentage computed by SwCMP and 99 % similarity percentage as computed by LiCMP.

## 5.5 Limitations

Due to the nature of the algorithm that has been implemented and used for isomorphism discovery, there exist certain limitations of what ligands can LiCMP process in its current version v0.7.1. The algorithm used is a brute-force algorithm that tries to enumerate and assess every possible isomorphism (but chemically plausible isomorphism, i.e. without pairing of atoms of different element type) of subgraphs of two molecular graphs. Every plausible isomorphism is discovered via synchronized DFS traversal of both molecular graphs. This step is not problematic in terms of computational complexity when the vast majority of atoms in both graphs is connected to at most two other atoms of different element type (total number of connections is not important here bercause of semantics restrictions). When the majority of atoms are connected to three (or more) other atoms of the same element type, total number of isomorphisms that need to be checked rises significantly, since each additional pairing of atom (that has for example four usable atoms and is paired to an atom from the second graph that has for example three usable atoms) creates another 24 possible pairings (enumerated by the equation 4.1). Ten pairs of atoms can, under ideal circumstances, increase the number of possible pairings $24^{10} = 6.34 \cdot 10^{13}$ times. In this way,

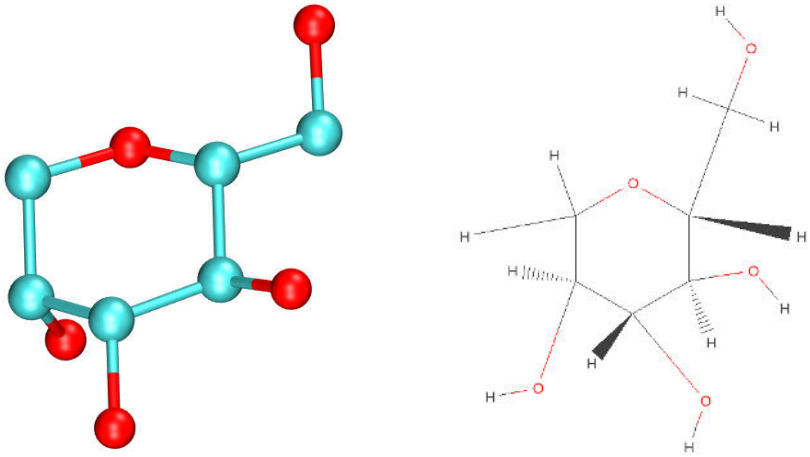


Figure 5.15: An example of a saccharide molecule (left) that has been annotated as ASO saccharide type (right, source: LigandExpo) with 100 % similarity by SwCMP and with 99 % similarity by LiCMP.

assessment of all plausible isomorphisms can cease to be computationally feasible. An example of aforesaid problematic molecule is a member of the heme ligand family (figure 5.16).

To prevent stalling the annotation process of LiCMP when such problematic molecule is encountered, two timeout mechanisms have been implemented. Local timeout is relevant for each pairwise comparison of a reference ligand and a ligand to be identified and limits its length to a set amount of time (10 minutes by default, can be modified via the -scTout command line parameter - for details about command line parameters, see section 4.2.2). When the given time runs out, ongoing comparison is stopped and the best discovered isomorphism so far is returned as a result of the aborted pairwise comparison. A note to the logfile is made as well.

Another timeout mechanism that has been implemented into LiCMP is the global timeout. The length of time that the identification process of each ligand to be identified is taking is measured as well, and if it exceeds set amount of time (30 minutes by default, can be modified via the -idTout command line parameter - for details about command line parameters, see section 4.2.2), the identification is immediately canceled. No identification from this canceled process is written to the logfile since there is no guarantee that the ligand to

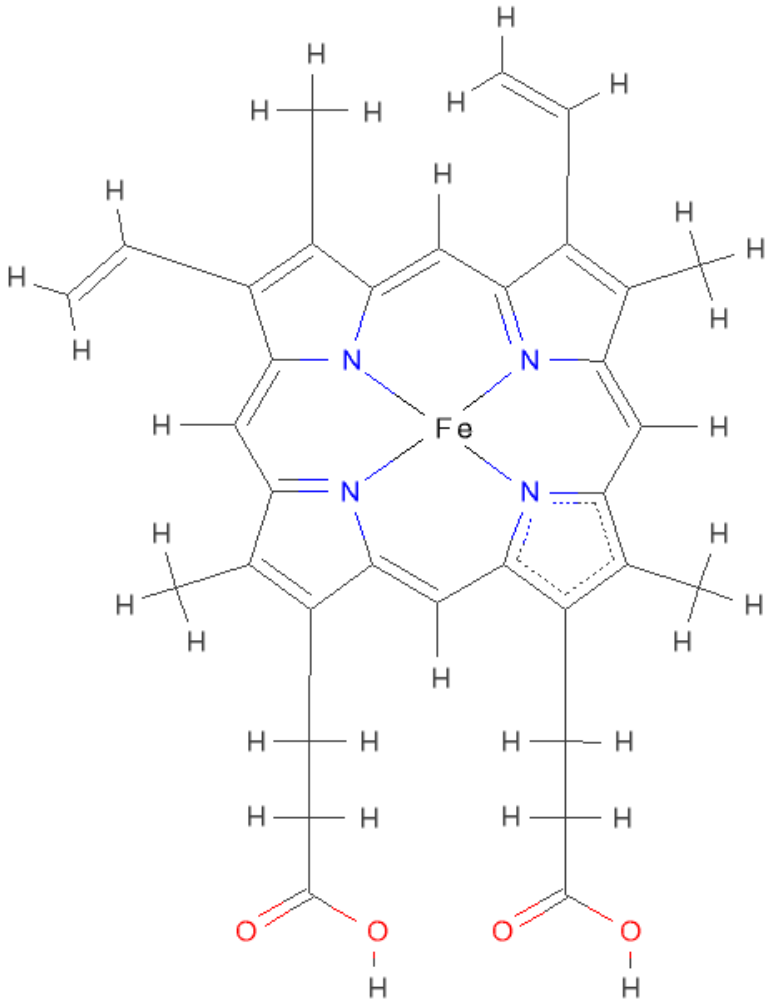



Figure 5.16: Protoporphyrin IX. Source: LigandExpo.

be identified has been pairwise compared to every reference ligand at least for a short time. Instead, only a short error message with the filename of the ligand to be identified is written to the output logfile.

## 5.6 Summary

Out of the input set of ligands to be identified, only 52.26 % of ligands have been annotated with absolute confidence and 26.6 % of ligands have been annotated with smaller similarity percentage than 95 %. This result shows that ligands, extracted from their source biomacromolecules, are of somewhat low quality and definitely need to have their annotation validated. Common problems include missing atoms (or atom groups), presence of atoms (or atom groups) that do not belong to the ligand, and wrong bound atom configuration around carbon stereogenic centers.

Even though LiCMP is more sensitive to a wider variety of errors and can, unfortunately, sometimes provide a false negative result, it does not negatively affect the ligand validation results in comparison to SwCMP, as was shown in section 5.4.2. Only 0.8 % of saccharide molecules have been annotated with smaller similarity percentage than 95 % by LiCMP as opposed to 0.62 % by SwCMP.

## 5.7 Presentation and utilization of results

LiCMP has been used for deep black-box analysis of results of MotiveValidator [48]. It is an interactive web-based validation tool that checks whether a ligand atom has correct annotation by comparing it to a model ligand of said annotation. If not, it outputs discovered differences.

## 5.8 Future plans

Functionality-wise, LiCMP is well equipped to discern ligands from one another with regards to their topology as well as their chirality configuration of all of their carbon stereogenic centers. The main current drawback of LiCMP is its low efficiency that stems from the

use of a brute-force algorithm. Extensive study of the state of the art graph algorithms that work with isomorphism discovery (both graph isomorphism and subgraph isomorphism) is required to design a more efficient algorithm. Another plausible way might be to represent each ligand not as a connected molecular graph of atoms, but as a connected graph of components. A component, in this meaning, can be an atom group, a ring, or any small motive that reoccurs in many ligands. Comparisons would be carried out on the level of components first, and then on the level of pairwise comparisons of components that have been matched together to catch smaller problems in 3D structure of a ligand to be identified.

Another way to improve identification performance would be to better split pairwise comparisons among available logical cores in a system. Concretely, the performance would improve if the program would not wait for two lengthy pairwise comparisons to finish and would start dispatching pairwise comparisons of another ligand to be identified to idling threads. This enhancement would not solve problems of the brute force approach, but it would increase the throughput of the annotation process, at least a little.

To increase the confidence of bond order detection and chirality comparison, it would help to revise their associated algorithms. Current solution works for the vast majority of ligands. When validating a ligand, however, there are many special cases that the programmer of validation and annotation software has to keep in mind and plan for in his algorithm design and implementation.

One of the main sources of functionality ideas are the users of (formerly) SwCMP and LiCMP. Without its users, a piece of software is only a little more than a completed exercise of the programmer's brain. LiCMP can easily be equipped with a GUI should the need arise since it has been developed with the aid of Qt SDK.

# 6 Conclusion

The goal of this thesis was to design and implement algorithms for automatic batch analysis, annotation, and validation of ligand molecules by comparing them to a set of reference ligands. Fulfillment of said goal required applying knowledge about variety of ligands and their structure to the process of finding suitable computer representation for ligand molecules. Such representation has been found in the form of the molecular graph that has been modified for the purpose of containing 3D structure information.

Application LiCMP has been designed and implemented as the result. It receives ligands to be identified from supplied PDB files, constructs their topology (represented by the modified molecular graph) and compares each of them with reference ligands obtained from a trusted source of quality data. The comparison of ligands is carried out in terms of construction of the best isomorphism of biggest possible subgraphs of two graphs. Quality of each discovered isomorphism is assessed using a custom metric that emphasizes atom presence deficiencies.

Implemented annotation algorithm in LiCMP has then been used to annotate a set of 10 247 selected ligands, extracted from metalloproteins, against a set of 1 865 reference ligands. Results indicate that input ligands to be identified definitely need to have their structure tended. Validity of results, returned by LiCMP, have then been cross-checked on an input set of 34 208 saccharide molecules with results from SwCMP v0.5 and it has been shown that LiCMP is more sensitive to subtle structure issues with the implementation of spatial configuration recognition of carbon stereogenic centers.

LiCMP in version v0.7.1 is a stable tool for batch annotation of ligands with nearly complete functionality set for this task. As such, it has been used to validate results of MotiveValidator [48], a web-based ligand validation tool which results have been published in the Nucleic Acids Research journal [48]. There are still reserves to be tackled in terms of annotation process performance in both isomorphism finding algorithm and function that splits pairwise comparison to worker threads. With said improvements and after more thorough verification of results that the program returns, it will be ready

to become useful tool for various projects that are being researched at National Centre for Biomolecular Research (NCBR).

# 7 Appendices

## 7.1 Contents of attached CD

- this thesis in PDF format
- LaTeXsource code of this thesis
- graphical figures used in this thesis
- source code of LiCMP v0.7.1
- documentation of LiCMP v0.7.1 in PDF and HTML format
- full class diagram of LiCMP v0.7.1
- compiled binary of LiCMP v0.7.1 for the Windows x86 platform
- source files of ligands that comprise the set of ligands to be annotated
- source files of reference ligands for the set of ligands to be compared against
- annotation and validation result for the set of ligands to be annotated
- source files of saccharides that comprise the cross-validation set
- source files of reference saccharides for the cross-validation set of saccharides to be compared against
- annotation and validation result for the cross-validation set of saccharides

## 7.2 Ligands from Ligand Expo mentioned in this thesis

| ligand code | name | chemical formula |
|---|---|---|
| 3GR | glyceraldehyde | $C_3H_6O_3$ |
| 5AX | 2-(acetylamino)-1,5-anhydro--2-deoxy-D-glucitol | $C_8H_{15}NO_5$ |
| A2G | N-acetyl-2-deoxy-2-amino-galactose | $C_8H_{15}NO_6$ |
| ADP | adenosine-5′-diphosphate | $C_{10}H_{15}N_5O_{10}P_2$ |
| ALL | D-allopyranose | $C_6H_{12}O_6$ |
| ANP | phosphoaminophosphonic-acid--adenylate ester | $C_{10}H_{17}N_6O_{12}P_3$ |
| ARA | $\alpha$-L-arabinose | $C_5H_{10}O_5$ |
| ASO | 1,5-anhydrosorbitol | $C_6H_{12}O_5$ |
| ASP | aspartic acid | $C_4H_7NO_4$ |
| ATP | adenosine-5′-triphosphate | $C_{10}H_{16}N_5O_{13}P_3$ |
| B2G | galactobiose | $C_{12}H_{22}O_{11}$ |
| B7G | heptyl-$\beta$-D-glucopyranoside | $C_{13}H_{26}O_6$ |
| BGC | $\beta$-D-glucose | $C_6H_{12}O_6$ |
| BMA | $\beta$-D-mannose | $C_6H_{12}O_6$ |
| BOG | $\beta$-octylglucoside | $C_{14}H_{28}O_6$ |
| BUA | butyric acid | $C_4H_8O_2$ |
| DMU | decyl-$\beta$-D-maltopyranoside | $C_{22}H_{42}O_{11}$ |
| END | 1,6:5,9:8,12:11,16-tetraanhydro-2,3,4,10,13,14-hexadeoxy-D-glycero--D-allo-D-gulo-heptadeca-2,13--dientiol | $C_{17}H_{24}O_7$ |
| FU4 | 2,6-anhydro-1-deoxy-D-galactitol | $C_6H_{12}O_4$ |
| FUC | $\alpha$-L-fucose | $C_6H_{12}O_5$ |
| FMN | flavin mononucleotide | $C_{17}H_{21}N_4O_9P$ |
| FMT | formic acid | $CH_2O_2$ |
| GAL | $\beta$-D-galactose | $C_6H_{12}O_6$ |
| GLC | $\alpha$-D-glucose | $C_6H_{12}O_6$ |
| GTP | guanosine-5′-triphosphate | $C_{10}H_{16}N_5O_{14}P_3$ |
| HEM | protoporphyrin IX | $C_{34}H_{32}FeN_4O_4$ |
| HOH | water | $H_2O$ |

| ligand code | name | chemical formula |
|---|---|---|
| LMT | dodecyl-$\beta$-D-maltoside | $C_{24}H_{46}O_{11}$ |
| MAL | maltose | $C_{12}H_{22}O_{11}$ |
| MAN | $\alpha$-D-mannose | $C_6H_{12}O_6$ |
| MG | magnesium ion | Mg |
| MUG | 4-methylumbelliferyl-$\alpha$-D-glucose | $C_{16}H_{18}O_8$ |
| NAG | N-acetyl-D-glucosamine | $C_8H_{15}NO_6$ |
| NDG | 1-(acetylamino)-2-deoxy-$\alpha$- -D-glucopyranopse | $C_8H_{15}NO_6$ |
| O | oxygen atom | O |
| SUC | sucrose | $C_{12}H_{22}O_{11}$ |
| XYP | $\beta$-D-xylopyranose | $C_5H_{10}O_5$ |

# 8 Literature


[1] GARRETT, R. and C. GRISHAM. *Biochemistry*. Belmont, CA: Brooks/Cole, Cengage Learning, 2013. ISBN 11-331-0629-3.

[2] LEACH, A. and V. GILLET. *An introduction to chemoinformatics*. Dordrecht: Springer, 2007. ISBN 978-140-2062-902.

[3] RCSB PDB. WWPDB. *RCSB PDB* [online]. 2014-04-30 [cit. 2014-05-03]. Available at: `http://www.rcsb.org/pdb/home/home.do`

[4] UHLIAR, Matúš. *Vyhl'adávanie a predikcia biochemicky významných motívov v molekulách poteínov*. Brno, 2010. Available from: `https://is.muni.cz/th/139735/fi_m/thesis.pdf`. Master thesis. Masaryk University, Faculty of Informatics. Thesis advisor RNDr. Radka Svobodová Vařeková, Ph.D.

[5] WORLDWIDE PROTEIN DATA BANK. *Ligand Expo* [online]. 2014-04-30 [cit. 2014-05-03]. Available from: `http://ligand-expo.rcsb.org/index.html`

[6] COX, T. *Inorganic chemistry*. London: Taylor, 2003. ISBN 02-034-8827-X.

[7] TEIF, V. Ligand-Induced DNA Condensation: Choosing the Model. *Biophysical Journal*. 2005, vol. 89, issue 4, p. 2574-2587.

[8] TEIF, V. and K. RIPPE. Statistical–mechanical lattice models for protein–DNA binding in chromatin. *Journal of Physics: Condensed Matter*. 2010, vol. 22, issue 41, p. 414105.

[9] CAMPBELL, N., B. WILLIAMSON and R. HEYDEN. *Biology: Exploring life*. Boston, Mass: Pearson/Prentice Hall, 2006. ISBN 978-013-2508-827.

[10] CORDERO, B., V. GÓMEZ, A. PLATERO-PRATS, M. REVÉS, J. ECHEVERRÍA, E. CREMADES, F. BARRAGÁN and S. ALVAREZ. Covalent radii revisited. *Dalton Transactions*. 2008, issue 21, p. 2832-.

[11] PYYKKÖ, P. and M. ATSUMI. Molecular Single-Bond Covalent Radii for Elements 1-118. *Chemistry - A European Journal*. 2009, vol. 15, issue 1, p. 186-197.

[12] PYYKKÖ, P. and M. ATSUMI. Molecular Double-Bond Covalent Radii for Elements Li-E112. *Chemistry - A European Journal*. 2009-11-23, vol. 15, issue 46, p. 12770-12779.

[13] PYYKKÖ, P., S. RIEDEL and M. PATZSCHKE. Triple-Bond Covalent Radii. *Chemistry - A European Journal*. 2005-06-06, vol. 11, issue 12, p. 3511-3520.

[14] WAGNIERE, G. *On Chirality and the Universal Asymmetry: Reflections on Image and Mirror Image*. Zurich: VHCA [with] Wiley-VCH, 2007. ISBN 39-063-9038-1.

[15] SOLOMONS, G. and C. FRYHLE. *Organic chemistry*. 10th ed. Hoboken, NJ: Wiley, 2011. ISBN 04-704-0141-9.

[16] FOX, M. a J. WHITESELL. *Organic chemistry*. 3rd ed. Sudbury, Mass: Jones and Bartlett, 2004. ISBN 978-076-3721-978.

[17] ELIEL, E., S. WILEN and L. MANDER. *Stereochemistry of organic compounds*. New York: Wiley, 1994. ISBN 04-710-1670-5.

[18] STREITWIESER, A., C. HEATHCOCK and E. KOSOWER. *Introduction to organic chemistry*. 4th ed. New York: Maxwell Macmillan International, 1992. ISBN 00-241-8170-6.

[19] MARCH, J. *Advanced organic chemistry: Reactions, Mechanisms, and Structure*. 4th ed. New York: John Wiley and Sons, 1992. ISBN 978-8126510467.

[20] MOSS, G. Basic terminology of stereochemistry (IUPAC Recommendations 1996). *Pure and Applied Chemistry*. 1996, vol. 68, issue 12.

[21] COOK, D. *Illustrated Dictionary of Chemistry*. New Delhi: Lotus Press, 2004. ISBN 978-818-9093-211.

[22] Nomenclature and symbolism for amino acids and peptides. *Pure and Applied Chemistry*. 1984, vol. 56, issue 5, p. 595-624.

[23] KHOURY, G., R. BALIBAN and C. FLOUDAS. Proteome-wide post-translational modification statistics: frequency analysis and curation of the swiss-prot database. *Scientific Reports*. 2011-9-13, vol. 1, article nr. 90.

[24] GASTEIGER, J. *Handbook of chemoinformatics: from data to knowledge*. Weinheim: Wiley-VCH, 2003. ISBN 35-273-0680-3.

[25] KVASNIČKA, V., M. KRATOCHVÍL, and J. KOČA. *Matematická chemie a počítačové řešení syntéz*. 1. edition. Praha: Academia, 1987, p. 12-13. Pokroky chemie, 16.

[26] MATOUŠEK, J. and J. NEŠETŘIL. *Invitation to discrete mathematics*. 2nd ed. New York: Oxford University Press, 2009. ISBN 01-985-7042-2.

[27] DEVLIN, K. *Sets, functions, and logic: an Introduction to Abstract Mathematics*. 3rd ed. Boca Raton, Fla.: Chapman, 2004. ISBN 15-848-8449-5.

[28] FOGGIA, P., C. SANSONE and M. VENTO. A Performance Comparison of Five Algorithms for Graph Isomorphism. In: *Proc. 3rd IAPR-TC15 Workshop: Graph-Based Representations in Pattern Recognition* [online]. 2001 [cit. 2014-05-19]. Available from: `http://www.engr.uconn.edu/~vkk06001/GraphIsomorphism/Papers/VF_SD_NAUTY_Ullman_Experiments.pdf`

[29] COOK, S. The Complexity of Theorem-Proving Procedures. In: *STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*. New York: Association for Computing Machinery, 1971, p. 151-158.

[30] ULLMANN, J. An Algorithm for Subgraph Isomorphism. *Journal of the ACM*. 1976, vol. 23, issue 1, p. 31-42.

[31] BUNIN, B., J. BAJORATH, B. SIESEL and G. MORALES. *Chemoinformatics: Theory, Practice and Products*. Dordrecht: Springer, 2007. ISBN 14-020-5001-1.

[32] WEININGER, D. SMILES, a Chemical Language and Information System: 1. Introduction to Methodology and Encoding Rules. *Journal of Chemical Information and Modeling*. 1988-02-01, vol. 28, issue 1, p. 31-36.

[33] HELSON, H. Structure Diagram Generation. *Reviews in Computational Chemistry*. 2007, Volume 13.

[34] GORDON, M. and J. POPLE. Approximate Self-Consistent Molecular-Orbital Theory. VI. INDO Calculated Equilibrium Geometries. *The Journal of Chemical Physics*. 1968, vol. 49, issue 10, p. 4643-.

[35] PARSONS, J., B. HOLMES, M. ROJAS, J. TSAI and C. STRAUSS. Practical Conversion from Torsion Space to Cartesian Space for In Silico Protein Synthesis. *Journal of Computational Chemistry*. 2005-07-30, vol. 26, issue 10, p. 1063-1068.

[36] GENTLE, J. *Matrix algebra: Theory, Computations, and Applications in Statistics*. London: Springer, 2007, p. 299. ISBN 0387708723.

[37] WwPDB Frequently Asked Questions. WORLDWIDE PROTEIN DATA BANK. *Welcome to the Worldwide Protein Data Bank* [online]. 2014-04-29 [cit. 2014-05-03]. Available from: `http://www.wwpdb.org/faq.html`

[38] HUBBARD, T., A. MURZIN, S. BRENNER and C. CLOTHIA. SCOP: a Structural Classification of Proteins database. *Nucleic Acids Research*. 1997, vol. 25, No. 1, p. 236-239.

[39] Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description. In: *Worldwide Protein Data Bank* [online]. Version 3.30. Worldwide Protein Data Bank, 2012-11-21 [cit. 2014-05-03]. Available at: `ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_A4.pdf`

[40] Chemical Component Dictionary. WORLDWIDE PROTEIN DATA BANK. *Welcome to the Worldwide Protein Data Bank* [online]. 2014-04-29 [cit. 2014-05-03]. Available from: `http://www.wwpdb.org/ccd.htmll`

[41] HUMPHREY, W., A. DALKE and K. SCHULTEN. VMD: Visual molecular dynamics. *J. Molec. Graphics*. 1996, vol. 14, No. 1, p. 33-38.

[42] STROUSTROUP, B. C++ Applications. STROUSTROUP, B. *Bjarne Stroustroup's Homepage* [online]. 2014-02-17 [cit. 2014-05-04]. Available from: `http://www.stroustrup.com/applications.html`

[43] STROUSTRUP, B. *C++ programming language* Third. ed. Massachusetts: Addison-Wesley, 1997, 910 p. ISBN 02-018-8954-4.

[44] Qt 5 | Documentation | Qt Project. Digia Plc. *Qt Project* [online]. 2014-02-05 [cit. 2014-05-03]. Available at: `http://qt-project.org/doc/qt-5/index.html`

[45] GAMMA, E., R. HELM, R. JOHNSON and J. VISSIDES. *Design patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley, 1995. ISBN 02-016-3361-2.

[46] HORSKÝ, V. *Annotation of saccharide molecules*. Brno, 2012. Available from: `https://is.muni.cz/auth/th/358970/fi_b/Thesis.pdf`. Bachelor thesis. Faculty of Informatics, Masaryk University. Advisor RNDr. Radka Svobodová Vařeková, Ph.D.

[47] Container Classes | QtCore 5.2 | Documentation | Qt Project: Algorithmic Complexity. DIGIA PLC. *Qt Project* [online]. 2013 [cit. 2014-05-07]. Available from: `http://qt-project.org/doc/qt-5/containers.html#algorithmic-complexity`

[48] SVOBODOVÁ VAŘEKOVÁ, R., D. JAISWAL, D. SEHNAL, C.-M. IONESCU, S. GEIDL, L. PRAVDA, V. HORSKY, M. WIMMEROVA and J. KOČA. MotiveValidator: interactive web-based validation of ligand and residue structure in biomolecular complexes. *Nucleic Acids Research*. 2014.