universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Graphical Visualization of 2D-Atom-Based Descriptors in PLS-QSAR-Equations Using MOE SVL

angestrebter akademischer Grad

Magister der Pharmazie (Mag. pharm.)

Verfasser:	Christoph Waglechner
Matrikelnummer:	0204938
Studienrichtung:	Pharmazie
Betreuer:	Univ.-Prof. Mag. Dr. Gerhard F. Ecker

Wien, im November 2009

Mein besonderer Dank gilt Herrn Univ.-Prof. Mag. Dr. Gerhard Ecker, der nicht nur die Arbeit ermöglicht und unterstützt hat, sondern auch immer mit Geduld, Übersicht und Empathie auf Fragen und Probleme eingegangen ist und mit seinen Vorschlägen die Entstehung begleitet hat.

Selbstverständlich danke ich Herrn o. Univ.-Prof. Dipl.-Ing. Mag. Dr. Christian Noe für die Bereitstellung meines Arbeitsplatzes und der Infrastruktur im Department für Medizinische/ Pharmazeutische Chemie.

Entscheidend zum Gelingen jedes Projektes trägt ein gutes Arbeitsklima bei. Daher möchte ich allen Mitgliedern unserer Arbeitsgruppe meinen besonderen Dank aussprechen. Danke Andreas, Barbara, Daniela, Freya, Martin, René, Rita und Wolfgang. Thank you, Andrea, Ishrat, Minh and Yogesh.

Im Besonderen gilt das für zwei Arbeitskollegen, Herrn Mag. pharm. Lars Richter und Herrn Mag. pharm. Michael Demel. Lars, danke, dass Du mich auf die Idee gebracht hast, in unserer Gruppe die Diplomarbeit zu verfassen. Danke Michi, dass Du mit vielen kritischen Fragen immer wieder auf Schwächen und Probleme in meinen Ausführungen hingewiesen hast.

Der Rückhalt in der Familie ist ausnehmend wichtig und entscheidend, nicht nur in finanzieller Hinsicht. Deshalb möchte ich mich bei meinen Eltern, Mag. pharm. Dr. Richard und Mag. pharm. Brigitte Waglechner, und meinen Geschwistern, Günther und Elisabeth, bedanken.

Ganz besonders danke ich Lisa für die Geduld und Ruhe, die sie im Laufe der letzten Jahre aufgebracht hat.

Disclaimer

SVL scripts or parts of SVL scripts used to assemble the QSAR Backprojection SVL script have been copied and/or modified from scripts owned by Chemical Computing Group Inc., Montreal, Canada and are labelled within the code. Chemical Computing Group Inc. grants permission to use, copy, modify and distribute used software provided that: (1) unmodified or functionally equivalent code derived from that software must contain this notice; (2) all code derived from this software must acknowledge the author(s) and institution(s); (3) the names of the author(s) and institution(s) must not be used in advertising or publicity pertaining to the distribution of the software without specific, written prior permission; (4) all code derived from that software be executed with the Molecular Operating Environment (MOE) licensed from Chemical Computing Group Inc.

MOE, SVL, and the Chemical Computing Group Inc. logo are trademarks of Chemical Computing Group Inc., Montreal, Canada or its licensors.

Tripos, SYBYL, Tripos Bookshelf, SARNavigator, Distill and HQSAR are trademarks or registered trademarks of Tripos Inc., 1699 South Hanley Road, St. Louis, MO, USA.

Molecule illustrations of section 4.1 were exported from MOE using the POV-Ray™ export option and rendered in POV-Ray™ 3.6.1, available online under the POV-Ray™ license via http://www.povray.org.

Graphics and flowcharts were created using LaTeX and the TikZ and PGF packages, which are available online via http://sourceforge.net/projects/pgf.

Contents

1. Introduction and Aim of the Work

Since Corwin Hansch started with the first linear regression equations to model activity and properties in the early 1960s (e.g. [1]), an enormous amount of models based on various methods has been developed to predict quantitative structure-activity relationships (QSAR) or quantitative structure-property relationships (QSPR). As both are pattern recognition models which identify structural trends of small molecules and link them with activity, they can be used for new molecule's properties, too. But these models can also be used to *understand why* certain molecules are active and others are not [2]. According to Guha, this is of big importance when the structure-activity relationship (SAR) is poorly known [2].

On the one hand, QSAR models can be intended as screening tools in large libraries or used to predict well-understood properties like boiling points or vapor pressure. In this field, the focus might be on the prediction and its quality only, with a possible interpretation being secondary. On the other hand, QSAR can be explanatory, QSAR can give hints on the SAR – especially if a receptor structure doesn't exist or the mechanism of action is obscure [2], and QSAR can imply modifications on the molecules to improve activity, known as inverse QSAR [3] or RQSAR [4], for which special algorithmic models or descriptors have been designed [2].

Nonetheless, even for standard models, an interpretation and a correlation of features and activity is possible. In principle, Guha considers model-based and model-free, thus algorithmic, methods. The latter, among them k-nearest neighbours or random forests, don't support the user in getting explanatory information [2]. Although there might be some interpretative information even here, we resultingly focus on the first type, to which e.g. linear regression or partial least squares (PLS) methods belong, but also neural networks and support vector machines. Nonetheless, within the model-based methods, there also exist differences in interpretability. However, usually a trade-off between interpretability and accuracy is denoted (see figure 1.1, [2]).

Therefore, the model determines the possibilities of interpretation. This work solely focus on methods classified as good interpretable, with the trade-off of a possibly weaker accuracy. There are multiple methods available which allow for a deeper insight into SAR. Chapter 2 gives a quick overview, taking into account approaches leading more or less to a final linear

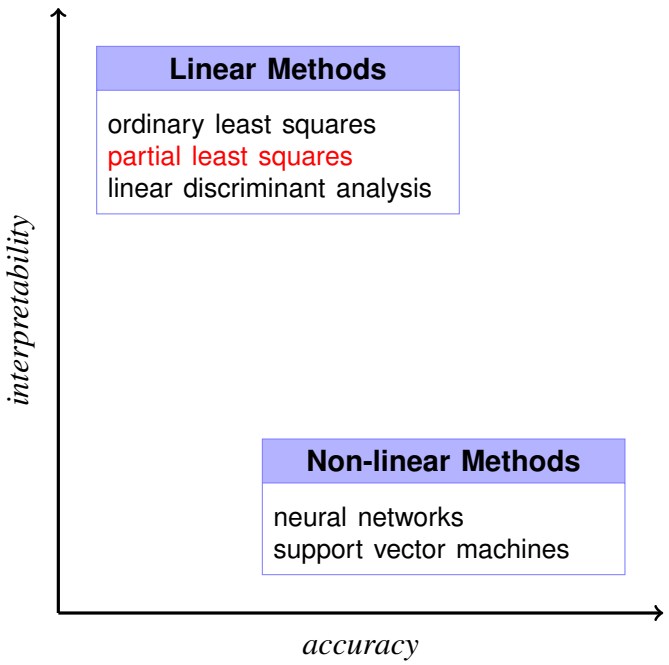


Figure 1.1.: Trade-off between accuracy and interpretability for various model-based methods (taken from [2]). The partial least squares method is classified as *better interpretable* [2].

equation representing the model. Nonetheless, as far as pure PLS methods are concerned, only the Partial Least Square-Components-based method (section 2.5) is capable of further exploring information included in the equations.

We concentrate on the PLS method, although a normal linear regression can be used as input as well, and in chapter 3, we create a new script which aims at retrieving and visualizing QSAR information out of PLS-2D-QSAR equations, with an exclusive focus on so-called atom-based descriptors as contributors to the equation, thus descriptors which are formed by combining values of individual atoms. The atomwise summing up of these values results in one single value per atom specific for the given equation. It represents the contribution of the atom to the equation. This approach was first introduced by Gombar in 2007 within his "visdom" (visual in silico design of molecules) tool and called RQSPR [4].

The script should be easy to apply, thus a graphical user interface (GUI) supporting the user is an absolute necessity, as well as the integration into a user-friendly chemical application, where MOE (Molecular Operating Environment) [5] was chosen.

Chapter 4 then proves reliability and performance of the script as well as a proof-of-concept study dealing with the basic applications of the script. This is examplified by using an in-house propafenones dataset with P-glycoprotein (P-gp) inhibitory activity, where multiple SAR studies have already been conducted.

2. Introduction to Various Methods Interpreting QSAR

Selected methods which are intended for interpreting QSAR are presented. We especially focus on those dealing with (a) linear QSAR equation(s) like fragment-based methods or partial least square components interpreting methods, but also include fingerprint- and rules-based methods as well as methods relying on physical substructures, and the fragment-descriptor method our script is based on.

2.1. Fragment-Based Methods

Considered as a rather modern QSAR method and obviously lacking a clear definition, at least two different approaches are currently called "fragment-based" methods, Sybyl's HQSAR in [6] and the FB-QSAR (fragment-based QSAR) approach, which was published in 2008 [7]. Both methods have in common that they are based on molecular fragments rather than on the whole compound and can explicitly be distinguished from fragment descriptors, where a molecule property is estimated by summing up submolecular fragment contributions subsequently used for normal QSAR approaches (compare section 3.2.1 and [8]). Within HQSAR, fragment generation is done automatically and based on some user parameters. It covers all available fragments of the database structures. In FB-QSAR, the user has to highlight the fragments, in most cases dependent on substitution patterns (compare [7]).

FB-QSAR splits a traditional 2D-QSAR as introduced by Hansch et al. [1] into submolecular parts and calculates traditional descriptors for them [7], in HQSAR the fragment count itself is taken as a fingerprint descriptor input [9]. In both cases, the resulting QSAR equation contains activity information not only for the molecule as a whole, but also for submolecular parts, thus giving a more detailed picture of positive and negative contributions as well as a better overview for the medicinal chemist.

2.1.1. FB-QSAR (Fragment-Based QSAR)

The FB-QSAR method introduced by Du et al. [7] relies on deviding the framework of a compound family into subfragments. Therefore, it can be seen as a combination of 2D-QSAR and the Free-Wilson approach. Using this process, the authors hope to even reveal small effects resulting from subtle changes in the substitution pattern of molecules, which they claim to be ignored when applying traditional 2D-QSAR methods. In principle, the QSAR equation can then be seen as the sum of these fragments [7] (compare figure 2.1).

$$(-\log EC_{50})_k = \sum_{i=1}^{N} b_i * f_{i,k}$$

$(-\log EC_{50})_k$ stands for the predicted activity of the k^{th} compound dependent on N fragments, b_i is a weighting factor allowing to encode a fragment's role in bioactivity and different microstructures within the molecule, and $f_{i,k}$ is the i^{th} fragment's contribution to activity of a specific compound. Then $f_{i,k}$ can be determined with a series of classical physical or chemical descriptors $p_{i,k,l}$, with the l^{th} descriptor calculated for fragment i of molecule k, again including a coefficient a_l as in traditional 2D-QSAR [7].

$$f_{i,k} = \sum_{l=1}^{L} a_l * p_{i,k,l}$$

Both subsets can be combined to the foundational equation of FB-QSAR with two sets of coefficients, a_l for the physicochemical properties and b_i for the fragments in a molecular family [7].

$$(-\log EC_{50})_k = \sum_{i=1}^{N} b_i * \left(\sum_{l=1}^{L} a_l * p_{i,k,l} \right)$$

Du et al. developed the iterative double least square technique (IDLS) capable of determining both coefficient sets iteratively by setting initial values for one coefficient group and thus reducing the three-dimensional simultaneous linear equation to a two-dimensional one. IDLS implements a feedback procedure to alternately improve the coefficient sets until a convergence criterion has been reached (for details refer to [7]).

The fragments have to be chosen individually, Du et al. suggest a division reflecting the substitution points of the data set's common scaffold. In contrast to Free-Wilson approach, the risk of

an "over correlation" due to a high number of variables is lower, e.g. 14 (10+4) variables in FB-QSAR versus 40 (4∗10) in Free-Wilson for 4 fragments and 10 properties in a neuraminidase inhibitor test case [7].

As contribution values are not only calculated for the whole molecule but for subfragments, the method inherently unveils activities for the previously defined fragments providing better structural information. Although primarily developed for fragment based drug design [7], within the limits of descriptor calculation and over prediction the size of fragments can be chosen individually giving a very detailed overview over the partial contribution of structures within the molecule. Additionally, in the neuraminidase example presented in [7], FB-QSAR returns slightly better results than traditional methods, especially when applied onto an external test set, resulting in a good contribution prediction of the fragments.

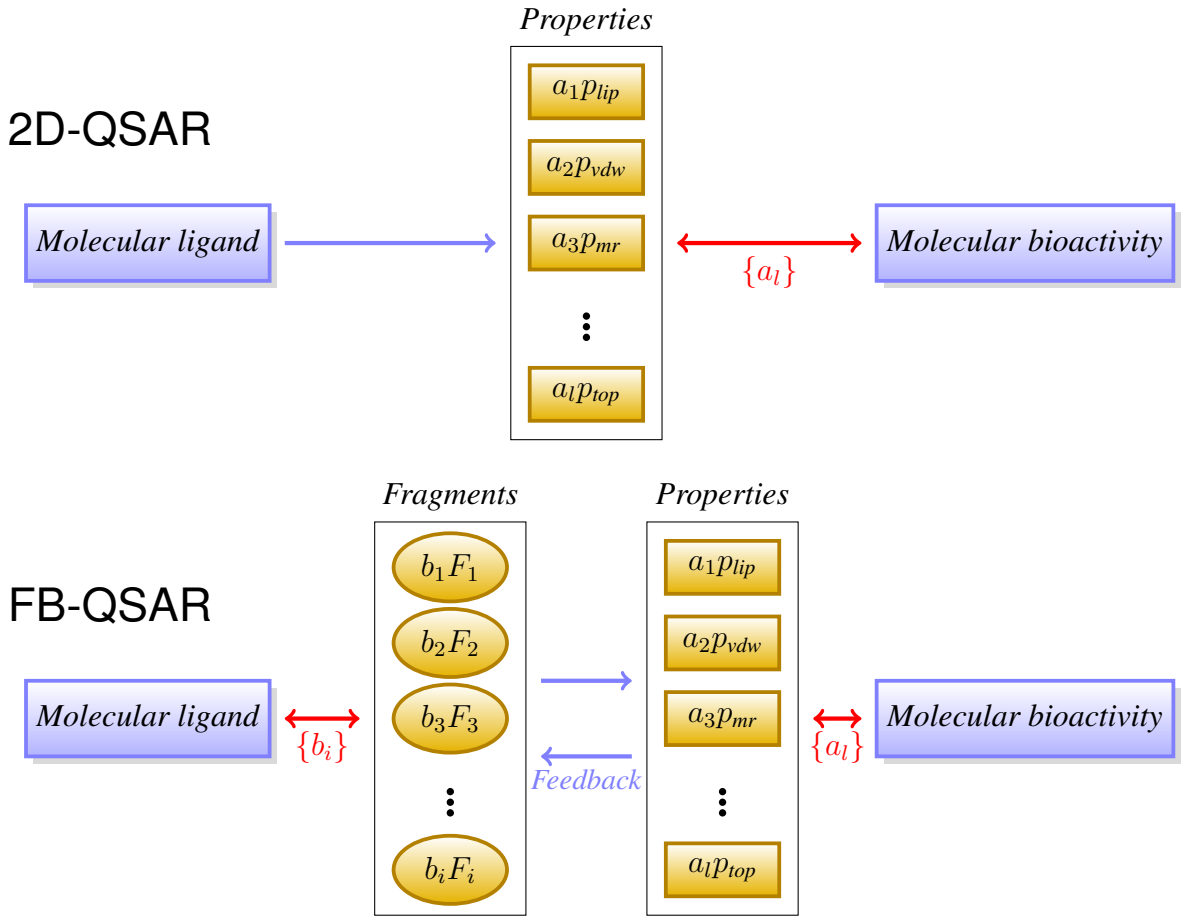


Figure 2.1.: Comparison of a classical 2D-QSAR to the FB-QSAR based on fragments. In classical 2D-QSAR, a molecule's properties are directly correlated with activity via coefficient set $\{a_l\}$, in FB-QSAR, the activity is correlated with fragments F_n through set $\{a_l\}$ and the fragment weight set $\{b_i\}$. Figure taken from [7].

2.1.2. HQSAR (Hologram QSAR)

The HQSAR method, available within the software package SYBYL by Tripos [10], is able to relate biological activity to patterns of substructural fragments, thus structural molecular compositions [9]. As many QSAR techniques, it consists of a descriptor calculation and a model generation step, but HQSAR only takes 2D structures and the associated activity as inputs [9] (see figure 2.2).

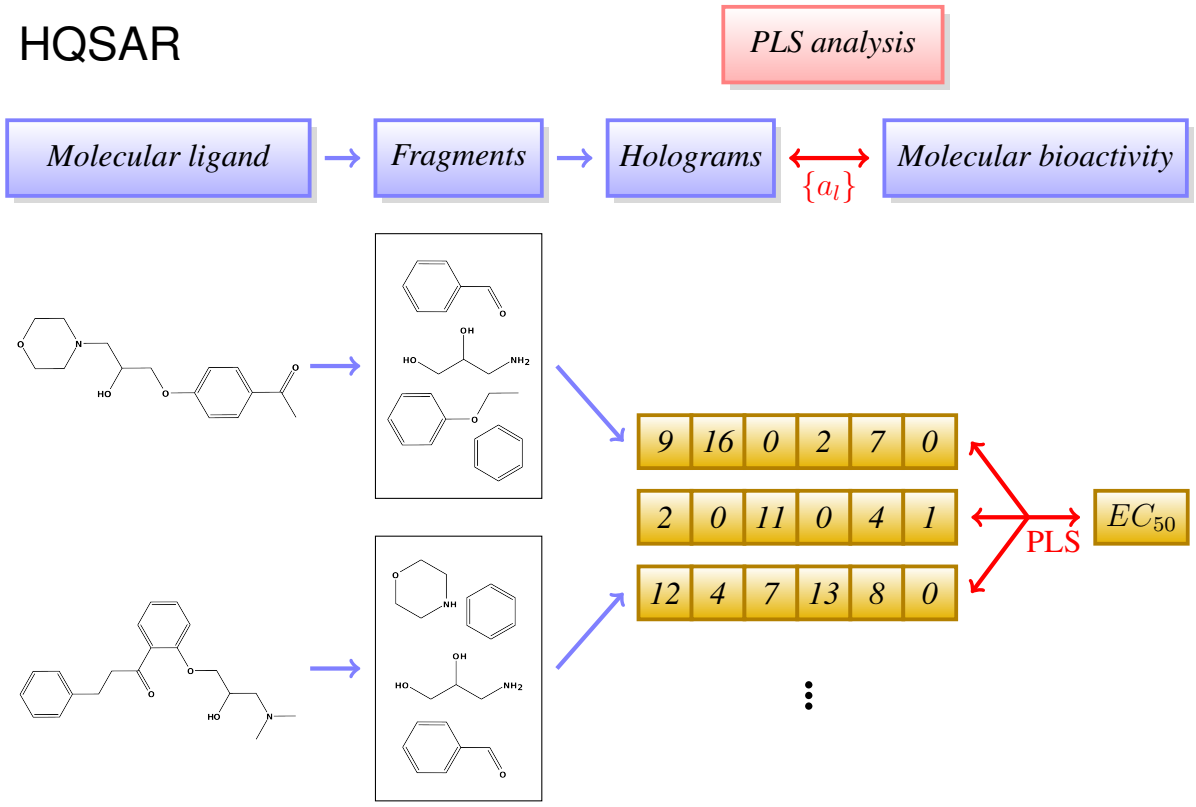
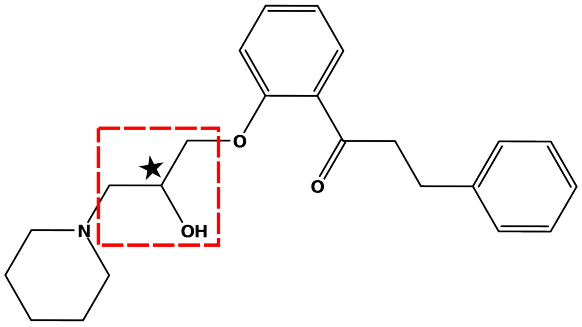
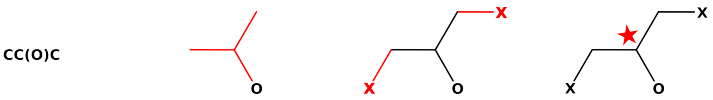


Figure 2.2.: Schematic overview on HQSAR. The fragments are derived from the compounds according to the criteria defined. After putting the fragments into holograms, a normal PLS-QSAR run is performed [6].

To code various input compounds, at first all possible fragments of length x to y are generated. By default, x is set to 4 and y to 7 resulting in fragments consisting of 4–7 atoms. Then HQSAR generates a so-called molecular hologram out of the fragment data, which is a molecular fingerprint generated on hashing using a cyclic redundancy check (CRC) function, which assigns each fragment a large integer and subsequently derives a bin number [11]. Resultingly, the fingerprint includes only fragments available in the database, in comparison to the standard structural key approach no *a priori* definition of features is necessary. Additionally, the molecular hologram is not a binary fingerprint but codes the number of fragment occurrences as the fingerprint bin value. Typically, the hologram length (by default 53, 59, 61, 71, 83, 97, 151 or



(a) GPV0002



(b) atoms

(c) atoms
bonds

(d) atoms
bonds
connections

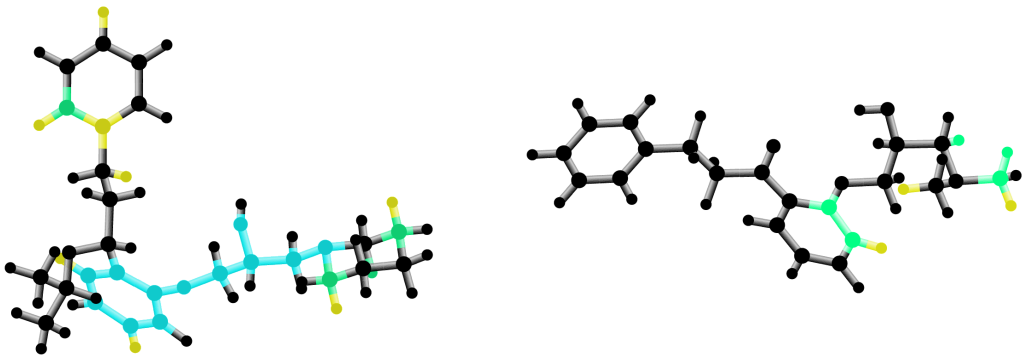
(e) atoms
bonds
connections
chirality

Figure 2.3.: Different flags set in the HQSAR option tab results in different fragments. All fragments are generated from the red-boxed moiety of the GPV0002 compound. In figure (e), the *X* denotes any heavy atom. Modified from [12]

199) is drastically lower than the number of different fragments generated from the dataset [9], so there are chances that two important features are hashed into the same bin. As the suggested bin sizes are prime numbers, typically the bins of different hologram lengths consist of different fragments to avoid that problem [11]. Resultingly, larger hologram size does not correlate with higher R^2 (compare [9]).

Apart from the size options, HQSAR offers various flags to control types of fragments for the fragment generation step (compare [11] and figure 2.3).

- Atom numbers. If selected, different elements are distinguished.
- Bond types. Decides whether HQSAR ignores the type of a bonding between two atoms.
- Atomic connections. If set to “on”, information about the hybridization state of atoms is retained.
- Hydrogen inclusion. Sets the behaviour in terms of inclusion of hydrogen atoms.
- Chirality. If active, chirality is taken into account.
- Donor/acceptor atoms. Substitutes donor/acceptor atoms with generic donor/acceptor types and calculates additional fragments .



(a) GPV0264. MCS highlighted.

(b) GPV0048. MCS hidden.

Figure 2.4.: HQSAR results exported from SYBYL. Subfigure (a) shows the MCS in turqoise, it is not displayed in subfigure (b). Yellow and green colours indicate positive contributions, neutral atoms are shown in black. In this case, no negative contribution is detected. For a detailed overview on SYBYL's HQSAR colour scheme, refer to 3.4.1. Figures taken from an examplary HQSAR model considering atoms, bonds and connections with 71 bins, 6 components, $R^2 = 0.87$, $XR^2 = 0.75$, which was performed on the propafenones dataset (see appendix B).

The QSAR model is then created using a standard PLS algorithm. It also includes an MLCS algorithm which highlights the MCS of the dataset if it is bigger than seven atoms in a connected structure (figure 2.4). It is said to be very efficient. Additionally, SYBYL provides a virtual screening feature, where a database can be scanned on basis of any HQSAR equation and a certain Tanimoto similarity cutoff value [11].

Apart from the MLCS visualization, the fragments unveil relationships between substructure and activity. As the descriptors are molecular holograms consisting of fragments, each bin is assigned a certain coefficient, a contribution to the activity. HQSAR additionally allows the generation of contribution maps which overlay the influences of all fragments at an atom level, which visualizes the individual amount each atom or substructure contributes to the equation. This information can be vital to identify key structural components. Additionally, the most important fragments in terms of contribution can be determined [6].

2.2. Fingerprint-Based Methods

A fingerprint's goal is to represent the structural features of a molecule within a bitstring, where each bit codes for a certain structural specifity. Fingerprints or parts of fingerprints can be used as simple descriptors in PLS or linear regression QSAR equations, which also allows for an

interpretation, as there is no abstraction [2]. Nonetheless, more sophisticated methods exist which can extract additional information out of fingerprints, e.g. MOE's MIBALS.

2.2.1. MIBALS

The Chemical Computing Group SVL-Exchange section [13, MIBALS] provides a MOE-tool by Chris Williams dealing with reverse fingerprinting, which are functions returning not only a fingerprint with bit positions set but also the atom keys the bit positions refer to [14].

Within a fingerprint, each bit position k_x as the k^{th} position out of K holds mutual information I on the activity state A of a molecule Q. Mutual information returns the mutual dependence between two variables [14]. This information is defined as

$$I_{A,k} = \sum_{A,k} \text{Pr}(A,k) \log_2 \frac{\text{Pr}(A,k)}{\text{Pr}(A) * \text{Pr}(k)}$$

Pr_A and Pr_k are the marginal propability distributions of activity A and position k, $Pr_{A,k}$ is the joint probability distribution of A and k which is the product of Pr_A and Pr_k if A and k are mutally independent [15], resulting in mutual information of 0 [14]. The equation can be rewritten to represent the mutual information conditioned on a set of actives [14].

$$I_{A,k} = \sum_{A} \sum_{k} \text{Pr}(A)\text{Pr}(k|A) \log_2 \frac{\text{Pr}(k|A)}{\text{Pr}(k)}$$

A can in principle be any set of activity values, the author restricts it to active (1) and inactive (0) leading to four terms, where the presence of bit k is written as 1, its absence as 0. The four conditional probability values $\text{Pr}(0|0)$ to $\text{Pr}(1|1)$ in these terms can be determined easily from known sets of active $\{C\}$ and inactive $\{F\}$ compounds. $\text{Pr}(k)$, the probability of bit k present in the chemical space, can be set via a third compound database $\{P\}$ or a union of the active and inactive set. Subsequently, the mutual information between a bit state k and the activity A can be obtained [14].

Then the mutual information of the four terms has to be combined to an activity scoring function S_k including affirmative $A = k$ and dissmissive $A \neq k$ information, always focusing on the bit being present in active compounds [14].

$$S_k = \sum I_{A=k} - \sum I_{A \neq k}$$

In the last step, the mutual information of K bits has to be combined, which is done by summing

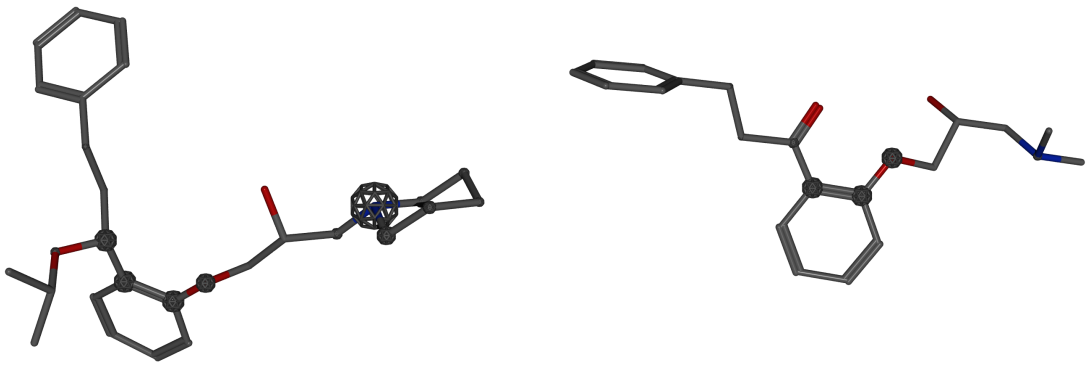
up [14].

$$S_Q = \sum_{k}^{K} S_k = \sum_{a}^{A} S_a$$

The equation already shows that the sum can also be broken down to the atom a, not only to the fingerprint bit! Each mutual bit information value S_k can be divided by the number of occurrences $n_{k,Q}$ in the compound Q giving one value per atom, the sum over all atoms is then taken as the atom mutual information S_a [14].

$$S_a = \sum_{k}^{K} \frac{S_k}{n_{k,Q} * m_k}$$

At this point, it is important to note that Williams includes a simple solution for bits k consisting of more than one atom, namely m_k atoms, which are weighted equally. Resultingly, all atoms of one fingerprint pattern are assigned the same values.



(a) GPV0264. The nitrogen carries the biggest positive contribution, a second center of contribution is on the central aromatic ring.

(b) GPV0048. The nitrogen only carries a small positive contribution, this is centered on the central aromatic system.

Figure 2.5.: POV-Ray™-exported results from an examplarily selected MIBALS model based on the reversed MACCS fingerprint and the propafenones dataset (see appendix B). Negative contributions are present, e.g. on the keto oxygen of subfigure (b), but their wirespheres are too small to be noticed in this depiction.

The MIBALS tool then uses a so-called “fragment scoring” to display the atom results as shown in figure 2.5. Fragments are highlighted which contribute positive or negative to activity of the compound, relative to the active core of the compound series in select MIBALS models, which have an $F \neq \{\}$. The magnitude of S_a is displayed in a wirespheres visualization, where the radii of the spheres $r_a = \frac{S_a}{100}$. Positive contributions are coloured in dark grey, negative contributions in red [14].

2.3. Rules-Based Methods

Fragment- and fingerprint-based methods require an attributes-based representation of the compound structure, as do many traditional algorithms used in (Q)SAR. In topological indices, local specifities within the molecule are combined into one scalar or tested as local attributes as in CoMFA, including disadvantages as a compulsory alignment of the compounds. Another concept would be to describe this knowledge directly, e.g. as "a benzene ring connected to a nitro group" [16], which is a relational description most attribute-based methods can't use [16].

2.3.1. ILP (Inductive Logic Programming)

One approach capable of working with relations is ILP (inductive logic programming), a non-parametric and nonlinear machine learning method, which was originally chosen by King et al. because it is "designed specifically to learn relationships between objects." [17] Additionally, human understandable symbols can be used to represent a problem, background knowledge can be added and the system delivers humanly comprehensible rules which directly can be checked with existent experience for consistency [17].

In contrast to the propositional logic, ILP includes first-order logic, *predicates* used to handle relational data between objects. A bond between e.g. atoms A and B is described as $\mathsf{bond}(A, B)$. Background knowledge also taken into account may include distances or various conformations.

Altogether, ILP programmes accept background problem knowledge (the Prolog facts) and a set of active and inactive compounds as input, as an example written in the language Prolog. The method is based on the concept of the "relative least general generalization" [17]. It tries to find this set of rules explaining the most actives possible while including the fewest number of inactive compounds. Therefore, it first starts with a single rule (e.g. a $\mathsf{bond}(A, B)$ predicate), which is then enlarged to maximize the gap between positives and negatives covered by the set of rules [16].

If a further extension is not useful, the new rule is kept and all explained positive compounds are removed from the dataset, while the wrongly classified inactives are kept. Then the generation of a new rule is triggered until all active compounds are treated. The set of rules, the "theory explaining the problem under study" [16] is returned as the programme output [16].

This setting would require a splitting of the dataset into actives and inactives and only produce binary results, which is not always useful. An example of this application can be found in [18], where Srinivasan et al. investigated the prediction of mutagenicity potential of nitroaromatics

using various methods, among them the ILP-programme "Progol" as a relational based model and the feature-based artificial neural networks, linear regression and tree-based methods.

King et al. demonstrate a rank-based model to predict trimethoprim analogues activity at the dihydrofolate reductase in [17], which is done in the ILP-programme "Golem" developed by Muggleton and Feng in 1990 [19]. "Golem" has already been superseded by various better implementations, e.g. "P-Progol" and later "Aleph" by Srinivasan [20]. Nonetheless, the rank-based "Golem"-model clearly shows the principles of ILP method and conclusions that can be derived from it.

For "Golem" input, King et al. created three files, one containing pairs of greater activity, e.g. great($d20, d15$) giving the example of compound $d20$ having a higher activity than drug $d15$. Similarly, the file providing the negative examples is a pairwise list of lower potency. Finally, the background knowledge is included in a third file. As the trimethoprim analugues are varied at three positions 3,4 and 5 and the common core is assumed to contain the same activity pattern in all cases, this file only includes a chemical structure representation of the substituents as predicates (struc (A, B, C, D), A equals the name of the compound, B, C and D represent the substituents at positions 3,4 and 5) and predicates coding properties like polarity of these substituents (e.g. polar(I,polar3) coding the polarity of the iodine substituent as 3). Altogether, nine properties were coded (polarity, volume, flexibility, presence and strength of hydrogen bond donors and acceptors, presence and strength of π- donors and -acceptors, polarizability and σ-property) [17].

After assigning the files, "Golem" retrieved nine rules predicting the relative potency of two compounds of the training set with 44 drugs. The results gave a significantly ($p = 0.05$) better Spearman rank correlation than the rank obtained from a Hansch analysis for the training set, for a test set ($n = 11$) the rank correlation was not significantly better than for the Hansch analysis [17].

As an example, the first rule derived was

great(A, B) :- struc(A, D, E, F), struc(B, h, C, h), h_donor(D, h_don0),
pi_donor(D, pi_don1), flex(D, G),less4_flex(G).

It is very easy to derive a self-explanatory interpretation of the rule [17]: Compound A is more active than compound B if

- drug B is unsubstituted at positions 3 and 5 (coded by h in struc(B, h, C, h)),
- drug A has a hydrogen donor of type 0 at position 3 (variable D),
- drug A has a π-donor of type 1 at position 3 (variable D) and

- drug A has a substituent less flexible than 4 in at position 3 (two sets: first the importance of flexibility by flex(D, G) with the second condition less4_flex(G))

The results include the so-called *coverage* coding the number of examples represented correctly and incorrectly, for the given rule the *coverage* is 119/0 on the training- and 105/0 on the testset, meaning that all relationship pairs are represented correctly [17].

As shown with the example above, rules based methods allow to derive common sets of clearly interpretable rules applicable to most compounds of a dataset. Depending on the input data, these rules contain substructural information up to the atom level. Resultingly, the user gets to know which features are important at which position and a relative score of their contribution to activity.

2.4. Topological Substructure-Based Methods

Topological substructure-based techniques are intended to visualize and determine SAR information of large datasets and don't rely on a specific QSAR-equation. Their aim is to highlight sets of structures having a systematic variation in only one or a few parts to visualize common features or allow for pharmacophore comparison [21]. If combined with activity data, the influence of single variations can be visualized easily, which allows to draw conclusions on the contribution of substructural molecule parts. Many programmes and methods are available, e.g. the Sheridan and Miller approach, Tripos Distill or MOE SAReport.

2.4.1. The Sheridan and Miller Approach

Already published in 1998, Sheridan and Miller's approach creates substructures of pairs of molecules and scores them. The following visualization presents pairs of molecules sorted by decreasing substructure score [22], which allows to identify similar molecules and their differences. In a second visualization option referred to as "scored atom" all atoms get one point for each occurrance within a significant substructure. The individual atom scores are then projected onto the molecule using different colours, which allows to identify conserved substructures [22].

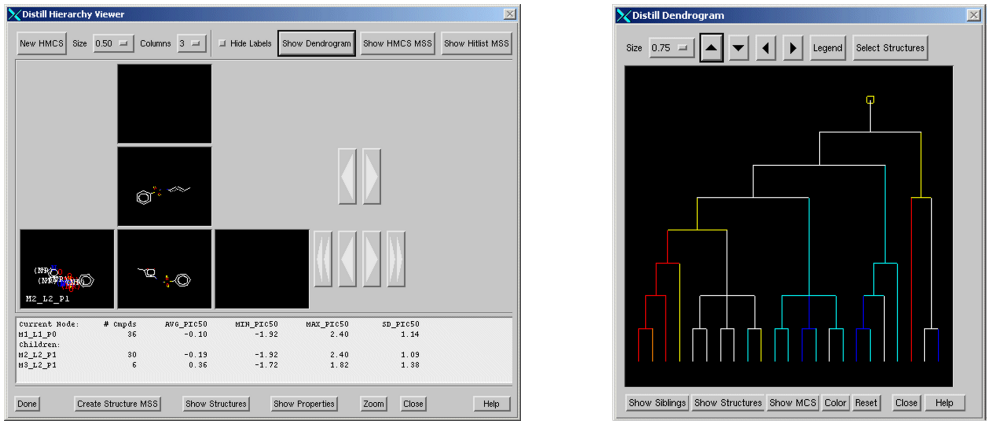
2.4.2. Tripos Distill

Within its SYBYL package, Tripos offers Distill, which is a "hierarchical clustering tool that classifies compounds according to their common substructures and organizes the results in a display that enables visualization of SARs" [23]. First, the compounds are clustered dependent on their common structural core, with the node in the graphical visualization containing a colour-coded hint at the average activity of the node's substances. Then the user can navigate through the roadmap, which also displays incremental changes between the structures and how they affect activity [23].

The main focus of Distill is the common substructure determination procedure [24], for which calculation it offers multiple options. They include the differentiation of atoms, bonds or bonds in rings; inclusion of hydrogens and fuzzy atoms which allow for one-atom modification when searching e.g. large backbones.

The most important MCS structures are then identified using a scoring function, which compasses number of atoms and bonds, ring bonds, as well as number of hetero atoms and branch atoms. It can further be controlled by setting weighting factors for ring bonds, hetero atoms and branch atoms.

During the clustering, structures are assigned the highest-ranked substructure node. Options on setting the target group size enable the user to define the approximate size of one cluster.



(a) Hierarchy viewer showing the common substructure of nodes.

(b) The dendrogram view in Distill. The whole dataset cluster is shown.

Figure 2.6.: Tripos Distill's main user dialogs showing the whole dendrogram tree (b) and one detail containing one node and its parents and siblings in the hierarchy viewer (a) (taken from [24]).

The Distill Hierarchy Viewer (figure 2.6) then presents the detected hierarchy as well as the common structure and activity values typical for the nodes and its parent and siblings. Buttons give easy access to the compounds' structures of one node and their detailed activity values. Additionally, the whole dendrogram can be visualized including colour-coded activity information, which ranges from red (largest) to blue (smallest).

2.4.3. MOE Structure-Activity Report

Starting with MOE 2008.10, the Structure-Activity Report offers similar functionality as Tripos Distill. In contrast to Distill, which is based on maximum common substructure generation, the MOE approach implements the Schuffenhauer scaffold searching algorithm [25]. It creates scaffolds by extracting the "molecular framework", which is retrieved by iteratively removing the sidechains and subsequently one ring after another starting with the least scaffold-like [25], thus avoiding problems arising from pairwise substructure generation [26], which to solve requires high computational efforts (compare 3.4.3).

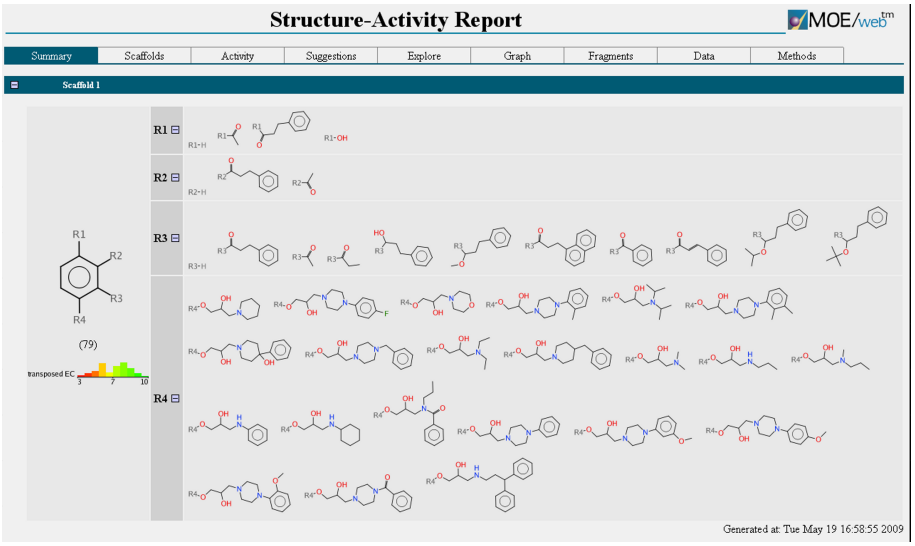


Figure 2.7.: The only common scaffold and its R-groups the Structure-Activity Report retrieves for the propafenones dataset (appendix B). The whole activity range is covered due to the transposed activity values.

The Structure-Activity Report tool then assigns a score to the scaffold as does Distill, it includes the fraction of compounds containing the current scaffold, the heavy atoms, bond parameters, the number of fragments selected in previous iterations and their Tanimoto coefficient using MACCS fingerprints. At first, the molecules containing the highest-ranked scaffold are selected and removed, then the remaining compounds are rescored until the best score is below one or no

molecule is left. Subsequently, the scaffolds generated are aligned to find common substitution points and then assigned to the molecules [26].

The output of the tool can be viewed in a web browser. It gives an overview over all retrieved scaffolds and all remnant groups substituted to them. The scaffolds are connected with an activity range, if provided; and when hovering over an R-group, the compounds' structures including both the R-group and the scaffold pop up. It is also possible to display a correlation matrix showing the activity ranges of two components, where both an R-group and a scaffold can be selected. This provides a good overview on how different scaffolds or R-groups contribute to activity. Additionally, the tool tries to make suggestions on how to modify the molecule on basis of the results.

Further options include a fragment tree displaying the scaffold hierarchy with respect to compounds containing the scaffold as well as a basic graph tool which allows for comparing different scaffold series in terms of $logEC_{50}$ efficiency, MW or $logP$.

We use the propafenones data set (see appendix B and section 4.2) as an examplary input to the Structure-Activity Report tool. It is loaded via the Compute – SAReport menu in the database explorer in MOE, then the activity field has to be assigned. Optionally, scaffolds can be entered manually, which nonetheless failed when providing the database's MCS structure. The output is written to a directory on the hard drive and can be investigated later.

Only one common scaffold was detected, namely the common central aromatic ring of the propafenones, which is included in all structures. Unfortunately, in some cases the correct detection of the scaffold is suboptimal and other benzene occurrances but the central system are taken, which leads to ambiguities concerning R-groups and substances.

In our case, the R-groups are mainly used to detect differences in activity. The activity scaling is fixed, resultingly we have to transpose our activity values to the range of 3–10. Figure 2.8 displays a part of the substituents connected to R3 and R4 of the common scaffold (see figure 2.7). It can be seen that the activity generally increases with increased lipophilicity and that an aromatic system is necessary for activity for the same R4 (first R4 line of figure 2.8).

2.5. Partial-Least-Squares Components-Based Methods

A partial least squares (PLS) regression analysis incorporates more information than the simple overall equation unveils. This is caused by the equation's coefficients representing more

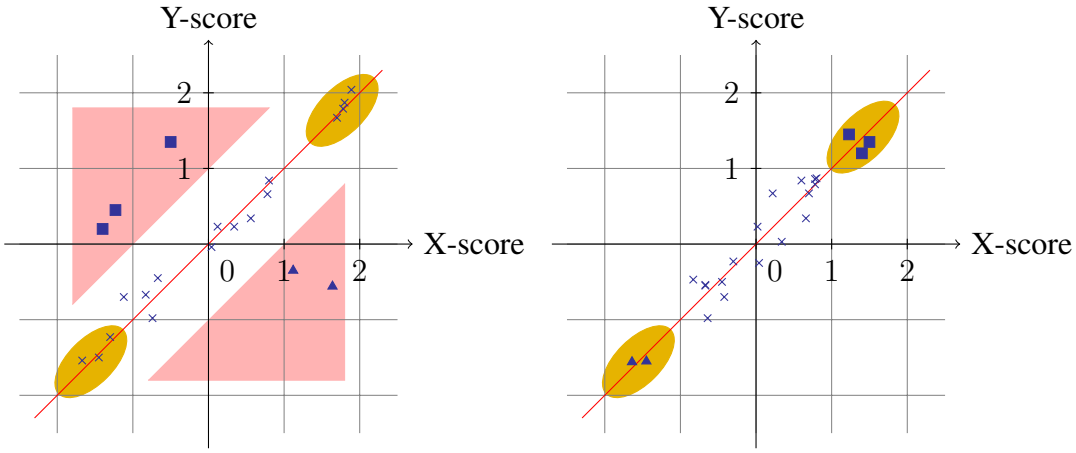
Figure 2.8.: The correlation matrix shows the activities of various R3 and R4 substitutions of the scaffold presented in figure 2.7. One cartwheel section represents one compound, the pop-up shows additional information displayed during hovering. The contribution of the substituents can be derived directly.

than one structural feature or trend. Nonetheless, in order to interprete a QSAR equation correctly, first the knowledge of which descriptor codes which molecular features and second the knowledge of where changes in structure lead to different properties are necessary [27].

2.5.1. PLS (Partial Least Square) Score Plot

The PLS Score Plot method by Stanton [27] is based on a PLS regression carried out on a finalized and validated model. In that way he obtains score plots and property weights for the components of the PLS regression explaining most of the variance within the property values. Investigation of these plots then leads to physical interpretation derived from the components, the so-called *structural trends* [27].

The PLS Score Plots are calculated for validated components only. Stanton defines these as all components for which the predicted sum of squared errors does not increase compared to the previous set of components, or, alternatively, where the leave-one-out crossvalidated values do not decrease. The score plot for the i^{th} component then includes the observed activity values at the Y-axis (Y-scores) and the predicted activity values at the X-axis (X-scores) [27]. The X-scores are a linear combination of the predictor variables, projections of the observations on



(a) Score plot component 1.

(b) Score plot component 2.

Figure 2.9.: The subfigures show fictional score plots of a first and a second component with exemplary weights taken from table 2.1. The first component in (a) focuses on the compounds included in the upper right and lower left corner which are highlighted in orange, triangles are overestimated and squares show underestimated compounds. A closer look on (b) then shows that the second component focuses on the wrongly estimated compounds of the first component [27].

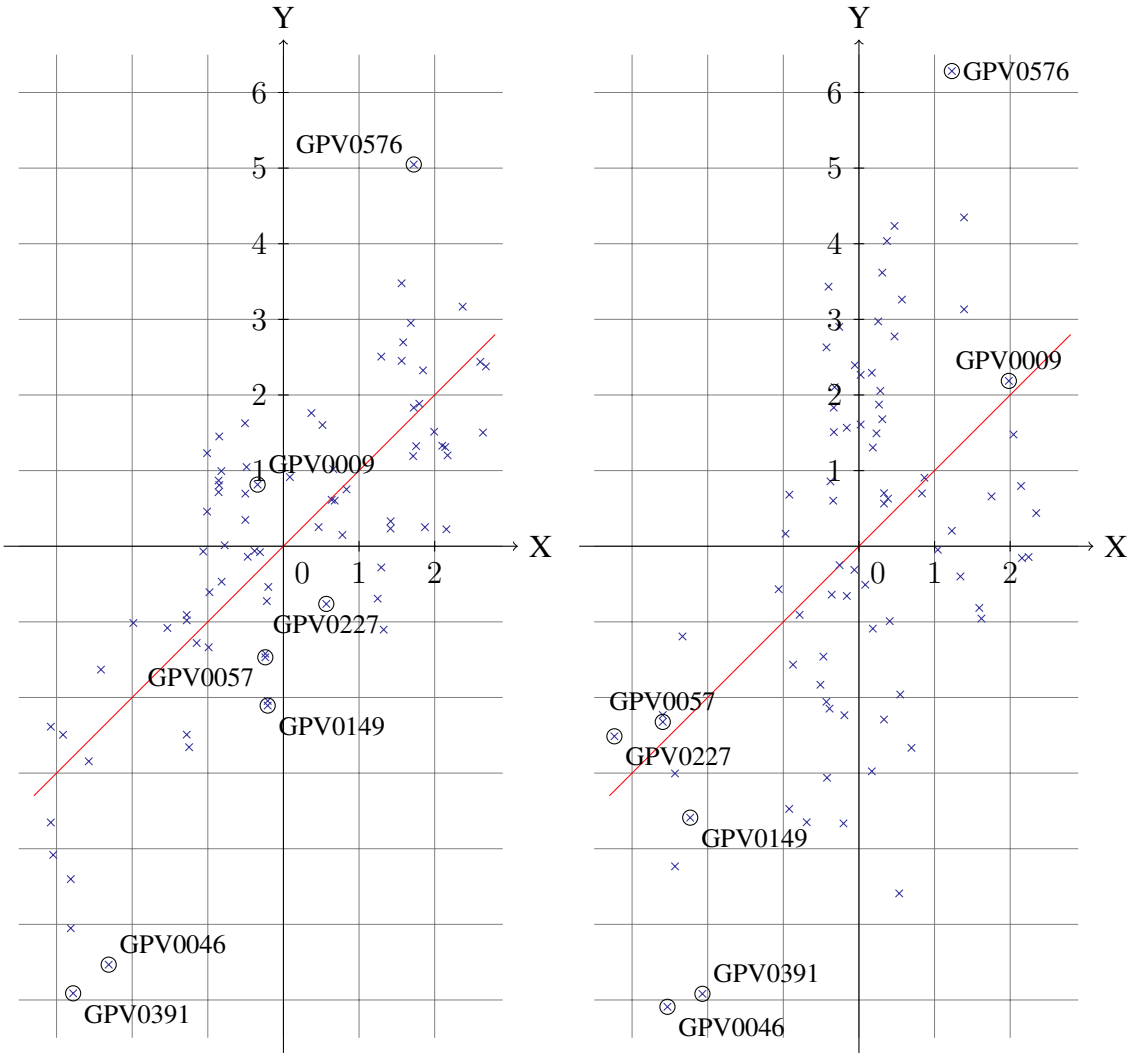
the PLS components. Similarly, the Y-scores are linear combinations of the response variables. Both form one $n \times m$ matrix each, where n is the number of observations and m the number of components [28, PLS, results]. Additionally, the X-weights are needed [27], which represent the correlation between the variables and the Y-scores. They are typically similar to X-loadings and used for interpretative purposes [29]. The sign and the magnitude of the weights assigned to each descriptor unveils the focus of the component and subsequently how changes in these emphasized features influence the property [27].

Figure 2.9 gives a hypothetical four descriptor model, the weights for the first two compounds are listed in table 2.1.

Descriptor	**X-Score weight 1**	**X-Score weight 2**
Desc-1	-0.113	-0.431
Desc-2	0.572	0.035
Desc-3	-0.498	0.022
Desc-4	-0.034	0.537

Table 2.1.: A fictional descriptor-weight for the first two components. For the first component, Desc-2 and Desc-3 are the most important descriptors, for the second Desc-1 and Desc-4 [27].

When applying the approach on the propafenones dataset using the QSAR-equation retrieved in section 4.2.2 using a test version of the proposed Minitab® 15 programme [30], we obtained six



(a) Score plot component 1.

(b) Score plot component 2.

Figure 2.10.: The score plots of the first two components of our examplary propafenones analysis. X denotes the X-scores, Y the Y-scores of the component. The regression lines were determined using Minitab® 15.

components for six predictor variables, five of them are valid in terms of Stanton [27] and are taken for the model. For the final QSAR equation, Minitab® 15 presents the same data as MOE in section 4.2.2. Table 2.2 shows the valid components, their residual sums of squared error (Error SS), the cumulative R^2 (Cum. R^2), the PRESS and the cumulative Q^2 (Cum. Q^2).

Table 2.3 presents the X-weights of the first three components as returned by Minitab® 15. For the first component, chi0v_C is the most important and SMR_VSA5 and PEOE_VSA_POS are also contributing descriptors. Mispredicted compounds should be corrected by the following principle components, where the PEOE_VSA+1 is the most important descriptor of the second and PEOE_VSA-0 of the third component, both with a negative influence.

Component	Error SS	Cum. R^2	PRESS	Cum. Q^2
1	25.064	0.597	27.496	0.558
2	19.046	0.693	21.221	0.659
3	15.667	0.748	18.281	0.706
4	14.937	0.759	17.571	0.718
5	10.910	0.824	13.083	0.790

Table 2.2.: Results of the PLS analysis of 79 propafenone-like p-GP inhibitors as retrieved from Minitab® 15

Example plots of the first components' scores (figure 2.10) give ambivalent results. The biggest outliers of component 1's plot (figure 2.10a), GPV0391, GPV0046 and GPV0576 are also among the biggest outliers in plot 2 (figure 2.10b). Nonetheless, it has to be noticed that GPV0046 and GPV0576 are defined as outliers over the whole equation, too; resultingly subprediction can't be accurate (compare section 4.1.1). Additonally, many structures are accounted for in the third or fourth component, which are not shown, e.g. GPV0149. For some compounds that are off-diagonal in plot 1, the correction can be seen in plot 2, e.g. GPV0057, GPV0227 or GPV0009 (figure 2.10).

Descriptor	Component 1	Component 2	Component 3
chi0v_C	0.682	0.268	0.129
PEOE_VSA+1	0.173	−0.749	−0.498
PEOE_VSA-0	0.034	0.375	−0.7465
PEOE_VSA_POS	0.438	−0.151	−0.177
SlogP_VSA9	−0.043	0.451	−0.354
SMR_VSA5	0.557	0.035	0.153

Table 2.3.: The X-weights for the first three components of our examplary analysis.

2.6. Fragment-Descriptor-Based Methods

As some descriptors are built upon a simple combination of values derived from substructures of the molecule in question (so-called fragment based descriptors, see section 3.2), it is possible to calculate one subfragment's contribution to a classical QSAR equation, which gives hints on which atoms are good for activity and which are not.

2.6.1. RQSPR

Within his tool “visdom”, Gombar included the RQSPR approach. It focuses on projecting back molecule properties gained via a normal QSAR run back onto the structure and on substructural parts (figure 2.11).

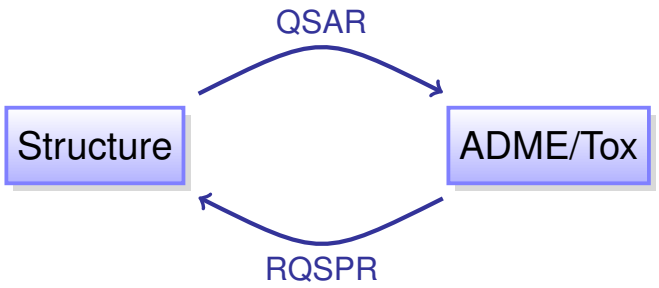


Figure 2.11.: The RQSPR approach tries to project back molecule properties gained via QSAR onto the substructure to extract information supporting the chemist [4]. It is the basis for the new MOE script.

These properties are then visualized on a per-atom basis in the molecule viewer of “visdom”, with different colours as well as the value indicating whether an atom is contributing positively or negatively to the whole molecule’s activity predicted by a QSAR model. Obviously, it is also possible to load new molecules into the visdom window (see figure 2.12). Unfortunately, no further details about visdom and its capabilities have been published by June 2009.

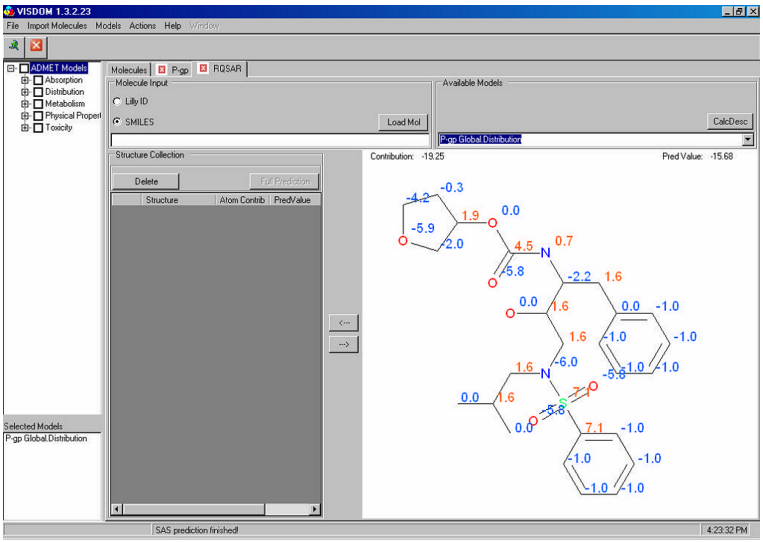


Figure 2.12.: visdom programme window showing an RQSPR result based on a predefined model. Atom contributions are indicated using different colours as well as their values [4].

3. A New MOE-Script for Improved Visualization of PLS-QSAR-Equations

The new script compasses both an in-depth exploration on submolecular contributions to the QSAR equation as well as visualization methods supporting the user in focusing on the critical parts of the molecule.

The QSAR exploration part is based on the RQSPR approach introduced by Gombar [4], for the visualization various methods are available including an MLCS (multiple largest common substructure) algorithm and additional colouring options.

3.1. Introduction to Molecular Operating Environment

In order to visualize descriptors on a per-atom basis, it is necessary to integrate descriptor calculation as well as its visualization on the molecule into one programme. A completely new implementation is definitely beyond the scope of this script, as it would also have to deal with graphical molecule representation. Additionally, as the whole script should be easily integrated into normal QSAR workflows, we would also need various import and export functionalities. Therefore, we choose to create an add-on to an existing chemical computing toolkit, which, as a special requirement, should have a good general scripting capability and its descriptor calculation routines available as source code to extract necessary routines.

For the Java programming language, the Chemistry Development Kit (CDK), which can be obtained from http://cdk.sourceforge.net, is available as open source software allowing to investigate and modify its descriptor routines. CDK offers 2D as well as 3D (integration with JMol) depiction of molecules and many features like aromaticity detection, input/output libraries, SMILES [31] strings functions, some descriptors and a simple maximum common substructure finding tool [32, 33], which are of potential use. Nonetheless, the CDK is only

a library to provide a basis for a programme, and not a programme on its own. It lacks an integrated user interface implementing its functions as well as database management tools, thus making an optimal workflow integration difficult.

Chemical Computing Group Inc. offers MOE (Molecular Operating Environment) [5], "an interactive, windows-based chemical computing and molecular modeling tool with a broad base of scientific applications" [34]. For our purposes, it includes structure-activity analysis, descriptor calculation and visualization tools like 3D molecule depiction. Furthermore, MOE is an open architecture with a built-in programming language, SVL (Scientific Vector Language) [34], a vector-oriented scripting language. In a vector-oriented programming language, the vector is the data primitive, operations can be applied to a whole data set organized in vectors using one command. Resultingly, the scripting environment enables flexible extensions of the programme at a relatively high performance [34, Introduction — SVL]. Many functions of the basic installation are available in the directory $MOE/lib/svl as SVL script files having the extension *.svl, and can therefore be investigated thoroughly and adapted if necessary. Additionally, the powerful SVL language allows a scripted call of nearly all programme features. MOE also has an integrated database as well as JDBC connectivity. Therefore, we decide to base our script on MOE.

3.2. Descriptor Availability and Recalculation

A recalculation of descriptors is necessary to obtain atom values, as they are typically not provided when the standard routines implemented in MOE are called. To cope with, the procedures have to be reengineered and reimplemented. Additionally, the fragment size of a descriptor has to be determined, as it is the main factor deciding if a recalculation on an atom basis is possible.

3.2.1. Fragment Based Descriptors

The idea of visualizing atom contributions of descriptors is based on structural theory, the core of which is a structural formula. A structural formula consists of functional groups — fragments carrying a set of properties. These assumptions allow to introduce additivity in the calculation of descriptors, which means to partition a molecule into its structural fragments and, after assigning descriptor subvalues to the fragments, simply sum up the values to get the property for the whole molecule [8]. A property A of a molecule can be partitioned into atomic

or bond contributions with n_i being the frequency of occurrence of a certain fragment of type i in the molecule and A_i being its contribution. N is the overall sum of fragment types i [8].

$$A = \sum_{1}^{N} A_i * n_i$$

Zefirov and Palyulin also notice an analogy of this fragment approach to the classical Hammett approach improved and modified by Hansch [1], who differentiates between a core and its substituents and assigns different properties to different molecule parts, giving them a specific contribution. This contribution value of the substituent, which can also be one atom only, e.g. —F, is calculated by

$$\pi = logP - logP_H$$

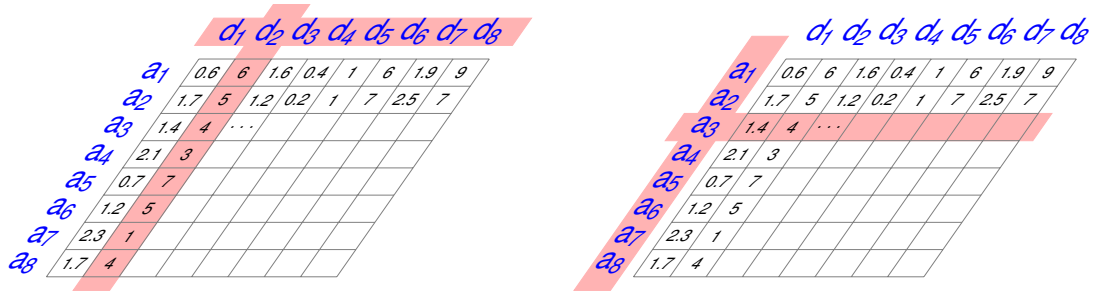
$logP$ stands for the logP derived from core plus substituent, and $logP_H$ equals the logP of the core with H as substituent [1]. Although these contributions are only relative to hydrogen and thus not additive in the same terms as we used before, the principle validity of assigning values to submolecular fragments and using these as independent descriptors is shown.

Furthermore, if a property describing a submolecular part as well as descriptors consisting of contributions of submolecular parts are valid (and they have been used since the beginning of QSAR), it seems to be reasonable to split up a descriptor to its fragmental parts again and take these values as an input for visualization. Furthermore, Guha declares that "certain topological descriptors are interpretable, especially when they are fragment based" [2].

The process can also be described as an inversion of the mathematical creation procedure for obtaining a molecule's estimated property. In normal cases, for each atom-based descriptor the sum $d_x = \sum a_{xy}$ over all individual atoms' partial properties a_{xy} is built first, then the sum over all weighted descriptor sums d_x gives the predicted property. It is perfectly accepted to extract and compare the atom-based descriptors part of the equation (figure 3.1a).

In our case, at first the $a_y = \sum d_{yx}$ is summed up, thus an individual value for each atom consisting of the sum of all weighted descriptor values d_{yx}. This information is then visualized in the script, it gains an additional view on the equation by returning one value per atom instead of one value per descriptor. Of course, theoretically the sum over all atoms can be formed, as a mathematical fact it is equal to the predicted property of normal QSAR (figure 3.1b).

Estrada et al. used atom contributions of fragment descriptors like PEOE atomic charges, molar refraction or polar surface as an input for their TOPS-MODE (Topological Sub-Structural Molecular Design) approach, which in contrast to our approach is based on bond matrices and thus requires a weighting [35].



(a) Traditional property prediction for fragment-based descriptors. First a descriptor sum d_x is created over all atoms a_y, then the weighted descriptors are added to the property value.

(b) Modified property prediction with atom-based descriptors giving a per-atom value. First an atom sum a_y is created over all weighted descriptors d_x, then the atom values are summed up.

Figure 3.1.: From a mathematical point of view, both methods of obtaining a molecule's property value are the same.

3.2.2. Descriptors Consisting of Larger Fragments

The limitation to fragment-based descriptors is accepted in general for "techniques that color the atoms and the bonds according to whether the fragment contributes positively or negatively to the QSAR", which are noticed as "also have a role to play" [3]. In principle, fragments can have any size of atoms, as long as they are smaller than the molecule. But the process of tracing back contributions down to the atom level is definitely easy for fragments having a size of one atom, while it can't be done intuitively for larger fragments. Here, we can in principle either:

- visually combine atoms to sub-fragments or
- weigh atoms giving them only a part of the fragments' contribution.

The first method would especially be suited for a fragment size of two atoms, as long as the atoms are connected by a bond. The value can be assigned directly to the bond giving it a contribution to the equation. The number of values to display rises by 50% if one-atom fragments are shown in parallel. Consecutively, the visualization and the perception of the contribution distribution is more difficult and less intuitive. For the settings we choose, the internal visualization of MOE gives an additional problem: MOE's bondings are not objects of their own, but built automatically as a combination of the atoms taking part. Resultingly, we can neither address them as graphical objects nor assign them individual attributes like colours.

For fragments larger than two, a combined visualization is difficult due to the overlay of single atom and multiple atom fragments of potentially different sizes, which makes an explicit colouring of one atom impossible, again resulting in a loss of distinctiveness. To solve that problem, we could e.g. split up the QSAR-equation subject of visualization into layers of equally sized

fragments, resulting in a rather parallel view of atom and fragment contributions than a simultaneous representation of the QSAR-equation as a whole, which we try to provide.

To sum up, objects of any type visualizing a contribution could be placed over the bonding for two-atom fragments, which could be realized in upcoming versions of the script. Larger fragments could be dealt with using a parallel view mode.

The second method relies on breaking down the value of the fragment to sub-units consisting of one atom. As the descriptor simply isn't calculated for atoms, this method inherently leads to interpretation. The easiest option seems to be a division of the value by the number of atoms belonging to the fragment, which would intuitively be accepted for a bond attribute divided and assigned to the two binding atoms — and it might even be correct as long as only simple bond descriptors like the number of double bonds are taken into account. In that case, both participating atoms could be given the factor 0.5 of one double bond. Nevertheless, that process ignores the fact that a binding property is not always substantiated equally by both atoms. As an example, the two atoms-sized fragment χ-indices of order one by Kier and Hall (see section 3.3.4) are calculated by multiplying the individual δ-values, which of course can be different for both participating atoms [36]. Although the descriptor itself is not weighted, the question remains if a simple division would represent the original meaning correctly as the sum might heavily be dependent on one atom only.

Estrada et al. inverted the proposed dividing method and used it for getting a bond value out of two individual atom contributions, the weighting in terms of contribution being calculated as

$$w_{(i,j)} = \frac{w_i}{\delta_i} + \frac{w_j}{\delta_j}$$

with w being the assigned values of atom i and j and δ representing the vertex degree or number of bonds of the atom resulting in $w_{(i,j)}$, the value for the bond between atoms i and j [35]. This method is comparable to simply dividing a bond's value by two, as here the atom's values are divided by the number of bonds giving equal assignments to all bonds of one atom, only dependent on its degree.

As difficult as a correct weighting seems to be for two involved atoms, it is even more inadequate for larger fragments, as those tend to increasingly represent the whole molecule (compare [36, p. 378 f.]).

To conclude, one of the fundamental conditions set for the script is giving a clearly structured graphical visualization without any interpretation. A backprojection based on a descriptor's value for each atom can only be done if and only if the descriptor's value for the whole molecule constitutes of a mathematically reversible combination of input values, at least one for each

atom in the molecule. Inherently, as not all descriptors fulfil this criterion, our script is not only conceptually restricted to so-called fragment-based descriptors, but also to descriptors which fragment size is one atom or less.

Hence, we prefer the restriction to fragment-one descriptors over a general method including interpretation able to visualize fragments of any size for the sake of distinctiveness in visualization.

3.2.3. Descriptor Calculation Routines

As the script is constructed as a MOE script using MOE's own scripting language SVL, the primary focus of descriptor selection is set on those included in the MOE package. Altogether, an out-of-the-box MOE 2007.09 installation offers calculation of 251 descriptors, of which 67 are internally classified as 3D-type, and subsequently 184 as 2D-based [37, Database Viewer — Compute — Descriptors]. The diploma thesis only concentrates on the 2D-based descriptors, although 3D-descriptors might also have one atom fragments.

Descriptor Calculation in MOE 2007.09

MOE descriptors are organized in the subdirectory ./lib/svl/quasar.svl of the $MOE directory, grouped by descriptor types in different files. A designated function, `QuaSAR_list_*`, lists all descriptors available in a certain file, the file-wide calculation algorithm is included in the function `QuaSAR_calc_*`. A central file also defining the GUI, qtools.svl, extracts the descriptor codes and descriptions out of the descriptor group files already loaded into the system by calling `sym_find_f`. It tries to find functions starting with `QuaSAR_list_` and lists them. Additionally, qtools.svl provides `QuaSAR_Descriptor_Calc [qcalc, mol]`, which serves as a sort of wrapper for calculation functions. Its arguments are a list of descriptor vectors and the molecule they have to be calculated for. It initiates calculation of each single descriptor using `call` [37, qtools.svl].

This concept allows for exposing a list of computable descriptors and a single central calculation function per descriptor group file only, while the calculation methods themselves can be declared as `local`. Subsequently, they accept only internal access from within the file itself and can't be called from the system, exposing only the final result of calculation functions, which is only one value per molecule.

Concept of Recalculating Descriptors in the new MOE SVL Script

Because MOE's calculation functions are declared as `local`, our SVL script can't use the standard procedures. The descriptor calculation algorithms have to be reimplemented. For that purpose, a similar concept is introduced. `Calc_Descriptors`, a calculation wrapper function, serves as a central function setting up the result vector `v` holding all values and calls the respective sub-calculation function. `Calc_Descriptors` takes two arguments, the MOE-internal molecule key referencing the already drawn molecule and a vector holding the alphanumerical descriptor identifiers of the input PLS-equation. These identifiers are provided by MOE and unique for each descriptor.

In contrast to MOE's own strategy, the different functions responsible for descriptor groups are included in one main file, qsar_backprojection.svl, following the convention `Calc_*`. To avoid name clashes with other MOE functions, the functions were declared as `local` as in the MOE concept. We also don't search for descriptors, as `sym_find_f` only detects functions set as `global` [34] and therefore is not appropriate for our structure. Additionally, we don't provide listing functions. On the one hand, these changes reduce the general programming effort needed by avoiding global list and group functions, where tracking of atoms and atom definitions would be quite complicated, as we deal with atoms and not with one value per molecule only. On the other hand, some problems and disadvantages occur due to our simplification.

- The main file is big and confusing exceeding 2,000 lines of code.
- The clear concept structure of independent descriptor files is not existent, thus lacking flexibility as well as coherence as calculation of one descriptor is distributed over the whole file.
- Adding new descriptors can only be done by changing the main file.

The calculations themselves are grouped in the functions `Calc_*` and expect three or four arguments, the value vector, the descriptors which should be calculated and the atoms. The optional fourth argument is the molecule in MOE's MDB (molecule database) format as returned by the function `mol_Extract[]` [34, MOL Data Structure], which is sometimes needed to extract saved information not available via the individual atoms. To avoid useless calculation steps, the group functions are only called if descriptors belonging to them have to be determined. Only if the `indexof`-step searching for the specific group's descriptors in the QSAR-list returns true, the group's calculation function is called. All descriptors belonging to that functions are determined at once.

```
1 if anytrue indexof[descr,smr_vsa_descriptors] then
2    //SMR_VSA calculated skipping H atoms
```

```
    values_woH = Calc_SMR_VSA[values_woH,descr,atoms_woH];
    woH_indices = cat append[woH_indices, pack indexof[
        smr_vsa_descriptors,descr]];
endif;
```

Listing 3.1: Sample decision step for SMR_VSA-descriptors.

A Special Case: Hydrogens. Hydrogens have to be dealt with separately. For many descriptors, hydrogens aren't autonomous atoms. Although hydrogens are or at least could be assigned partial values, their values are combined with the binding heavy atom to hydrides in a later step. To mimickry that behaviour, two arrays of all atoms have to be created, `atoms` holding all molecule atoms and `atoms_woH` consisting only of heavy atoms. This is caused by descriptor calculation algorithms returning values in dependence of hydrogens being attached to the atom or not. Two arrays of atoms provide the calculation algorithms with the correct atoms. After the process is finished, both arrays are aggregated into one including hydrogens. For this reason, all descriptor indices calculated without hydrogens are saved in `woH_indices` (see listing 3.1) and then looped over, assigning the `values_woH` to the inverse hydrogen-masked main descriptor values variable `values` (listing 3.2).

```
for i in woH_indices loop
    local holder = values(i);
    (holder|inv_H_mask) = values_woH(i);
    values(i) = holder;
endloop
```

Listing 3.2: Aggregating non-hydrogens into hydrogens

Some descriptors additionally require to specifically consider hydrogen values, which are discussed in the corresponding details section. Hydrogens cause VSA descriptors to deliver different overall sums (see section 3.3.3). For the apol-descriptor, MOE always adds hydrogen causing problems with atom numbering (details in section 3.3.2) and other descriptors.

Adding a new descriptor group. As pointed out, some steps have to be performed in the main file qsar_backprojection.svl to add a new descriptor group:

- A new `static` variable has to be declared following the convention `*_descriptors`. Ideally, its name characterizes the descriptor group
- The static variable has to be initialized as a vector containing the alphanumerical MOE abbreviations of descriptors it is responsible for.

- Then it has to be added to the main variable `descriptors` using **`cat`**.
- A new **`if ... then`**-control structure is needed in **`Calc_Descriptors`** to kick off using the new function if part of a QSAR-model. The corresponding **`Calc_`**$*$-function has to be called and the return values have to be added to the value holder variable `v`.
- The need of hydrogens has to be determined and subsequently the **`if ... then`** conditional structure modelled as required.
- Finally, the calculation function according to the convention **`Calc_`**$*$ must be included and return the calculated variables. The new function must also be added to the function list at the top of the script.
- To add a singular descriptor to an already existent group, it just has to be added to the corresponding list and its function.

3.3. Descriptors in Detail

Apart from the commonality of having a one-atom fragment size, descriptors we use for backtracking differ groupwise in how they are retrieved, which has to be reflected in our script. For the vast majority, our implementation is heavily dependent on how they are calculated in MOE, but various methods have to be derived on how their atom fragments can be extracted.

3.3.1. Partial Charge Descriptors

Altogether, MOE supports two versions of partial charge descriptors, the difference being the source of the partial charges. On the one hand, descriptors prefixed with Q_ are not triggering any calculation but using the atomic charge values stored with each molecule in the database, which can be extracted using `db_mol(4)(MOL_ATOM_CHARGE);` with `db_mol` being a molecule stored in MOL Data Structure in the database. This approach gives the possibility of performing external partial charge calculations, as example with various charge calculations implemented in MOE [34]. On the other hand, the prefix PEOE_ indicates a calculation based on MOE's Partial Equalization of Orbital Electronegativities (PEOE) values [38]. PEOE was developed to assign parts of the electronic distribution to specific atomic centers and is based on orbital electronegativity, which is defined as the electronegativity of a certain orbital taking into account the valence state of the atom. Further, PEOE considers influences of directly bonded neighbours using iteration steps resulting in q_i values for each atom [38].

As MOE's partial charge descriptors are generally set up by summing up individual atom's partial charge values, we can use the whole set of descriptors. They include for PEOE_* and Q_* the sum of positive and negative partial charges *_PC+ and *_PC− as well as various vander-Waals surface area descriptors (compare section 3.3.3) depending on charges, for instance the accumulated vdW surface area (VSA) for positively charged atoms (*_VSA_POS). Additionally, MOE provides fractional versions of the descriptors denoted with an additional F (e.g. *_VSA_FPOS). The specific descriptor VSA is devided by that of the whole molecule, resulting in a fraction. Mathematically we can split up the dividend to the single atom contribution and thus implement the fractional versions, too. Only the *_RPC-set giving the relative value of the highest positive or negative partial charges compared to the aggregated sum of charges is not usable, as this value is specific for the whole molecule.

3.3.2. Lipophilicity and Related Descriptors

Lipophilicity

Lipophilicity often plays a major role in drug absorption, plasma protein binding or hydrophobic interaction as well as solubility and is therefore a prime physico-chemical parameter [39]. As a result of this importance, there exists many calculation methods estimating logP, which are either fragment-, atom-based methods or consider conformation and approximate surface area [39]. MOE offers two versions of logP-calculation, too, with both being atom-based models [34]. These are classified as only medium-potent by Mannhold and Dross and only have a modest accuracy especially when focusing on complex drug-like structures. In general, atom-based estimation of logP is difficult with different algorithms giving up to 38% unacceptable results (defined as more than 1 log-unit difference to experiment) [39].

On the one hand, MOE integrates a linear atom based model developed by Labute [34, unpublished, q_logp.svl] relying on 100 descriptors representing different atom type features and rules. The training on 1827 small molecules results in an r^2 of 0.931. 94 rules are based on defined environments of a certain atom type, and for each atom of the specific type a SMARTS search is performed. SMARTS pattern language is derived from SMILES, and MOE uses SMARTS to search specific patterns in molecular objects. In terms of syntax, it also closely resembles the SMILES representation [34, SMARTS Functions]. We will denote a molecule pattern as SMILES, as long as it is not used in context of a MOE search, where we refer to it as SMARTS string. If the atom in question is matched, the specified rule applies, subtracting or adding a value to logP. From the remaining six parameters, four modify the logP value if certain interactions occur and are thus taking into account two atoms, one is designed for amino acids

and one specific for alkanes, the latter both accounting for the whole molecule. Although tracing back the 94 single atom descriptors can be done, the four interaction descriptors consisting of two atoms raise the problem of assigning weights as well as the problem of singularity (see section 3.2.2), and the alkane and amino acid descriptors can't be recalculated. As a result, Labute's implementation of logP can not be used in our script.

On the other hand, the second logP calculation available is based on Wildman and Crippen's strictly atom-based method where only atom classes account for intramolecular interactions. These have been previously assigned a certain contribution to the logP value which are dependent on neighbourship. After classification, the property is calculated by simply summing up.

$$logP_{calc} = \sum n_i a_i$$

$logP$ is the estimated logP value of the molecule, i are all 68 defined classes, n_i the number of atoms in the molecule assigned to class i, and a_i the logP contribution of class i [40]. The power of estimation of the MOE implemented Wildman and Crippen algorithm gives an r^2 of 0.918 with $\sigma = 0.677$ when trained on 9920 small molecules of the Medicinal Chemistry Project by Hansch and Leo [40].

The algorithm was also tested in a huge logP comparison study by Mannhold et al., where the algorithm was referred to as *ALOGP98* [41]. For all datasets, it was mostly classified as rank II, thus giving a better performance in terms of $RMSE$ than an *a priori* defined logP derived from the average of the dataset, thus an Arithmetic Average Model (AAM), but significantly worse than the best AB/LogP and ALOGPS methods [41]. The observed $RMSE$ values for ALOGP98 were 0.70 and 1.00 for two public datasets derived from an 266 compound public set; and 1.12 and 0.73 for an in-house Pfizer and Nycomed dataset with 95809 and 882 compounds, respectively. Nonetheless, it was amongst only 7 methods giving results better than AAM for both sets [41].

As one atom gives exactly one value, an atom-based interpretation of this logP variant is possible. The MOE implementation in q_slogp.svl classifies on basis of a SMARTS search and then assigns the values stored in constants. The built-in function **`aSLogPMRef[]`** offers calculation of MR and logP values atomwise. If the molecules are provided without implicit hydrogens, the values for them are added to the heavy atoms they bind to, forming hydrides, otherwise the functions gives them back explicitly. Of course, that behaviour does not change calculation of the whole-molecule descriptor, but atom-wise results are influenced.

Molar Refractivity

In contrast to the estimation of logP, the results of MR calculation are by far closer to reality [39]. Both MOE implemented algorithms claim to have an r^2 of 0.997, Labute's atom-type based linear model trained on 1947 small molecules [34, unpublished] having a slightly bigger σ-value than the Wildman and Crippen model, which is calculated analoguely to the logP value.

As a result, the script can cope with Wildman and Crippen's scheme as shown for logP using the same function, which returns the atom's MR-value as the second vector.

Labute's model is based on 11 descriptors, among them are apart from fragment-one sized like a simple atom count or chi connectivity of order zero also bond-based descriptors and the chi connectivity of order one. Therefore, Labute's descriptor can't be used in the script.

Polarizability

The same MOE file, q_mref.svl, also deals with polarizability offering apol, the sum of atom polarizabilities, and bpol, the difference of bonded atom polarizabilites. The polarizabilities taken for these descriptors are static and extracted from the CRC Handbook, defined within one constant and summed up to get the apol-descriptor value. As a result, we can assign a value to each atom, which inherently is equal to the numbers in the CRC Handbook. In contrast, the bpol values can not be used, as they are the summed up bond values, where the value of one bond is defined as the absolute difference in polarizability of the two atoms taking part.

Apol and Hydrogens. The apol descriptor calculation in MOE always includes hydrogens, even if they are not provided by e.g. taking into account implicit hydrogens using `Add_H`. For the original calculation, that isn't a problem as all values are aggregated to one molecule property. In our script, an exact reproduction of the MOE behaviour would return an unpredictable number of values, for which no fixed-length vector could be prepared before, if not all implicit hydrogens were handed over. Two solutions meet the viewer's eyes: On the one hand, one could always add implicit hydrogens to provided structures for all descriptors. Although this solution seems simple, problems arise when thinking of descriptors returning different results in dependence of provided hydrogens (e.g. charge-descriptors, see section 3.3.1), thus making these incompatible to original MOE results. On the other hand, we can proceed as done for many other descriptors by just taking hydrides instead of atoms, thus ignoring the individual hydrogens even if they are provided separately. We decide to implement the second variant.

Aqueous Solubility

Similar to logP and MR, the aqueous solubility can only be estimated, with most methods belonging to either fragment, descriptor or an experiment related approach. Consequently, all have disadvantages of missing fragments problem, heavy dependence on estimated descriptors like logP or the need for experimental data. Hou et al. therefore constructed an atom contribution model based on the same idea as logP estimation (see section 3.3.2). The inherent atom classification system in this case includes 76 atom types and two correction factors for hydrophobic carbons and the square of molecular weight. The model gives an r^2 of 0.96 when built on a training set containing 878 molecules [42].

Calculation in MOE is done slightly differently, the numbers of atom types are straightened out leaving just 54 different values for heavy atoms and eleven for hydrogens plus only one correction factor accounting for hydrophobic carbons. The fewer number of descriptors leads to a slightly worse but still sufficient r^2 of around 0.90 when trained on 1340 small molecules [37, q_logs.svl].

As the atom classes as well as the correction factor are atom based, we can utilize the descriptor. It is calculated in the function `Calc_LOGS`. The constant atom type definitions and assigned parameters are not globally available in MOE and thus have to be copied into the script. The linear model consists of three parts, one accounting for heavy atom parameters, one for attached hydrogens and one for the correction factor. The atom calculations are done elementwise and added in the last step, so again we can copy them. The correction factor reducing logS by -0.31345 per aliphatic hydrophobic carbon is realized by three masks leaving only the atoms meeting the criteria, one mask for each of the conditions. In listing 3.3, line 1–5, three variables code for them, `numbermask` including carbon atoms, `distancemask` excluding all carbons being one, two or three variable (“∼”) bonds away from a non-carbon, non-hydrogen atom “[#X]”, and `atomclassmask` only including aliphatic carbons (uppercase) bonded to four atoms (“CX4”) or to three atoms with the atom taking part in a π-bonding (non-single bond, “i”), but being in a ring sized 0 (“q0”) [34, SMARTS Functions]. Finally an `andE` step takes only these atoms for which all masks are true (line 6 listing 3.3). As the SMARTS searches now have to be executed twice for all heavy atoms and not only for carbons, performance is expected to be a little bit worse than in the original MOE version, nonetheless, calculation time needed is by far less than the time required by the MLCS calculation.

```
1 local numbermask = aAtomicNumber atoms == 6;
2 local distancemask = zero atoms;
3 local atomclassmask = zero atoms;
```

```
4 (distancemask|numbermask) = not sm_Indexof [(atoms|numbermask), ['*~[#X
    ]', '*~*~[#X]', '*~*~*~[#X]']];
5 (atomclassmask|numbermask) = sm_Indexof [(atoms|numbermask), ['[CX4]',
    '[CX3iq0]' ]];
6  local hyd = -0.31345* andE[distancemask,atomclassmask];
```

Listing 3.3: Building an atomwise correction factor for logS

The MOE original descriptor function denies hydrogen's autonomy and only takes into account hydrides. We follow that example by combining hydrogen values with the attached heavy atom. Additionally, we ignore the constant term of the equation, as distributing them on all atoms would lead to incomparability between molecules of different size. When comparing the sum of contributions displayed on our molecules with the original value of the equation, this discrepancy can be noticed.

3.3.3. Binned vdW Surface Area Descriptors

Binned vdW Surface Area Descriptors (referred to as VSA) are among the most complex descriptors the script is able to break down to the atom level. The principle idea is classifying the molecule's atoms according to their properties of any type (e.g. lipophilicity) and sum up the atom surface areas of each resulting bin. MOE's 32 integrated binned VSA descriptors were introduced by Labute [43]. He uses lipophilicity (SlogP_VSA-descriptors) to capture hydrophobic and hydrophilic effects, molecular refraction (SMR_VSA-descriptors) for polarizability and charge (PEOE_VSA-descriptors) in order to define electrostatic interactions as classifying properties. They are calculated in q_charge.svl (PEOE_VSA) and q_slogp.svl (SMR_VSA and SlogP_VSA) [37, 43].

Atom Surface Area

The atom surface area is defined as "the amount of surface area of that atom not contained in any other atom of the molecule" [43]. In this context, the surface area of an atom equals that of a sphere having the atom in the center and the atom's van-der-Waals radius as a radius. The sum of all atom surface areas gives the surface area of the molecule.

If the sum of both radii A_r and B_r are bigger than the distance d_{AB} between the atoms, an interfering between two van-der-Waals-spheres A and B occurs. The area VSA_A can then be

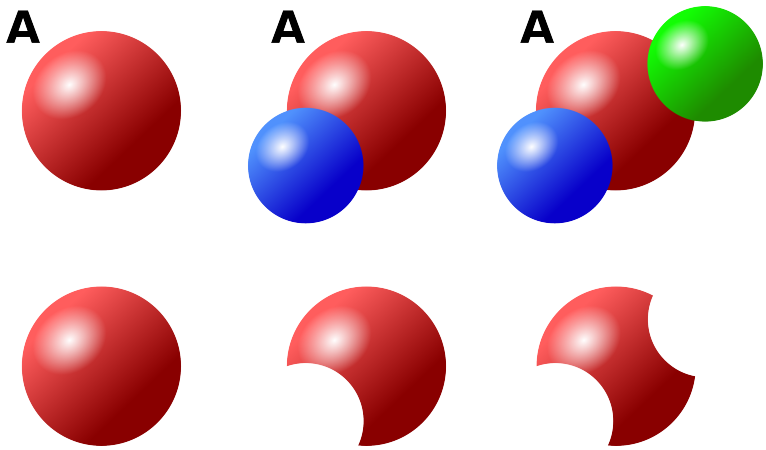


Figure 3.2.: Surface area of atom A dependent on neighbours [43].

calculated as follows (figure 3.2):

$$VSA_A = \begin{cases} 4A_r^2\pi - \frac{A_r\pi}{d}\left({B_r}^2 - (A_r - d)^2\right) & \text{if } |A_r - B_r| < d < A_r + B_r \\ 4A_r^2\pi & \text{otherwise} \end{cases}$$

If more than one neighbouring atom is interfering with the surface area of the atom specified, the atom's area is reduced by the sum of these atoms. As a complication, it might happen that portions of the surface area are part of more than two spheres resulting in an inappropriate sum; however, the algorithm for calculating the VSA neglects that problem for simplicity. Additionally, Labute ignores implications caused by atoms not bonded directly to the atom in question [43].

When compared to a 3D-descriptors [43], the VSA descriptors simplify in points of

- multiple subtractions of portions of the area belonging to more than two atoms,
- ignoring effects of atoms not directly bonded to the atom in question,
- generalizing the bond length which is derived from MMFF94, the Merck Molecular Force Field (for details on MMFF94 refer to [44]), and only dependent on both atom types and bond order,
- estimating the van-der-Waals radii of the atoms using MMFF94,
- founding the 3D-surface area solely on 2D-information and thus losing conformation information and
- using a simple sum-up method to get the whole molecule's property.

Nevertheless, Labute gets an R^2 of 0.97 when comparing to a 3D-based van-der-Waals-surface calculation using MOE 1999.05 and therefore concludes the method being suitable [43].

Forming of descriptors

The bins needed for Labute's VSA descriptors are then assigned a certain value range of a property `X` with the range referring to one single atom. The descriptor value of one bin `X_VSA` is the sum of all individual VSA values of all atoms having the `X` property value within the bin's range. Labute suggests an atom-based logP and MR by Wildman and Crippen [40] as properties, resulting in 10 SlogP_VSAx and 8 SMR_VSAx descriptors respectively, as well as Gasteiger's partial charge PEOE giving 14 charge descriptors PEOE_VSAx.

Recalculation based on atoms

As Labute's VSA descriptors are inherently consisting of subdescriptors containing only parts of the molecule, also only one atom, we conclude that a split-up into the single atoms is valid in terms of descriptor definition. In principle we modify a subdescriptor's range so that each subdescriptor contains one atom only. The value of this fictive subdescriptor is the VSA value of this single atom. The validity of VSA calculation has been shown by Labute [43]. Getting the values out of the descriptor calculation is also a quite simple process and performed using the functions **`Calc_SMR_VSA`**, **`Calc_SLOGP_VSA`** (both assigning values to hydrides and ignoring hydrogens in the following VSA calculation) and **`Calc_PEOE_VSA`** (accounting for hydrogen atoms). The undocumented difference concerning the treatment of hydrogen atoms leads to unequal VSA sums over the molecule when comparing PEOE and SlogP or SMR. We reproduce this reference implementation behaviour in the same way in our script.

The property calculation can be done atomwise by using MOE's global functions **`aSlogP`** `atoms` and **`aSMRef`** `atoms`, or, if PEOE properties are calculated, by **`PartialChargeMOL[`** `molecule,method`**`]`**. The vdW surface area of an atom is available atomwise via the function **`aIdealVSA`** `atoms`. The atoms have to be split up according to their values and their VSA values have to be assigned to the corresponding descriptor using a similar method as in the original MOE SVL file q_slogp.svl (see listing 3.4).

```
1 (v | descr == 'SlogP_VSAx') = nest mput[zero atoms,(logp > -0.40 and
      logp <= -0.20),(vsa | logp > -0.40 and logp <= -0.20 )];
```

Listing 3.4: Extracting VSA values in dependence of logP.

Again, variable `v` contains all descriptor values and `descr` holds all descriptors which have to be calculated (listing 3.4). The MOE internal function **`mput`** selects from the newly created array **`zero`** `atoms` (first function argument) these atoms having the property in the specific range specified (second function argument). The variable `vsa` saves all VSA values of the molecule's atoms, again only these falling in the specified range are selected (third function argument). Then, **`mput`** changes the selected values of the first argument with the corresponding values of the third argument giving a vector containing one value per atom (zero or the VSA value, if the atom's property is within the specified range). As in q_slogp.svl, one line is necessary per (sub-)descriptor. PEOE_VSA and SMR_VSA are processed analogously.

The Hydrogen Problem

When comparing the aggregated VSA values for PEOE with SMR or SlogP descriptors, a difference caused by different treatment of hydrogens will be noticed. In principle, hydrogens are accounted for in all functions, but SMR and SlogP values are aggregated to hydrides. As a result, hydrogens aren't accounted for in the VSA determining function **`aIdealVSA`**, which always returns only the VSA of the atoms provided and doesn't add implicit hydrogens. In SMR and SlogP calculation, the VSA values of hydrogens are therefore ignored, as well as the overlapping area of hydrogens with the corresponding heavy atom, thus giving a slightly larger surface area values for the heavy atoms. On the opposite, PEOE calculates VSA values for hydrogens, too. Generally spoken, PEOE VSA descriptor values therefore tend to be bigger as the molecule is imagined to be bigger with hydrogens attached.

3.3.4. Molecular Connectivity Chi Indices and Kappa Shape Indices

Hall and Kier offer an interesting set of descriptors widely used for different applications like solubilities, boiling points or thermochemical properties as well as QSAR studies [36, p. 367 f.]. Due to the generally molecule-based approach, only a few are suitable for backprojection.

Molecular Connectivity and χ-indices

By Kier and Hall, the molecular connectivity method is referred to as the "quantitation of molecular structure based on the topological and electronic character of the atoms in the molecule" [36]. So the descriptor is in principle atom based and usable.

Molecular Connectivity. χ-indices depend on the molecular connectivity approach, thus the atom's identity in a hydrogen-suppressed graph which also includes the electronic character of atoms. Subsequently, an atom is characterized through the so-called δ-values consisting of the "simple" and the valence delta value, δ^v. Kier and Hall calculate the simple value as the number of neighbour atoms in the molecule's skeleton hinting at the number of bonds being equal to the number of σ-orbital-assigned electrons as an electronic interpretation. δ_i as the δ-value of the i^{th} atom can therefore also be expressed as

$$\delta_i = \sigma_i - h_i$$

σ gives the number of electrons in σ-orbitals and h the number of bonds to hydrogens. Although including an electronic interpretation, the δ-value lacks a specific encoding for the atom's identity, which is possible when referring to the ${\delta^v}_i$-value counting all valence electrons.

$${\delta^v}_i = ({Z^v}_i - h_i)/(Z - Z^v - 1)$$

Z refers to the electronic number and Z^v to the number of valence electrons of the i^{th} atom, thus specifying each atom differently in terms of core and valence electrons. Together, the δ-values allow for characterization of an atom's electronic state [36, p. 373 ff.].

χ-indices. The idea of creating χ-indices dependent on molecular connectivity is then based onto creating subgraphs of the graph representation of the molecule, where the order of the index refers to the size of the subgraphs, more precisely to the number of bonds. A single χ-index is then a weighted count of subgraphs having the same type. As a result, it can be deducted clearly that the simplest χ-index of order zero is related to fragments with zero bonds, thus atoms. This enables the calculation of a value for each atom as the number of subgraphs is "simply the number of skeletal atoms or vertexes" [36, p. 376]. Hall and Kier define the indices as being the reciprocal square root of each delta value:

$$c_i = {\delta_i}^{-\frac{1}{2}} \text{ and } {c^v}_i = {\delta_i}^{-\frac{1}{2}}$$

Subsequently, the resulting χ-index would be the sum over the molecule, that step is skipped while recalculating the index. Altogether, the order zero indices are only influenced by the neighbour atoms, giving little information about the structure of the molecule.

χ-indices of order one represent a bonding. They are defined as the product of the individual δ-values of the corresponding bond atoms, but as they can't be broken down to the contribution of an atom, our programme can't recalculate them. Nevertheless, an extended version of the

script could assign the values of the χ-indices of order one to the bonds of the molecule.

Indices of higher order are not suitable for tracing back to the individual atom, as they are "quite dependent upon the structure of the molecular skeleton" and are, in contrast to those of order zero and one, already exhibiting "both additive and constitutive character" [36, p. 378 f.], thus not encoding solely for the single atom but also carrying structural information.

Molecular shape and κ-Shape indices

Apart from various supposed geometric models, it is generally accepted that molecular shape information is also included in the chemical graph leading to various indices (e.g. Wiener), which, to some extent, are intended for encoding shape information. Nevertheless, the first indices having the objective of encoding relative shape derived from molecular graphs are Kier and Halls κ-shape-indices, relying on the fundamental assumption that shape is a result of bonds and atoms. They focus on number of atoms A, one-bond-fragments ${}^1P$ and further n-bond-fragments ${}^nP$ consisting of n bonds and assume that a combination of the number of existing fragments leads to a representation of the molecule's shape. The values of the shape indices give the proportion on the fragment numbers ${}^nP_m$ available from the current molecule m in comparison to the maximum fragment number possible with the given number of atoms A. ${}^nP_{max}$ is therefore derived from a molecule which graph representation is equal to a complete graph in the case of ${}^1P$, a star graph in ${}^2P$ or a twin-star graph for ${}^3P$, while the minimum fragment number ${}^nP_{min}$ is always counted in a corresponding linear graph. Both ${}^nP_{max}$ and ${}^nP_{min}$ can be easily calculated in dependence on the number of atoms A [36, p. 394 ff.].

The first κ-index takes only ${}^1P$, thus bondings, into account and is calculated

$$ {}^1\kappa = \frac{2 *^1 P_{max} *^1 P_{min}}{({}^1P_m)^2} $$

The other ${}^nP$-indices are calculated similarly [36, p. 396ff.].

As a result of the algorithm giving back a number for the whole molecule and of the fact that shape is always dependent on the whole structure and not on the individual atom, as the κ-indices show by always taking the sum of fragments, a recalculation based on atoms is not possible. A marginal value for removing/adding a certain atom having a bonds to heavy atoms could be defined with respect to the calculation, which might help in a programme visualizing the impact of molecule modifications on descriptors on the fly.

In MOE, the χ-indices of order zero and one are implemented in the file q_btop.svl (chi0, chi0_C, chi1 chi1_C; the valence versions with a v as suffix). The _C-variants only include

C-atoms in the final sum-up, but the original connectivity determination is not influenced and takes into account all heavy atoms as the normal variant. Additionally, the first three κ-shape indices (Kier1, Kier2, Kier3; the modified having an A as suffix) and KierFlex, a combination of Kier1 and 2 relative to the number of atoms, are calculated in q_kappa.svl.

The descriptors which can be backprojected (chi0, chi_C, chi0v, chi0v_C) are calculated in the function Calc_BTOP. After getting the values using identical procedure as in q_btop.svl, the χ-values are taken for the individual atoms and assigned to variable v.

3.3.5. Electrotopological State Indices

The electrotopological state indices, introduced in 1990 by Hall and Kier, combine electronic as well as topological features of the molecule, an approach the authors refer to as a "unified attribution model" [45]. In contrast to many topological indices dealing with the whole molecule, the electrotopological state indices are dependent on the intrinsic electronic state of each atom, taking into account the electronic influence of neighbour atoms. Therefore, the intrinsic state I of an atom is calculated

$$I = \left(\left(\frac{2}{N} \right)^2 * \delta^v + 1 \right) / \delta$$

N is the principal quantum number of the atom's valence shell, this term allows to represent different electronegativities for different quantum levels relative to the second level. δ^v refers to the number of valence atoms, 1 is used as a correction term to distinguish between different hydride states. δ characterizes the position of the atom in the skeleton. As a result, I corresponds to electronegativity, giving large values to atoms with high electronegativity, especially to those with few skeletal bonds where δ tends to be 1 [45].

The E-state for the specific atom i then sums up as

$$S_i = I_i + \Delta I_i$$

Apart from the intrinsic topological state, pertubation from the neighbour atoms with respect to their distances is of importance. ΔI_i stands for these influences, I_j giving the pertubation term and r_{ij} the distance between atoms i and all other atoms j as the graph distance plus one.

$$\Delta I_i = \sum \frac{I_i - I_j}{r_{ij}^2} \text{ , for all atoms } j \neq i$$

Although the E-state can be used as a descriptor "as is" [45], Hall and Kier additionally provide

an atom type scheme [46] which classifies the atoms according to element, valence state, number of bonded hydrogens and identity of bonded atoms. Therefore, three values are necessary: element type described in the number Z, valence state $(\delta + \delta^v)$ and a binary marker of aromaticity, correctly distinguishing between 69 of 79 groups derived from the mentioned criteria. The last 10 are classified by bonded atom analysis and the difference in δ-values, $(\delta^v - \delta)$. On the basis of these 79 groups, on the one hand Hall and Kier suppose a summative atom-type descriptor consisting of the added E-state-values of all atoms in one class, on the other hand a basic count-descriptor just counting the number of atoms being part of a specific class [46].

MOE doesn't integrate the E-state indices for atom types in the basic installation, but provides the additional file q_estate.svl in the sample directory, which gives atom-type dependent E-state descriptors for E-state sums matching one class (kS_) and atoms count (kC_). For MOE version 2007.09 and below this calculation scheme lacks one correct atom ordering thus giving wrong results:

```
1 local Etype = atom_type atoms;
2 local [idx,mask] = sam Etype;
3 Etype = Etype | mask;
```

Listing 3.5: The E-state bug in MOE <2008.10

The function `atom_type`, which is also provided in the file, assigns the right Hall and Kier atom types to the provided atoms `atoms`, which are then processed by `sam` standing for "sort and mask". On the one hand, `sam` returns the indices of the sorted input, on the other hand a sorted mask giving a 1 for all unique elements [34, MOE Function Index — sam]. Line 3 then assigns the sorted mask to the unsorted Etype resulting in wrong behaviour. The solution is simple — the sorting has to be imposed on `Etype` using the already determined index `idx` in line 3:

```
3 Etype = Etype[idx] | mask;
```

MOE 2008.10 delivers a correct version of the file [5, $MOEDIR/sample/q_estate.svl].

As the E-state values are atom-based, we can easily recalculate them. Despite that fact, a definition of the atom types is necessary. They are copied from q_estate.svl [37]. MOE implements only 56 out of the 79 defined types, skipping atom sequences containing atoms (Pb, Ge, Se, As, Sn) obviously thought of not being relevant in drug design, a limitation we also have for compatibility reasons. The assignment of a given set of atoms to the classes is done by the functions `atom_type` and `assign_types`, both are also copied from the provided MOE file. Again, a call of the integrated functions is not possible, as they are declared private. The atomwise E-state values themselves can be obtained by calling `Estate`, which is defined in q_estate.svl

and globally available. In our script, `etype` contains first a list of atom type definitions existent within the current set of atoms, which we get equal to the MOE implementation using a **`sam []`** function. Then, we create the `ctype` and the final `etype` vector holding the fully qualified MOE abbreviations for the E-state descriptors, which are composed of the prefixes for counts and values and the E-state specific atom class representations, both being created with an **`apt tok_cat`** applied on all available E-state classes. For both counts and values we then iterate over all descriptors part of the QSAR-equation (for the value part, see line 2 in listing 3.6). Within the loop, at first the descriptor names for the atom types contained in `types` are created, then we change the zero values of the specific descriptor element for descriptor `k` in `v` with the values `estates` of all atoms of the class `k` (line 5, 3.6).

```
1 //VALUES
2 interesting_descr = descr | indexof [descr,etype];
3 for k in interesting_descr loop
4    local etypeFull = apt tok_cat['kS_',types];
5    (v | descr==k) = nest put [cat (v | descr == k),cat indicesof[k,
        etypeFull],(estates|(etypeFull==k))];
6 endloop
```

Listing 3.6: Creating the E-state value vectors. Counts are processed similarly.

3.3.6. Topological Polar Surface Area Descriptor

MOE offers one descriptor for the polar surface area of a molecule. As the surface area tends to be a 3D-dependent property, traditional approaches include generating a 3D-structure raising the problems of different conformations, followed by calculating the overall surface and reducing to the polar part after identifying polar atoms [47] — a time consuming process which could not be broken down to the atom level. Ertl et al. overcome those limitations by introducing a fragment based method, TPSA (Topological Polar Surface Area), which relies on a table connecting polar groups with a certain polar surface area contribution. The property value for the whole molecule equals the sum of all polar groups. The numbers in the table were obtained from a fitting procedure to the 3D-polar surface area of 34810 molecules from the World Drug Index [48] containing at least one polar atom and having a molecular weight between 100 and 800. The procedure resulted in an R^2 of 0.982. As the method itself is based on adding atom fragments, taking atom values and not the sum for the whole molecule is inherently valid. Breaking this descriptor down is simple and done in the function **`Calc_TPSA`** which uses the built-in MOE function **`aErtlTPSA`** already returning one value per atom.

3.3.7. Adjacency and Distance Matrix Descriptors

BCUT and GCUT descriptors

Adjacency matrix descriptors, in that context originally developed by Burden, are in principle based on producing a molecular identification number out of the lowest eigenvalues of a connectivity matrix. After all hydrogens are deleted and the remaining heavy atoms numbered, the symmetric matrix is established. The diagonal elements $B_{i,i}$ equal the atomic numbers, for each $B_{i,j}$ a value of 0.1 times the bond order is taken if i and j are connected, with an additional plus of 0.01 if the bond is terminal. If the corresponding atoms are not bonded, the value is 0.001 [49].

Pearlman and Smith improved the concept to BCUT (for **B**urden, **C**AS to which the concept was applied to by Rusinko and Lipkus and Pearlman's **U**niversity of **T**exas) by enlarging it. More than one matrix represents a compound, the values in the diagonal are changed to more biochemical relevant than atomic number and the off-diagonal elements are modified [50].

MOE implements BCUT-like-descriptors offering diagonal elements substituted by atom-wise PEOE- (see section 3.3.1), SMR- (3.3.2) and SLogP-values (3.3.2) and allows returning the smallest, the $\frac{1}{3}$ and $\frac{2}{3}$ percentile and the largest eigenvalues coded as BCUT_*_0 to BCUT_*_3 where * stands for different diagonal elements. GCUT-descriptors are calculated similarly, but instead of an adjacency matrix, a distance matrix is used.

At any case these descriptors return one value representative for the whole molecule as this value is one eigenvalue of an adjacency or distrance matrix, thus any recalculation is not possible.

Diverse Adjacency or Distance Matrix Descriptors

The MOE descriptors radius and diameter both are derived from the distance matrix and defined as the lower and upper bound of eccentricity $E(x)$ for a molecule as set of points $\mathbb{P}$, where $x \in \mathbb{P}$ and eccentricity being the maximum distance between x and $y \in \mathbb{P} \setminus \{x\}$. Subsequently, they are specific for the whole molecule and can't be determined atomwise. The same goes for petitjean and petitjeanSC, as they are calculated as the difference between diameter and radius devided by diameter or radius, respectively [51]. The value for wienerPath also origins in the distance matrix, it is defined as a half of the sum of all distance matrix entries, closely resembling wienerPol where only distance matrix entries with value 3 are added and then divided by

2. VDistEq is defined as

$$\text{VDistEq} = \log_2 m - \sum \frac{p_i \log_2 p_i}{m}$$

where m equals the sum of distance matrix entries (thus being twice wienerPath) and p_i is the number of entries in the matrix where $p = i$. For VDistMa, a similar calculation method causes similar problems. As these descriptors only have one value standing for the whole molecule, the whole group of adjacency and distance matrix descriptors can't be used for our purposes (compare [52]).

3.3.8. 2D-Volume and van-der-Waals Surface Area Descriptors

Based on the 2D-van-der-Waals (vdW) surface area, a few descriptors are included in MOE dealing with ligand interactions. In principle, these resemble the binned surface area descriptors (see section 3.3.3) as they again are equal to the 2D-surface area of a subset of a molecule's atoms. The basic vsa_area descriptor equals the van-der-Waals surface area of the whole molecule, it is based on the vdW-surface area calculation.

In contrast to the binned vdW descriptors, atoms for the partial surface area descriptors are not selected due to having a certain property value range, but on generic features like hydrophobicity or hydrogen bond acceptor capability. MOE uses different functions belonging to the **ph4_*** group to categorize the atoms, e.g. **ph4_aHydrophobe** for hydrophobicity or **ph4_aAnion** combined with **ph4_aResonanceAnion** for anion detection. All descriptors of these group can be used. Calculation in the script is done similar to the binned surface area descriptors [37, q_vdw2d.svl].

If the surface A of a sphere, as done for the surface area, can be calculated mathematically, the same goes for the volume V, as no additional parameter but the radius r is required [37, q_vdw2d.svl]:

$$A_{\text{sphere}} = 4r^2\pi \text{ compared to } V_{\text{sphere}} = \frac{4}{3}r^3\pi$$

As a result, it is possible to have a van-der-Waals volume descriptor being based on summing up the individual atoms' values [37, q_vdw2d.svl]. This allows us to implement these descriptors in our script in the same way as other vdW-descriptors. A special case is the density descriptor. In principle the mass and the volume of each individual atom is known and could therefore be calculated, nevertheless one has to bear in mind that the backtracing can't be done by simply adding the atom density values.

3.3.9. Fundamental Atom Property Descriptors

A large amount of descriptors in MOE simply represents atom-based fundamental properties like number of nitrogen atoms or number of hydrogen donors and acceptors. As they are just binary for the atom itself and then summed up, it is possible to assign a value to each atom. Inherently, this is either a 1 if the property is available for a certain atom or a 0 if not. As an example, a_nO codes for the number of oxygens, thus each oxygen has a value of 1 for the descriptor, all other elements have 0. In the special case of a_count, which is simply counting the number of atoms, we calculate hydride values, so heavy atoms include the numbers of attached hydrogens. The same goes for a_nH, the number of hydrogens, which values are also assigned to the corresponding heavy atoms.

A bigger problem are properties which are not based on one atom, but on a bond. As described above (see section 3.2.2), problems with weighting can occur. Therefore, we decided to skip these descriptors, examples would be number of double bonds (b_double) or number of rotatable bonds (b_rotN).

Although also belonging to these easy understandable type of descriptors, some represent a fraction or a maximum or minimum, which can't be allocated to one single atom. They describe a molecule's property, as an example the descriptor reactive, which is 1 if one or more reactive groups are present in the molecule. In principle, one could divide the descriptor value by the number of reactive group or you could assign a 1 to all reactive groups, but this procedure would cause problems when comparing different molecules on the one hand and distort the original meaning of the descriptor on the other hand. Thus, these descriptors are not included in the script as listed in appendix A.

Other descriptors just giving basic information are Weight (available via `el_DefaultMass aElement atom`) for the atom weights of hydrides, or FCharge representing the ionization state of the atom, as included in the database (`db_mol(4)(MOL_ATOM_ION)`) where `db_mol` is a molecule in MDB format. The descriptor chiral gives the number of chiral centers determined by the built-in function `aRSWaterChiral`. Each chiral atom is assigned a one. The chiral_u property excludes those having a chiral constraint, it equals the return value of the function `aForceRS`.

Furthermore, non-usable descriptors include rings giving the number of rings in the whole molecule and thus having a ring as fragment and nmol returning the numbers of molecules, which is of no relevance as we are dealing with atoms.

Depending on the necessity to include hydrogens, calculation of simple descriptors is dealt with using the function `Calc_BTOP` or, if hydrogens can be excluded, `Calc_BTOP_woH`.

3.3.10. Drug- and Leadlike Descriptors

In an attempt to address ADME issues, some descriptor sets are available which try to give reasons for druglikelyness by means of simple properties. They tend to have cut-off rules figured out by redesign of drug databases (compare e.g. [53]). MOE offers two sets, Lipinski's Rule-of-5 and Oprea's leadlike descriptors.

To implement Lipinski's Rule-of-5, MOE needs descriptors for hydrogen donor and acceptor count as well as lipophilicity and molecular weight. Lipophilicity and molecular weight are implemented, logP is calculated using Labute's SlogP estimation (see section 3.3.2) in that case, leaving two easy-to-understand properties, Lipinski donor and acceptor count (lip_don and lip_acc). The donor count equals those nitrogen and oxygen atoms which are connected with at least one hydrogen each, while the acceptor count is just the amount of atoms of these types [37, btop.svl]. Both are thus binary descriptors summing up properties of atoms. Currently, we have not implemented lip_don and lip_acc, as the other druglike descriptors can't be used.

The descriptor really expressing Lipinski's Rule-of-5 is lip_violation, as it describes the number of violations occurring when applying the four Lipinski rules on the molecule. The number is a property of the whole molecule, which can't be broken down. The lip_druglike descriptor is finally based on lip_violation, being one if lip_violation is smaller than two and zero otherwise, therefore also a property being specific for the whole molecule.

The Oprea leadlike set includes, beside the properties used for Lipinski, three additional properties: opr_nring counting the number of rings having three to eight members according to the smallest set of smallest rings algorithm, which gives a one for each ring, but no smaller fragments. Further opr_nrot codes for the number of rotatable bonds, which is implemented in the local function `OpreaNrot`. It equals the number of freely rotatable, nonterminal bonds excluding sulfonamides, esters and other groups plus the number of rotatable bonds in non-aromatic rings with six or more bonds. As a result, opr_nrot gives a one for each rotatable bond, but not for atoms. The same goes for the third property, number of rigid bonds opr_brigid, which also includes terminal single bonds and is defined as the difference between all bonds and opr_nrot [53].

The factors above are then combined to opr_violation descriptor, counting the number of violations for logP, molecular weight, Lipinski donor and acceptors as well as number of rings and rotatable bonds, thus again being a property of the whole molecule. opr_druglike is built similarly to lip_druglike, but takes into account opr_violation.

3.4. Concepts of Visualization

Beside the recalculation of descriptor values on a per-atom basis, this work primarily aims at giving a good overview over the distribution of values within the molecule and thus providing a clear visualization of the individual atoms' contributions to the whole PLS-QSAR model. We decide to implement a graphics mode enabling us to project back the values to the molecule, clearly indicating the contributions by means of colours or graphical objects. In principle, only few scientific literature is available on how to visualize molecule properties on the molecule itself, but many chemical software suites offer various solutions.

The amount of contribution can be shown for instance by applying a colour scheme encoding various ranges, additionally other methods can help to improve understanding. In PLS-QSAR, where datasets optimally consist of only one or a few scaffolds and subsequently have many atoms "in common", contributions of these common atoms might not be as important as these of atoms being unique to a molecule — thus we allow to exclude the multiple largest common substructure (MLCS) from the contribution calculation giving an emphasis on a molecule's unique features. The same goes for hydrogens — many descriptors are calculated for hydrides (see section 3.3), thus hydrogens are omitted when using the standard display mode.

The additional optional multiple view mode displaying one to four molecules in parallel finally alleviates molecule-to-molecule comparisons. All in all, the techniques implemented should give a good graphical overview of contributions.

3.4.1. Colour Scheme and Object Visualization

When using a graphical representation and assigning an object a special colour, this colour can contain information. Colouring atoms and bonds, for example, has already been used by various chemical software packages, e.g. SYBYL HQSAR [11].

Colour	from	to
red	$\ll$	-0.2415
red_orange	-0.2415	-0.1449
orange	-0.1449	-0.0966
white	-0.0966	0.1018
yellow	0.1018	0.1526
green_blue	0.1526	0.2544
green	0.2544	$\gg$

Table 3.1.: Colour coding in dependence on the value in SYBYL HQSAR [11].

SYBYL offers the already introduced HQSAR package including fragment contribution prediction (see section 2.1, [11]). SYBYL therefore breaks down the contributions to colours using seven discrete codes (see table 3.1) ranging from red for bad contributions via white for neutral to green for good ones. Between white and green, yellow and green-blue are set as intermediate values, thus giving an in principle positive contribution a yellow sign. Commonly, the colour yellow typically is interpreted as “Watch out!”, “Be careful!”. Furthermore, yellow has a negative connotation in both folk tradition and Goethe’s theory of colours [15]. Additionally, green-blue is seen as the “coldest” of all colours, thus also not suited for signalling positive contributions. On the other hand, the MIBALS add-on package for MOE ([54], see also section 2.2) assigns only two colours, grey and red. The amount of contribution is then visualized by object size of so-called wirespheres, a wireframe model of spheres placed over the specific atom [34, Graphics Object Functions]. Red is used for negative, grey for positive contributions — a not very intuitive colouring either, especially as grey is already used for colouring carbon atoms of the molecule.

Implementing a Good Colour Scheme

Our script therefore provides an easy interpretable, intuitive colouring scheme ranging from red showing negative contributions to white, which stands for neutral atoms, and then to green for good ones. We skip intermediate colours and don’t classify the values into histograms like SYBYL, but use a floating scheme assigning pastel-coloured for intermediate and an intensive red and green for the extreme values respectively. MOE supports 24 bit RGB colours, thus has a 256 steps scale (1 byte) for red (R), green (G) and blue (B), the resulting colour is computed by adding the three components. It is represented by one integer value with the high byte coding for red [34, Color Widget]:

$$\text{code} = \text{red} * 256^2 + \text{green} * 256 + \text{blue}$$

As the focus is set on one molecule, the colour scheme is scaled over red and green for the molecule’s values to have either an extreme green or red colour. The colour is calculated using the local function `calculateColors`, shown in listing 3.7.

```
1 local function calculateColors[atoms,values]
2 local maximum = max (abs values);
3 local multiplier = 200/maximum;
4 local colors = zero atoms;
5
6 local posvalues = values | values > 0;
```

```
7  local posgreenvalues = mput[zero posvalues,igen(length posvalues),255];
8  local posredvalues = ceil (200-multiplier*posvalues);
9  (colors | values > 0) = pow[16,4]*posredvalues + pow[16,2]*
       posgreenvalues + posredvalues;
10
11 //similar calculation for negative values skipped
12
13 (colors | values == 0) =  pow[16,4]*255 + pow[16,2]*255 + 255;
14
15 return colors;
16 endfunction
```

Listing 3.7: Determining object colours in dependence on values

In listing 3.7, `atoms` is an array of atoms for which the colours have to be calculated in dependence on `values`. To scale, the maximum value is determined in lines 2 and 3. `multiplier` is one for the maximum value, 0.5 for the half. For all (in this case) positive values, the red, green and blue values have to be determined. To get the pastel scale, we set the maximum to green (255, line 7). The red and blue amount is then dependent on the value, thus determining a grade between green (RGB code 0,255,0) and (near) white (200,255,200). In line 9 we subsequently calculate the integer code for MOE as shown above. Zero values are set to white (line 13), then the colour array is returned.

When using the parallel view mode, the colours are also only scaled over the molecule itself, thus an intensive green of one molecule can't be automatically compared with an intensive green of other molecules displayed simultaneously.

Activity by Colours

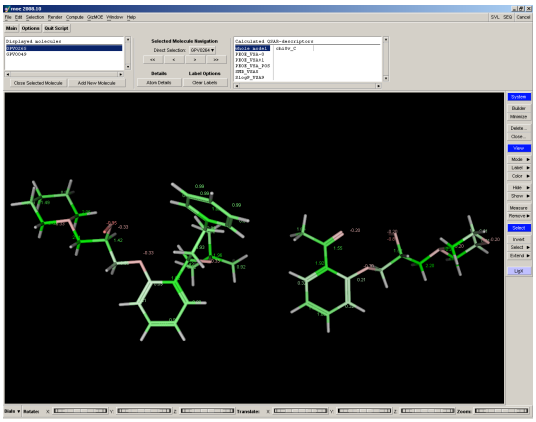
As shown in the SYBYL HQSAR package, one of the easiest ways to signal contribution is to simply colour the atom according to its value. MOE therefore offers the `aSetRGB [atoms, color]` function which defines a freely assignable RGB colour for atoms. At that point, the RGB colour of an atom is defined, but not shown, as the package offers various colour modes for atoms. By default, atoms are displayed with the colour assigned to their element type, `'element'`. To change to RGB, the command `aSetColorBy [atoms,'a:rgb']` has to be stated for the corresponding atoms. The colouring itself is done using the function `colorAtoms[]`.

"Colouring the atom" in terms of MOE's graphical capabilities is misleading, as the atom not only includes the core point, but also half of all bondings the atom takes part in. On the one

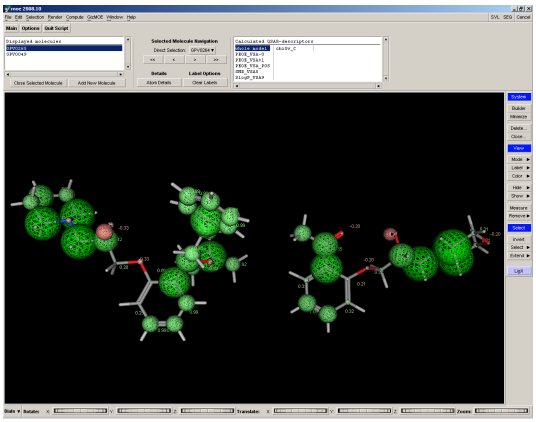
hand, this inclusion makes the assignment clearly visible as the colour is not only reduced to one point. On the other hand, it is impossible to assign a different colour to the bond itself.

An inherent disadvantage of this method is that the size of the value is only characterized by the colour, thus making it sometimes difficult to clearly distinguish between only slightly different properties. As the original colour is overwritten, the colour coding of the elements in the molecule is hidden, which might lead to problems, although typically chemists know their molecules used for QSAR very well. Additionally, the atoms' elements can be indicated by using the element text feature of MOE.

The main advantage of this method is the simple calculation, which is fast for a large number of atoms, as there are no additional graphical objects to draw. Furthermore, the graphical representation is self-explanatory and easy to understand. The textual representation of the value is clearly visible, as no additional objects can hide them. An example of the colour visualization is shown in figure 3.3a.



(a) Activity by colour.



(b) Wirespheres mark the activity. Scale factor 0.5.

Figure 3.3.: Figures show the two possible visualization options of contributions to activity. In both cases, GPV0265 and GPV0049 (see appendix B) are displayed. The MLCS is not shown.

Wirespheres as Visual Marker for Activity

In contrast to a simple colouring which is using pre-existing objects like bonds or atoms, the MIBALS approach ([54], see also section 2.2) draws new objects in order to visualize its model fragment scores, so-called graphic object spheres. These are wireframe models of spheres, which radii are dependent on the score of each fragment. The only source of estimating the value of the score is therefore the object size, as the colouring is just red or grey. The function `wiresphereAtoms[]` implements the wiresphere creation procedure in our script.

The graphical object has to be defined using `GCreate`, then a vertex object is established as a wiresphere model, for which an own MOE graphical function `G_WireSphere` is provided. It accepts four parameters, apart from colour and position a radius for the sphere and a quality parameter, which can be defined from "poor" (0) to "excessive" (4) or even higher [34, Graphics Object Functions]. We decide for 2 as a compromise. If speed problems occur with wirespheres, setting this parameter to 1 might be a good approach. The creation of the sphere itself is then done by `GVertex` combining the generic graphics object with the parameters set up by `G_Wiresphere`.

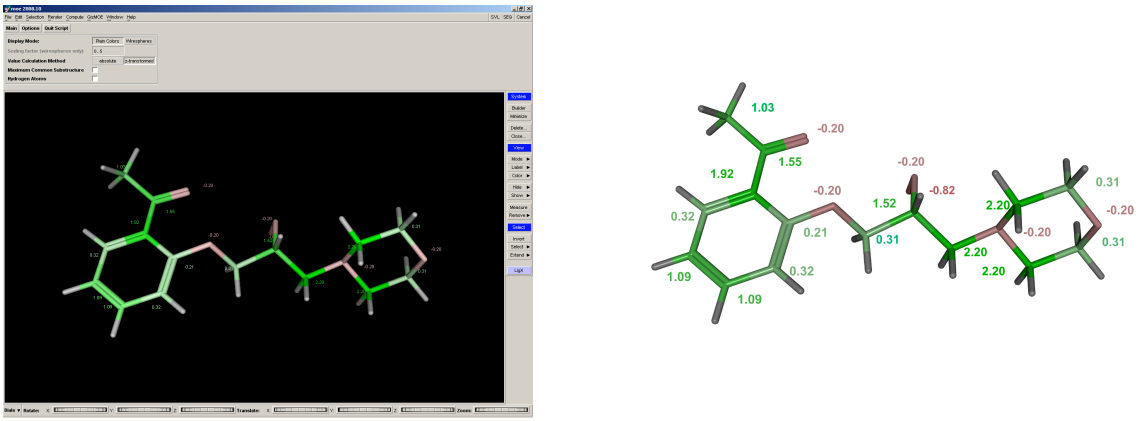
As the spheres' radii are based on the absolute values of contribution, their sizes can differ extremely. Very large spheres can even be bigger than the whole molecule, thus the middle-points of the spheres, which are identical with the atoms, are difficult to determine, resulting in a problematic assignment task, difficult interpretation and a potentially hidden textual value representation. Very small spheres make comparisons impossible. To overcome that problem, we implement a scaling feature `wirespherefactor`. This factor is used to resize all visible spheres by entering numeric values, the standard being two. It is accessible via the graphical user interface on "Options" tab and only active if "Display Mode" is switched to "Wirespheres" (see section 3.5.1).

Apart from slow creation when displaying more than one molecule (see section 3.4.5), the wiresphere model offers many advantages. As the original molecule can be seen, determination of elements or other MOE implemented features is possible. Additionally, the spheres double-code the values: On the one hand, the colour equals that of the bond colouring, on the other hand the size of the spheres is adapted to the value. For an example, refer to figure 3.3b.

3.4.2. Textual Representation of Values

Apart from visualizing values by means of colours (see section 3.4.1) or size (see section 3.4.1), a textual representation is necessary in order to get exact values for comparison. In principle, these values have to be situated near the atoms they belong to, but they should not be hidden by the atom's bonds or neighbouring atoms, which are themselves dependent on molecule structure. As a result, placing the texts in threedimensional space is not as easy as one would expect. Andrew Henry, Chemical Computing Group, offers a special script dealing with labelling chemical structures in the MOE Window via SVL-Exchange [13, label_atoms.svl], which labels in accordance to the bond count of the atoms. In most cases, the labels are set properly, so we decide to adapt the provided function as `label[]` taking three arguments: the atom to label, the colour the label should have and the value of the label.

For single atoms without bonds, the label is just set 0.5 grid units off the atom. For one-bond atoms, the position of the bonded atom is taken into account putting the label on the opposite of the bonding. Atoms having two bonds require a **rot3d_vCross** operation returning the cross product of the input vectors equal to the bond directions specified, this vector being orthogonal to both input vectors. As the label gets transformed along this vector, it is set orthogonal to both bonds and therefore visible. Higher bonded atoms are dealt with differently. At first, the longest bond determined via Euclidean distance is eliminated. Afterwards, the position is calculated as an average of the remaining atoms' positions bonded to the atom to label. As a result, the label is opposite to the longest bond and in the middle of the remaining ones, thus having only a small chance of interfering with drawn bonds [13, label_atoms.svl].



(a) Original MOE screenshot.

(b) POV Ray™-converted [55] image.

Figure 3.4.: Although most of the textual representations are visible, MOE lacks scaling abilities and therefore texts might be too small for export. The raytracer POV Ray™[55] allows for scaling after saving the window in *.pov format, but often requires shifting of texts.

Finally, the graphical object can be created using **GCreate** and later defined as **GText** with four arguments, graphical object key, colour, position and value. To raise clarity, the colour used for the label equals that of the corresponding atom providing an additional visual connection. Unfortunately, graphical text style and size can't be influenced in MOE, and it actually might be thought of being too small. This might especially be a problem when the user wants to export the MOE screen for further usage. A workaround for this problem might be the conversion into a POV-Ray™-file, where the scaling of the text can be reset manually and very easily. At any case, the text gets formatted to two decimals to avoid lenghty numbering using the capabilities of **twrite**, expecting as first element in its value vector an input format (`n:.2f`), which defines the value to be a number with two decimals (figure 3.4, listing 3.8).

```
1 value = twrite['{n:.2f}',value];
```

Listing 3.8: Text formatting for labels.

3.4.3. Multiple Largest Common Subgraph

A molecule visualized with all partial values gives a quite complex picture of the molecules' atoms contributing to activity. For each atom, a certain colour (see section 3.4.1) or even a 3D-object (see section 3.4.1) is used accompanied by a textual representation of the value (see section 3.4.2) for up to four simultaneously presented molecules. As a result, the user might encounter problems extracting important information. Often a dataset used for QSAR studies contains molecules having just one or only a few common scaffolds, which typically give the same contribution for the equation. This is caused by many descriptors only taking into account the very proximate neighbourship of the fragments under investigation, which within a common scaffold is identical for all molecules. Of course, there might also occur differences in contribution for some descriptors, but to get a good in detail view our programme needs the possibility to exclude the nearly equally contributing common parts of the dataset from visualization.

To extract the common set of atoms, we have to determine the largest substructure common to all molecules of the database. Bayada et al. refer to this structure as the "multiple largest common subgraph" (MLCS) [56]. Unfortunately, there is only limited literature available dealing with dataset subgraphs, as examples Bayada et al. [56, 57], Takahashi et al. [58] or Varkony et al. [59].

The Chemical Graph

As the term "subgraph" already implies, the MLCS procedures we deal with are based on a simple, undirected graph representation of chemical molecules. These graphs are a reproduction of 2D-molecules and consist of vertices and nodes, where each node or vertex represents an atom and each edge a bonding between two atoms [56], both having labels which code for atom and bond properties. As a result, it is generally possible to apply graph matching or isomorphism techniques on them, which allow for the calculation of common subgraphs and, if the graph is reconverted into the molecule, of common substructures. A graph is a common subgraph G_C of graphs G_1 and G_2 if subgraph isomorphisms from G_C to both G_1 and G_2 exist. Subgraph isomorphism occurs if graph isomorphism between G_C and G_1 as well as G_C and G_2 can be established, thus both having a bijective mapping $f : V_x \rightarrow V_C$ for $x = 1, 2$ in terms of vertex and edge labels for $\forall V \in V_x$ [60]. So isomorphism between two graphs is defined as "a one-to-one correspondence between their vertices" and includes that "an edge only exists between two vertices in one graph if an edge exists between the two corresponding vertices in the other graph" [61].

Among the set of common graphs, there exists at least one maximum common subgraph having the most nodes of all common subgraphs [60]. In chemical sense, a maximum common subgraph is the graph representation of the maximum common substructure (MCS) between two molecules. Raymond and Willett list different use cases for MCS, among them protein-ligand docking, searches within databases or interpretation of molecular spectra [61].

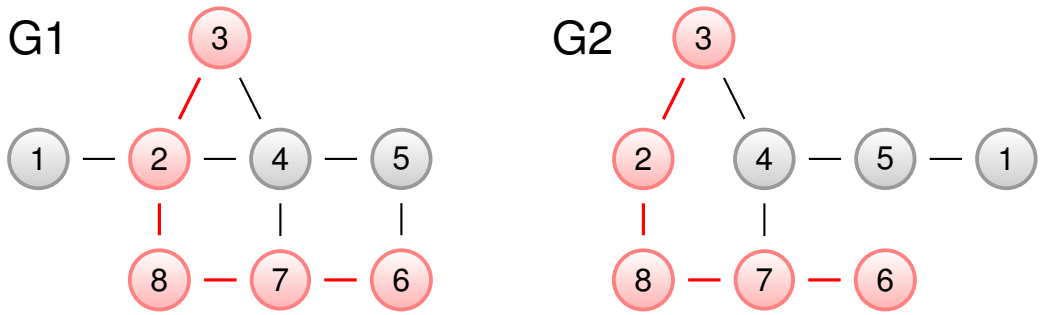
Graph Similarity

For chemical graphs, additional labels assigned to vertices and/or edges have a direct influence on the output of an MLCS calculation, as they define which bonds or atoms correspond and thus which atoms and bonds are equivalent. This equivalence can be expressed either by the term "commonality" [58] or "similarity" [59]. We use both terms similarly. At any case, "commonality should be strictly defined when common substructures are searched for" [58]. Varkony et al. list various simple criteria a chemist might apply to express similarity of atoms at two levels. On the one hand, similarity based on descriptions of substructures dealing with characteristics of bonds and atoms themselves, taking into account cyclic position of atoms, atom element and bond multiplicity. On the other hand, they define similarities of attachment of substructures within structures. Here Varkony et al. distinguish between "internal" atoms only having bonds to other atoms in the substructure and "external" atoms, which are also connected with non-MCS atoms (compare also the MCIS and MCES variants of maximum common substructure, see section 3.4.3). They define the simple method, which is not differentiating between internal and external atoms, and various other where external atoms have to fulfil criteria in both graphs to be taken as equivalent, like equal number of external bonds, equal hybridization states, or both, as well as the "pattern" similarity, where different structures have the same origin [59].

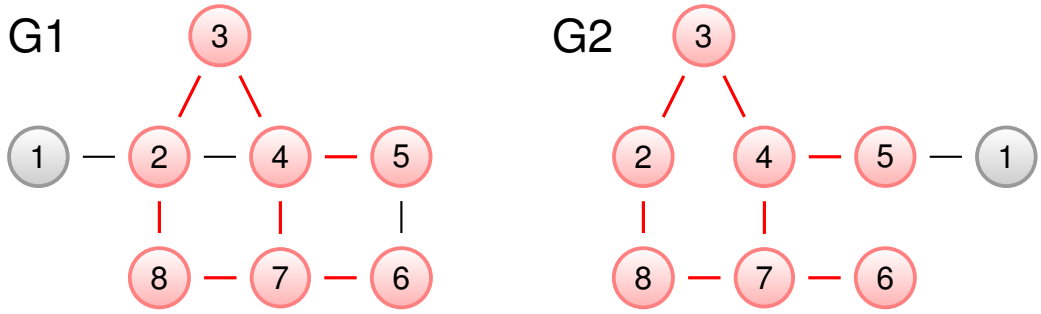
Another approach to chemical graph commonality distinguishes between five levels of commonality. Takahashi et al. define these levels in dependence on which labels of bonds and atoms are taken into account. Level 1 only notices the structure without any labels, levels 2 and 3 consider either bond or atom labels, level 4 is a bond and atom-weighted representation and level 5 includes also hydrogen atoms at level 4, which are otherwise ignored [58].

Types of Maximum Common Substructures

Raymond and Willett distinguish between a maximum common induced subgraph (MCIS) and a maximum common edge subgraph (MCES) [61]. MCIS is a representation of the largest number of common vertices including a correspondence of all edges existent between vertices part of the subgraph. Two atoms are then part of the substructure, if all bonds between these atoms



(a) Maximum Common Induced Substructure (MCIS) of graphs G1 and G2.



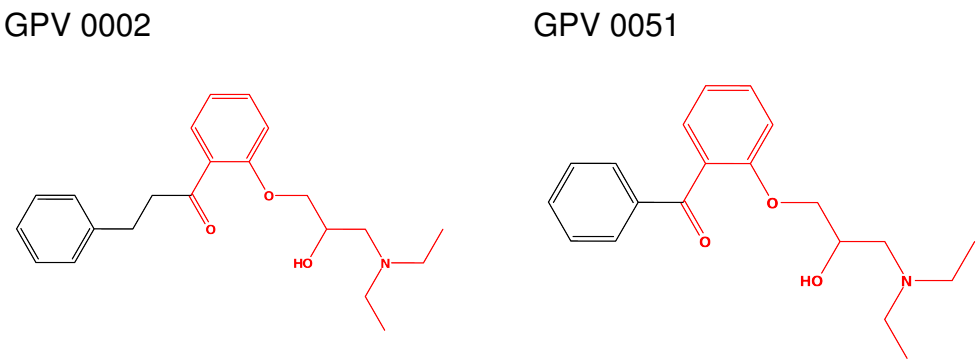
(b) Maximum Common Edge Substructure (MCES) of graphs G1 and G2.

Figure 3.5.: Comparing MCIS (a) and MCES (b) as in [61]. Edges and nodes shown in red are common to both structures.

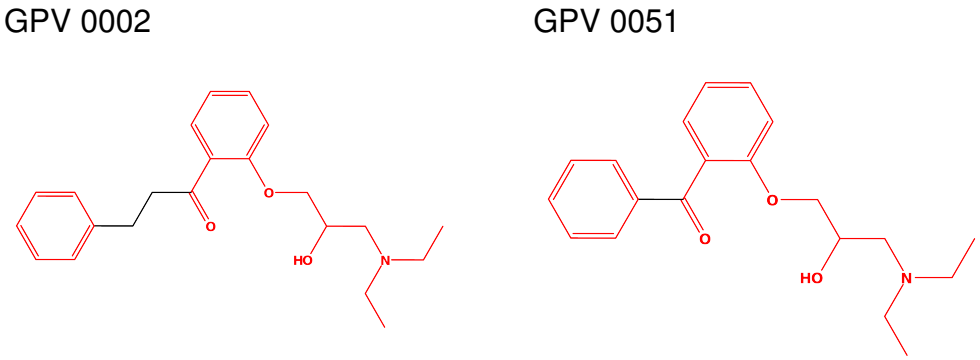
are in common. The subgraph is then isomorphic to the induced subgraphs of both graphs, which was denoted by Levi [62]. MCES are simpler to understand and a weaker definition of a subgraph [63], as they are simply the largest sum of common edges of both graphs under consideration [61]. Figure 3.5 illustrates the difference between MCIS and MCES. McGregor and Willett are of the opinion that MCES closer resembles the needs of a chemist than does MCIS, an opinion which is also shared by Raymond and Willett (compare [63, 61]). Other authors use an induced subgraph algorithm, e.g. Cao et al. [64].

Considering our application, it can be said that an MCIS represents the common structure also taking into account bondings within the structure and thus parts of the neighbourship, while MCES has a focus on the bond, thus being closer to our focus on the atom. We decide to use MCES in our algorithm, as many of the descriptors we use are only dependent on the atom itself, and all have fragment size one (compare section 3.2). We we will refer to the MCES simply as “MCS”.

A second classification, which to some extent is derived from MCIS and MCES, can be done by differentiating between connected and disconnected subgraphs. A connected maximum subgraph only includes those vertices with at least one path between each of them, whilst a disconnected subgraph might consist of more than one fragment [61]. In principle, it is difficult to



(a) Connected maximum common substructure.



(b) Disonnected maximum common substructure.

Figure 3.6.: Comparing connected (a) and disconnected (b) MCES as in [61]

say whether a disconnected or a connected MCS represents similarity between two molecules better, and it is mostly dependent on its purpose. McGregor and Willett note that for the detection of bond changes in a reaction, a "maximal overlapping structure" originally used by Vleduts in 1963 is needed, which also implies that there might exist more than one MCS fragment. Despite that opinion, the results of a disconnected MCS algorithm in many cases are a bad measurement of similarity between two structures. To reduce this problem, often the number and the size of the fragments are limited. Additionally, this procedure drastically reduces the overall size of the problem [64]. There are also solutions where just connected structures are taken into account, e.g. by Koch, who enumerates all maximum structures and thus needs this limitation to reduce calculation time [65], as all possibilities have to be iterated. Figure 3.6, b) displays one problem of traditional disconnected MCS, as it shows that the position of the isolated phenyl group is not taken into account when determining the MCS, it is only important that both molecules have an isolated phenyl group. That happens because the bond between the atoms of the two fragments, which would necessarily be taken into account when calculating an MCIS can be left out during MCES calculation. Therefore, the linker between two fragments

is not important (compare [63]).

Most general algorithms published deal with MCIS, causing a necessity to have a transformation between MCES and MCIS. Its existence was proven by Whitney in 1932 stating that an "edge isomorphism between two graphs, G_1 and G_2, induces a vertex isomorphism provided that a ΔY exchange does not occur" [61]. A ΔY exchange happens when two different root graphs G_1 and G_2 have the same line graph $L(G_{1,2})$ (see figure 3.7). A line graph $L(G)$ is a graph where all edges of G are vertices in $L(G)$ and vice versa. Two vertices in $L(G)$ are adjacent if and only if the corresponding edges in G are incident, thus have one vertex in common. If an ΔY-exchange can be excluded, an MCES can be extracted by getting the MCIS of the corresponding line graphs. To prove that a ΔY-exchange doesn't happen the degrees of the sorted MCS vertices projected onto the original molecule have to be identical, which is not the case after a ΔY-exchange (see figure 3.7, [66]). The degree of a vertex or an atom is defined as the number of its directly connected neighbours.

It has to be taken into account that an MCES algorithm returns the maximum substructure in terms of edges, thus the structure consisting of the most bonds in common. This substructure does not necessarily equal the substructure having the most atoms in common.

MLCS Approaches

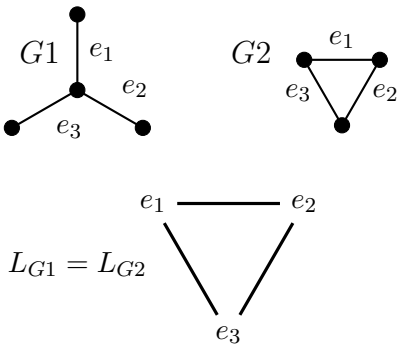


Figure 3.7.: ΔY exchange leading to isomorphic linegraph [61] of graphs $G1$ and $G2$.

Basically, two totally different approaches to obtain the MLCS are described in literature. On the one hand, the maximum common substructure (MCS) of two molecules can be extended to an MLCS by numerous pairwise comparisons of graphs [56, 57], on the other hand some algorithms start with the smallest available graph, check for its existence over all molecules and extend it stepwise while constantly checking if the graph is still present in all molecules until no further extension can be found [58]. According to Bayada et al., the first option is by far more efficient and time-saving, so we will concentrate on that type of algorithm.

In principle, $\frac{n*(n-1)}{2}$ pairwise MCS steps have to be performed to obtain an MLCS for n graphs [56]. Fortunately, this number can be reduced drastically in various ways: At first, Bayada et al. suggest using a backtracking algorithm, which is a general solution finding algorithm systematically extending partial solutions stepwise to a valid solution. If a problem occurs, some

steps are taken back and modified [15]. Bayada's backtracking procedure works as follows: In a set of n graphs and after calculating all common structures of two input molecules $G1$ and $G2$ bigger than a certain threshold, a common structure set $R21 \ldots R2x$ is obtained. Then the first result $R21$ is compared to $G3$, having structures $R31$ to $R3x$ in common, which again exceed the threshold. The algorithm continues with $R31$ versus $G4$, until comparing $R(n-1)1$ with Gn resulting in the first MLCS proposals $Rn1$ to Rnx, which are saved. Then the backtracking step is applied: The algorithm tracks back one step to the level of $R(n-1)$ and compares $R(n-1)2$ with Gn, updating the MLCS if the result is bigger than the current. Afterwards, the algorithm again tracks back. This is continued until $R(n-1)x$ has been finished, followed by stepping up one more level and comparing $R(n-2)2$ with $G(n-1)$, again going deeper. This backtracking is continued until the last common structure $R2x$ of the first two molecules $G1$ and $G2$ has been tracked down, delivering one MLCS of the data set [56, 57].

Figure 3.8 illustrates a simple example with $n = 5$. After processing the root `getCS[]`-function which retrieves all common structures from $G1$ and $G2$ above threshold $limit$, the algorithm continues on the next deeper level with getting the structures of the largest result $R21$ and $G3$ (step 1). This is continued until all structures n have been matched (after step 3). Then a set of MLCS is obtained from `getMCS[]`, which returns all maximum common substructures, and set as $MLCS$. The algorithm tracks back one level (step 4) and compares the next bigger result $R42$ with $G5$. If the results are bigger than the current MLCS, $MLCS$ is reset. In step 6, we again track back, but no bigger structure $R43$ is found. So the backtracking is continued (steps 7,8) until one larger substructure still not matched is detected: $R22$. Step 9 compares this again with the next unmatched graph, $G3$. The same procedure is processed, until step 14 returns to level 1, where no substructures are left. $MLCS$ contains the MLCS of the dataset.

As Bayada et al. don't want to enumerate all solutions but find one MLCS, a branch is only entered if the potential MLCS under investigation, which is at most the size of the current common structure, is larger than the MLCS available at this point. If a possible result R is smaller than or, as we just want to have one MLCS (and the algorithm returns only one), the same in size as the MLCS already given, the whole branch of the tree can be omitted. Resultingly, many branches are left out reducing the number of comparisons, but we therefore need an MCS procedure capable of setting a minimum MCS size [56, 57].

Principles of a Two-Molecules MCS Algorithm

As the MLCS algorithm we choose to implement is based on a two-molecule comparison, we first deal with the traditional MCS. An MCS algorithm for the general case is known to be an NP-complete problem, thus having no solution within polynomial time complexity. It is

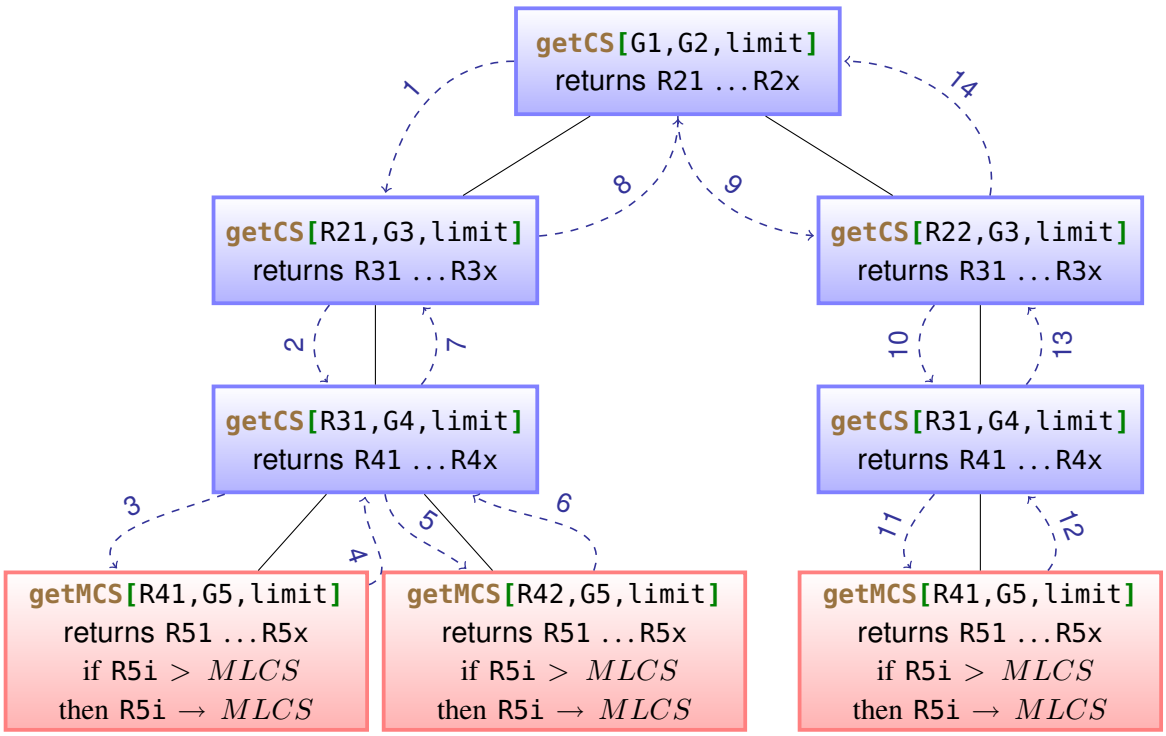


Figure 3.8.: Schematic workflow of the Bayada algorithm. Adapted from [56]. Further explanations and details can be found within the text.

possible to get all subgraphs having length k of two graphs with a and b atoms, but that gives an extremely large number [62]:

$$\frac{a! * b!}{(a-k)! * (b-k)!k!}$$

As an example, for a matching of two structures having 22 and 25 atoms, which should identify subgraphs with length 3, the maximum number of atom comparisons is 21.3 million. Thus, sufficient algorithms have been developed. In general, there are exact and approximate algorithms. All known exact algorithms are NP-complete problems, but "have proven to be very efficient when applied to many of the graphs of chemical interest" [61]. Raymond et al. classify exact algorithms into maximum clique-based (for instance RASCAL [66]), backtracking algorithms, as an example Cao et al. [64], and into dynamic programming algorithms as by Akutsu [61]. Maximum clique-based algorithms are based on finding the maximum clique of a compatibility or association graph G, a subset of vertices of G, where each vertex of the clique has to be connected directly with each other member of the clique. The Akutsu dynamic programming procedure, which is used for sequential decision problems, works with caching subproblem solutions. Backtracking algorithms work by searching all vertex correspondences of the graphs to compare, which are organized in a search tree. Although backtracking algorithms are said

"to have been surpassed in subsequent years by the significant developments in the efficiency of clique detection" [61], a newly adapted algorithm has been published in 2008 [64] with also having a good, if not excellent performance [64]. Due to the lack of a standardized test set as well as different in-detail approaches, different testing environments and algorithm implementations, speed comparisons between algorithms and datasets are difficult [64, 66, 57].

MCS Algorithms Founding Our Implementation

Especially for maximum clique-based and backtracking algorithms, lots of variants and literature is available. One of the more recent ones is RASCAL by Raymond, Gardiner and Willett, which shows good performance when compared to other maximum clique-based and simple edge projection technique algorithms [66], as far as a comparison is valid and possible. A second one published by Wood in 1997 [67] is said to be inferior to RASCAL in terms of speed [66] when considering labelled graphs. Nevertheless, the concept of Wood only implements the maximum clique finding step and is therefore a more general procedure, as it does not rely on labelled graphs and is not specifically designed for comparing chemical structures. Therefore a valid comparison is in fact difficult. We base our implementation of an two-molecule MCS upon the general concept of the Wood maximum clique finding procedure, as it is easy to understand, and add some chemical graph-features of the more specific RASCAL algorithm.

Both algorithms are so-called branch-and-bound algorithms, which are in principle enumerating algorithms for solving optimization problems. These problems are characterized by having at least one objective function which should reach a maximum or minimum, thus the optimal feasible solution. In principle, they are based upon two operations, branching and bounding, which are applied recursively on the dataset and its subsets [68]. "Recursive" means that a function is calling itself while it is active.

- branch: divide collections of sets into subsets
- bound: set bounds to the value of the objective function concerning the current subsets

To solve the problem quickly, the algorithm tries to enumerate as little feasible solutions as possible whilst eliminating the majority by administering bounds, which give a lower and an upper estimation for the solution taking into account the current subset. The bounds are obtained using suitable estimation methods, the branching takes place within a tree structure of nodes and links. Each link connects a node to its successor nodes, which form a subset [68].

Maximum clique problem and modular product. A maximum clique represents the biggest set of vertices which form together a clique, thus are connected to each other. In our case, we look for the common clique in the compatibility graph, also denoted as association graph. On the contrary, an independent set is a set of vertices, in which there exists no vertex $v_i \in S$ which is adjacent to any $v_j \in S$ [61]. The MCIS, which can then be transformed to the MCES, is equal to the projection of the vertices part of the maximum clique of the compatibility graph back on to the original graphs, as proved on different occasions. Subsequently, the problem of finding a maximum common substructure is reduced to the NP-complete problem of finding the maximum clique in a compatibility graph [66]. This graph, also denoted as modular product graph, $G_1 \diamond G_2$ [61, 66] is defined on the vertex set $V(G_1) \times V(G_2)$. One point in the vertex set (matrix) is denoted as (u_i, v_j), with $u \in G_1$ and $v \in G_2$. Two vertices in the common vertex set (u_i, u_j) and (v_i, v_j) are said to be adjacent, thus connected via an edge and possibly part of one clique, whenever

$$(u_i, u_j) \in E(G_1) \ \vee \ (v_i, v_j) \in E(G_2) \ \vee \ p(u_i, u_j) = p(v_i, v_j)$$

or

$$(u_i, u_j) \notin E(G_1) \ \vee \ (v_i, v_j) \notin E(G_2)$$

where $p(u_i, u_j)$ specifies the properties of the bond taken into account to determine compatibility between (u_i, u_j) and (v_i, v_j), and $E(G_x)$ denotes the edge set of graph G_x [66].

As most of the atoms in organic chemistry are carbon atoms and typically molecular graphs are very sparse, the second term will be applied in most cases, and so the "modular product graph will be dense as well as large" [69], which is critisized [64], as calculation time grows. Additionally, the concept lacks some flexibility (e.g. concerning mismatches or predefined maximum numbers of disconnected fragments) when compared to the backtracking approach [64].

A general maximum clique finding algorithm. After the modular product has been calculated, the algorithm continues with the maximum clique finding procedure. In very general terms, this process consists of the following steps (compare [66, 67]):

- Step 0 — Initialize: Set variables to initial values, set level to one. Continue.
- Step 1 — Lower Bound: Find any clique to have a lower bound. This is done by setting the minimum similarity index in RASCAL, or guessing a clique in Wood for the first time, later the current maximum clique usually is the lower bound. Continue.

- Step 2 — Upper Bound: Find an upper bound of the current branch by colouring methods, projections bounds or any other procedure. If the size of the maximum possible clique determined by the upper bound is larger than the current, continue with step 3, otherwise go to step 4.
- Step 3 — Branching: Choose one of the vertices left at this level which has best chances to give good results. At this point, Wood simply chooses the vertex with the maximum degree, RASCAL uses partitioning methods. The available set for the next level is then equal to all still available vertices additionally connected to the chosen one. The level is incremented by one. Continue with step 1.
- Step 4 — Backtracking: If the level reached is the original level one, stop — the maximum clique has been found. Otherwise decrement the level as long as the upper bounds of the specified level is smaller than the current available maximum clique.

Important Steps within a Two Molecules MCS

As RASCAL as well as the maximum clique finding algorithm by Wood are branch-and-bound algorithms, bounds need to be defined — lower bounds as an absolute minimum to start investigation, upper bounds as the absolute maximum the current branch is expected to give. Additionally, the algorithms need a way to choose the next vertex. The following section gives an overview of sub-procedures available within the algorithms, which are important in terms of our MCS algorithm implementation.

Lower Bound Finding. The trivial variant of defining a lower bound is just setting it to the length of the current maximum common clique — but both algorithms offer more sophisticated options. RASCAL defines a size derived from the minimum similarity index mentioned above as a lower bound, as it can give an estimation on the smallest common clique to be expected. Wood's algorithm uses an own sub-algorithm which tries to quickly get a valid clique as an estimation for the branch and adapts it temporarily as the maximum clique, if it is bigger than the biggest clique determined so far. The subbranch is defined as $G = (V, E)$ with V being all vertices of the subgraph G and E being its edges. Then is

$S \leftarrow V$
$Q \leftarrow \emptyset$
while $S \neq \emptyset$
$Q \leftarrow Q \cup \{v\}$ where $v \in S$ with maximum degree in G

$S \leftarrow S \cap N_G(v)$
endwhile

$N_G(v)$ is defined as all vertices $\in G$, which are neighboured to v. As an interpretation, the algorithm always chooses that vertex with the most neighbours, adds it to its substructure set Q and adapts the choice of still available neighbours [67].

Upper Bound Finding. As well as lower bounds, upper bounds are needed to estimate the maximum clique size a branch can offer. In principle, Wood offers two different colouring approaches depending on vertex colouring. Vertex colouring (k-colouring) means assigning one colour to each vertex and assuring that all neighbouring vertices are coloured differently. In principle, the colouring problem itself is again an NP-complete problem, as far as the minimum necessary number of colours has to be determined [15]. In contrast to the NP-complete problem of MCS determination, approximate algorithms are the only choice in this case. The set of vertices having the same colour i form a colourclass (C_1, C_2, C_3,...,C_i). Subsequently, each colourclass consists of an independent set of vertices, where no vertex is connected with another from the same set. Further, the number of colourclasses gives the maximum clique size, as one colourclass can only include one member of a clique. The lowest number of colours meeting the criteria is then the chromatic number χ. Various algorithms are available returning a number slightly above χ, Wood mentions COLOR and DSATUR as well as a fractional method built upon these, FCP. For COLOR, U are the uncoloured vertices and C_k a colourclass set, which is to be determined [67].

$S \leftarrow U$
$C_k \leftarrow \emptyset$
while $S \neq \emptyset$
assign colour k to $v \in S$ with maximum degree in G
$C_k \leftarrow C_k \cup \{v\}$ $S \leftarrow (S \backslash v) \cap N_G(v)$
endwhile

DSATUR follows a slightly different approach by always assigning the lowest possible colour to the vertex available with the highest saturation degree, thus the highest number of different coloured neighbour vertices. COLOR gives a higher number of colourclasses, but uses up to one magnitude less CPU time for very sparse or dense graphs [67].

Raymond et al. on the other hand qualify colouring approaches as only suited for $125 \leq |V(G)| \leq 175$, as the difference between χ and the maximum clique tends to rise as the number of nodes increases. Therefore, they suggest using the so-called projection bounds as an upper

bound for their algorithm, which despite its simple design returns often tighter bounds than the colouring approaches [66], but are limited to labelled graphs. Considering the modular product which lists preserved edges, one line equals one edge of G_1 and one column one edge of G_2 thus allowing to assign two labels e_1^i and e_2^j for the i^{th} and j^{th} edges of G_1 and G_2. Now two bounds K_α and K_β are projected back on the factor graphs, giving $\min(K_\alpha, K_\beta)$ as an upper bound, the Bessonov bound [66]. Raymond et al. further refined this technique by introducing vertex and bond labels as identifiers to further refine equality between two edges. They grouped edges with same identifiers i of graph G_1 into classes S_1^i with n being the number of classes and suggest a sharper upper bound K_{wp} as

$$K_{wp} = \sum_{i=1}^{n} \min\left\{\left|S_1^i\right|, \left|S_2^i\right|\right\}$$

As the improved Bessonov bound is based on labelled graphs (figure 3.9), it gives an upper bound solely on the graph representation and does not need the compatibility graph. It is in general suited to give an estimated upper bound when comparing two graphs, thus two molecules. Therefore, we also use the improved Bessonov bound to get an estimation over the maximum MCS to be expected when searching for the two initial molecules in the MLCS algorithm (see section 3.4.3).

New vertex selection. Selecting a new vertex is a very important step within an MCS algorithm as it decides how fast the MCS is found. Ideally, the first n vertices selected in the branch-and-bound section are all n vertices of an MCS, allowing for excluding many branches at a very low level and for a quick traversal through the tree. Algorithms try to guess the importance of a vertex in different ways. Wood suggests a very basic selection method. His algorithm always takes the vertex with the highest degree, thus with the most neighbours, as the next to investigate. Additionally, the vertex must be included in the last colourclass, which represents an independent set (see section 3.4.3). With that method, the number of colourclasses representing the upper bound can be reduced quickly, as the last colourclass tends to have less members than the others [67]. RASCAL implements a more complex variant including a repartitioning after the original creation of independent sets. It tries to re-assign vertices from so-called excess partitions, thus from independent sets with higher indices than the current maximum set, to those below the current estimation of an MCS, as these don't have to be investigated. The next node is then chosen from the vertices left in the excess partitions, having the rational of those nodes being unable to repartition due to their high degree [61].

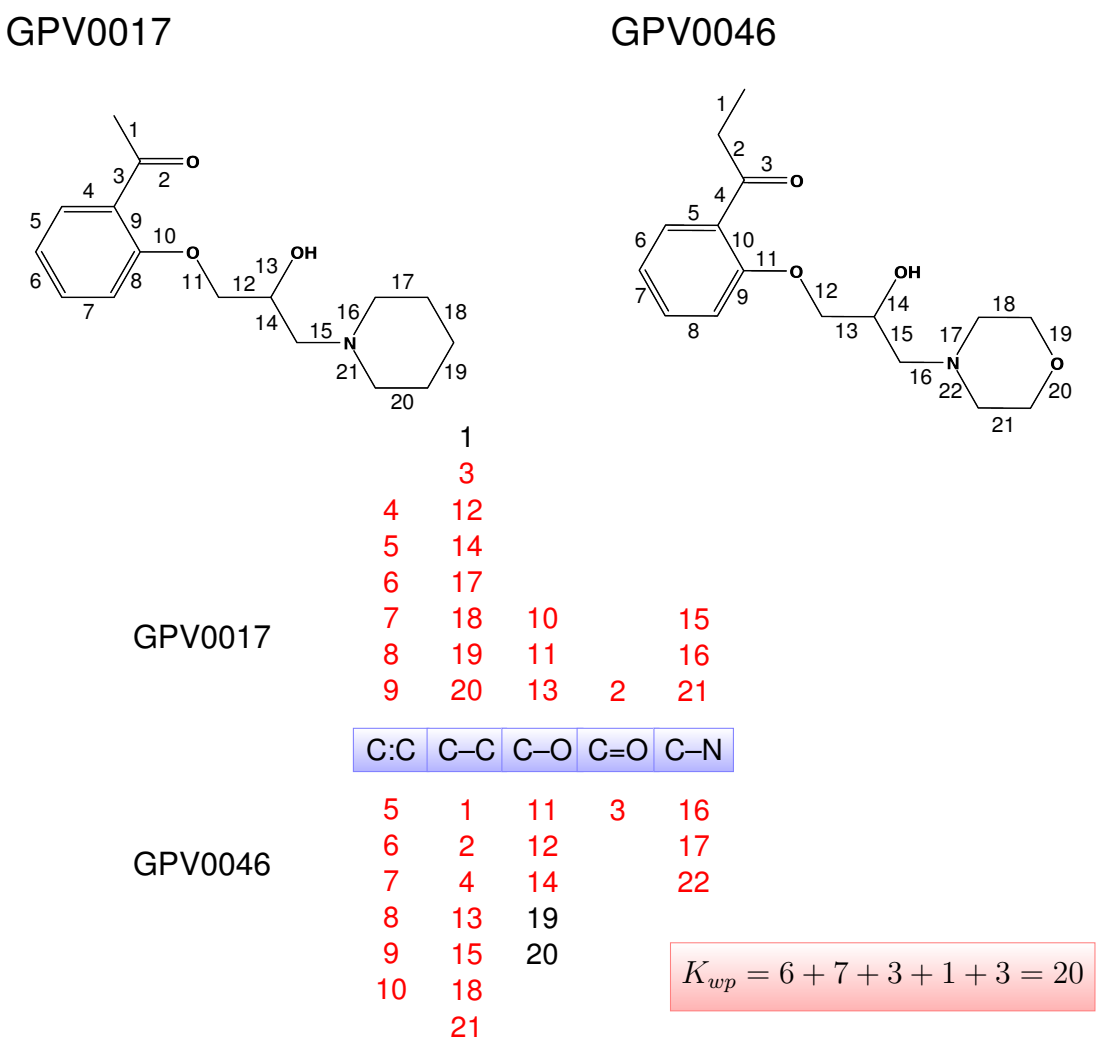


Figure 3.9.: The Raymond labelled projection upper bound derived from the Bessonov bound including GPV0017 and GPV0046 (taken from [66]). The bound's size is 20, which is close to the true MCS of 19.

Implementation — Calculating the Modular Product

Although the definition of the modular product is quite simple (see section 3.4.3), a not-too-slow SVL implementation needs to take advantage of vector operation. The creation of the modular product is done in the `calcModularProduct[keys1,keys2,bonds1,bonds2]` function, available in the file mcs.svl, having two atom arrays of the molecules in question as arguments. Optionally, the bonds between the atoms can be defined, which is especially necessary when dealing with substructures of a molecule (see section 3.4.3).

In the beginning, the bonds have to be extracted using MOE's **`BondGraph`**, which gives a neighbour list representation of the graph, and **`graph_uedges`**, which converts the list representation to a list of bonds already eliminating duplicates. Then the line graph representation has to be

constructed by iterating over all nodes of the line graph, which are edges in the original graph, separately for molecule one and two (see listing 3.9). The edges connected to one line graph node are then built up by determining all other nodes containing one of the current node's two atoms, which result from the original bond converted to the node. To sum up, two nodes of a line graph are connected if the two have one atom in common in the original molecule.

```
local linegraph1 = zero edges1;
for x=1,length edges1,1 loop
 a = app anytrue apt m_join[edges1,edges1(x)(1)];
 b = app anytrue apt m_join[edges1,edges1(x)(2)];
 linegraph1(x) = cat indicesof[1,xorE[a,b]];
endloop
```

Listing 3.9: Creating the line graph.

`edges1` contains an atom number representation of all edges of the original graph, `edges1(x)(1)` therefore gives the first atom of the x^{th} edge. **`apt m_join`** gives a mask with a one for all elements of `edges1`, which are contained in the second argument, the **`apt`** iterates over all pairs of edges in `edges1`. Then **`app anytrue`** reduces the number of arguments in `a` and `b` to the number of edges, indicating all directly connected with a one. The final **`indicesof`** gives back the number of the directly connected edges in the array, **`xorE`**, the exclusive or, eliminates the edge under consideration itself. `linegraph1(x)` is the list of neighbours of the x^{th} element.

`calcModularProduct[]` has three subfunctions, with **`prepareCheckEdges[`**`keys1, keys2`**`]`** simply determining the atom types forming the bonds and their degrees. We define an aromatic bond to have the value 0. **`checkEdgesVertices[`**`bond1,bond2`**`]`** deals with comparison of bonds, returning the equivalence state of two bonds.

After the **`prepareCheckEdges[]`** has built the type lists, the modular product edges1 $\times$ edges2 matrix is determined. We use the **`stretch`** and **`resize`** functions of MOE to avoid time consuming loops when iterating over all elements. Together, both functions generate a vector having all possible combinations of two arrays, thus saving two encapsulated **`for`**-loops (see listing 3.10).

```
x = stretch [x_id edges1, length edges2];
y = resize [x_id edges2,length edges1*length edges2];
```

Listing 3.10: Generating the first iterating vector.

Listing 3.10 thus generates an **`[x,y]`** array containing all nodes of an $[x, y]$-matrix. Then we create a vector holding another vector including all directly connected neighbours for each node of this matrix. In principle, both edges forming the matrix node have to be compatible. The

incompatible edges are therefore eliminated in the beginning by applying a compatibility mask returned by `checkEdgesVertices` on the `[x,y]` array. Then the connection buildup is started. A node is then connected to another node, if the conditions set in section 3.4.3 apply. Therefore, each node left in the $[x, y]$ matrix has to be checked against all other nodes $[x_i, y_j]$ for either x/x_i and y/y_i each representing two edges of the original graph of one molecule having one atom in common, including the compatibility of those two edges (atoms, degree). Alternatively, both x/x_i and y/y_i cannot be part of their respective line graph. Only if one of these two cases apply, one node in the matrix is connected to the other.

```
1 local x2_igen = length edges1 - x;
2 local y2_igen = rep [x_id edges2,length y];
3
4 x2 = apt stretch [(app igen (x2_igen))+x, length edges2];
5 y2 = apt resize [y2_igen, x2_igen*length edges2];
6
7 mask = not (y==y2);
8 x2 = x2||mask;
9 y2 = y2||mask;
```

Listing 3.11: Generating the second iterating vector.

To allow for that check, listing 3.11 again builds up a loop-saving vector pair. In principle, we create a vector in which each element represents one node in the matrix and has a list of x/y-coordinates the node has to be checked against. So we avoid double checks, which would occur if every node is compared with all others, and save calculation time. Thus, the vector contains only the nodes which are beyond the node in question. As a result, we later have to mirror the results to get the full set again. `x2_igen` therefore contains the `igen`-vector for the rows after the certain element `x`, where `x` is holding all available x coordinates of the matrix. Determining the columns to check is easy, as they are just all available for every row. We don't care about the columns left in the current row, as they are eliminated later, allowing for just repeating an all-column containing vector once for each element coded in `length y`. The `apt` command in lines 4 and 5 applies the `stretch` and `resize` commands to each of the first elements in both lines equalling the length of the left matrix nodes (`length y` or `length x`). This vector has already been created for the columns in line 2, it is built up from the left rows scheme via an `app igen` command over rows to cover, corrected by `x` to shift the rows behind the current element. The number of stretches of each element in the vector is then equal to `length edges2`, thus the number of columns. Then the column coordinates are set using the basis vector holding them. It is just enlarged to the size of `length edges2*x2_igen` and equals the number of left rows times columns.

As already mentioned, a check with nodes in the same column or row is unnecessary, as for one dimension the same reference to the bond remains, which would indicate an edge being incident to itself, which is not possible in a chemical graph [61]. In the rows dimension, we eliminate those cases during `x2_igen` creation, for the columns an application of the mask `mask` on both `x2` and `y2` (listing 3.11, lines 7-9) removes the node's own column. At the end of this step, `x2` and `y2` contain arrays of row and column numbers of all nodes compatibility checks have to be performed on.

```
1 local vertexValue =apt checkEdgeType[x,x2,y,y2];
2 vertexValue = cat vertexValue;
3
4 local splitter = app length x2;
5
6 local x_all = cat stretch[x,splitter];
7 local y_all = cat stretch[y,splitter];
```

Listing 3.12: Compatibility check and preparation for the XNOR-gate.

In listing 3.12, `apt checkEdgeType` applies one `[x,y]` pair with the corresponding `[x2,y2]` array pair, which holds all nodes to check, to the check function `checkEdgeType[]` returning connectivity arrays giving a one if and only if the edges of the original graph under consideration both have one atom in common, thus are connected. This `vertexValue` variable is later used for determining the connected MCS.

Then the `[x,y]` pair is enlarged to be vector operation compatible to `[x2,y2]`, with `splitter` holding the length of the individual segments (see listing 3.12). These can easily be obtained by `app length`, thus applying the `length` command on the elements of a vector.

```
1 local sw1, sw2;
2 local mol1_conns = app sort tr [x_all,x2];
3 local mol2_conns = app sort tr [y_all,y2];
4 sw1 = indexof [mol1_conns,linegraph1_edges];
5 sw2 = indexof [mol2_conns,linegraph2_edges];
6
7 local firstterm = andE [sw1,sw2];
8 local vertex_comp = zero firstterm;
9 (vertex_comp|firstterm) = get[linegraph1_edges_labels,sw1|firstterm] ==
      get[linegraph2_edges_labels,sw2|firstterm];
10
11 local msw = vertex_comp or andE not [sw1,sw2];
12
```

```
13 [x_all,y_all,x2,y2,vertexValue] = [x_all,y_all,x2,y2,vertexValue] || [
       msw];
```

Listing 3.13: Building up for the XNOR-gate.

The next step in listing 3.13 is to look whether the possible compatible connections are part of the line graph edges. First, both x and y values are transposed, so that each element of the resulting array holds the x coordinates in `mol1_conns` as well as the y coordinates in `mol2_conns`. With **`app sort`** each of these coordinates is sorted, starting with the lower value. As our connections are not directed, this is allowed and necessary to find the connections within the sorted linegraph, which is done in lines 4 and 5 in listing 3.13. `sw1` and `sw2` are then either the index values, if the edges are part of the linegraph, or otherwise 0 for each pair of `mol1_conns` or `mol2_conns`. To finally conduct the XNOR, we have to determine not only if both connections are part of the line graphs, but also if their labels (the connecting atom between the two bonds under investigation) are the same. First, we assign `firstterm`, the result of a logical **`and`** between `sw1` and `sw2`, and initialize a new vector `vertex_comp` defining label compatibility holding primarily only zeros. The zeros are converted into ones signalling compatibility if and only if the results of looking up the edge labels saved in `linegraphx_edges_labels` are the same for both graphs. To save time, that search is limited to only these edges contained in the linegraph.

We can now apply a sort-of logical XNOR-gate to the dataset (see listing 3.13) to implement the connection condition for the modular product. The first term is equal to `firstterm`, the second includes all items included in neither of the linegraphs, the result `msw` is then applied to our coordinate variables.

Before returning the correct modular product, we have to mirror the result, which up to now only contains connections $A \rightarrow B$ where $A < B$, to get both $A \rightarrow B$ and $B \rightarrow A$ (see listing 3.11 for details). For that reason, we simply append the other part in listing 3.14.

```
1 x_all = cat append[x_all,x2];
2 y_all = cat append[y_all,y2];
3 x2 = cat append[x2,x_all];
4 y2 = cat append[y2,y_all];
5 vertexValue = cat append[vertexValue, vertexValue];
```

Listing 3.14: Mirroring the results.

The modular product is then returned as a one-dimensional vector, where each element represents one matrix node. First, all unique x/y-nodes are extracted from one `x_all`/`y_all` combination using the built-in function **`uniq`**, then a vector is built containing a vector of con-

nected matrix nodes for each of these unique nodes. The variable `modularProduct` is initialized having a zero for each element in `vertices`, thus for each node in the matrix. `x_all` holds all nodes with connections, which are listed as `x2`, `y2`. `edges2` is the number of bonds in molecule 2 equalling the columns in the matrix. As a result, `linevalue+y_all` is exactly an x/y-coordinates linearized position, which is filled with all linearized coordinates it is connected to (listing 3.15, line 3). The edgetype is set in the same way. As a last step, listing 3.15 returns both modular product and the array containing the position-equivalent vertices.

```
1 local linevalue = (x_all-1)*length edges2;
2 local modularProduct = rep[0,length vertices];
3 modularProduct[linevalue+y_all] = ((x2-1))* length edges2 +y2;
4 edgetype[linevalue+y_all] = vertexValue;
5
6 return[modularProduct,vertices];
```

Listing 3.15: Building the modular product.

Implementation — MCS Algorithm

As the MCS function in general is only a visualization helper, we just implement a basic version of obtaining an MLCS based on Bayada's algorithm. We limit the calculation to a connected subgraph (compare section 3.4.3) over the whole dataset as in [58]. For our purposes this seems to be more useful, as small fragments might not be equivalent in terms of contribution for the whole database. Therefore, the MCS algorithm is only a basic version, nonetheless, beside of returning a connected substructure, also able of retrieving disconnected MCS.

We found our algorithm on the clique-based Wood alternative, as it is well documented and for our purposes easy to implement. Additionally, we also integrate some of RASCAL's features especially designed for comparing chemical structures, as an example the projection bounds method for estimating bounds. Like the modular product retrieving script, the MCS script is coded within the file mcs.svl. To meet the various criteria and options that can be set, we offer four globally available functions. All of them call the main function `getStructure[]` after parameter modification and have some parameters in common: `akeys1` and `akeys2` give the atoms of the two molecules the calculation is performed on, the parameter `connected` returns a disconnected structure when set to 0 or a connected MCS otherwise, `minsize` allows for setting a certain minimum expected size.

The typical function is `getMCS[keys1,keys2,options]`, which returns an MCS with the above mentioned parameters. `getMCSBonds[]` further expects two parameters, `bonds1` and `bonds2`,

which allow to specify the bondings between the atoms which have to be taken into account. Normally the bonds are extracted from the molecule structure, but if the given atoms are only a part of the molecule, it cannot be automatically assumed that the partial bond structure is equal to the structure of the whole molecule, as we are dealing with MCES subgraphs. Another function is `getCS[]`, which does not return the MCS, but all common structures larger than or equal `minsize`. Here we also implement a bond-aware function, `getCSBonds[]`.

The branch-and-bound step is encoded in the function `branchbound[options]`. The MCS script and the `branchbound` function heavily depend on `static` defined variables. An SVL static variable's context is, in contrast to `local` variables, one module, thus one *.svl-file, but it is not defined for the whole programme like `global` variables. So they allow for an interchange between functions. Additionally, they are not deleted after the function returns, so we always have to reset them manually if the value of the preceding function call is not important [34, SVL Declarations]. Although our script is capable of finding more than one MCS if there are different possibilities, we don't claim to find all available MCS structures.

The definition of commonality is very important [59]. In our case, we set the level to 4 according to Varkony's proposal, thus taking into account both atom and bond labels. For atom labels, we just take a look at the element type. For bond labels, the degree of the bond is of importance, including aromaticity. Other attributes e.g. cyclic/acyclic state of atoms are ignored. Nonetheless, a modification of equivalence definition within the script is not too complicated and can be done by adapting the function `checkEdgesVertices[]` in the file mcs.svl.

Linearization and general organisation. Typically, branch-and-bound algorithms are denoted as recursive algorithms. This form of implementation is not possible in MOE, as the maximum number of functions called seems to be limited by a stack size of approximately 2^{10} which is easily reached during MCS branch-and-bound algorithms, even if molecules are small. In some cases, a recursive call can be linearized by using a loop, fortunately this is possible for both Wood and RASCAL. The substituted recursions are put into one main loop in `branchbound[]`, which is entered after the beginning initialization of `getStructure[]`, which is the main function doing the work of step 0 in the general scheme (see section 3.4.3). `branchbound[]` covers then all steps left. As the Wood algorithm calls step 2 after the initialization, the loop starts with upper bound calculation using a projection bounds implementation. Then the backtracking step 4 is called which checks if branches are left and tracks back if the current branch is finished, which is linearized using a `while ... loop` construction equal to step 4. After finishing, step 4 at any case calls step 3, so step 3 can be iteratively added after the loop. Step 3 has only one possibility of continuing, too, so step 1 is added last to the big

loop, which afterwards has to continue with the step positioned first, step 2. If the loop is interrupted in step 4, all found MCS structures are returned to `getStructure[]` and the calling global functions.

The variable `h` indicates the current level, `S(h)` the set of vertices `V` which have to be taken into account, thus have not been investigated so far and form a common clique considering all levels `1` to `h-1` at level `h`. `S(h)` is derived from `D(h)(1)` and `D(h)(2)` holding all theoretically available vertices not yet investigated at level `h`. This differentiation is due to the implementation of the connected variant of an MCS, where only these vertices have to be considered which are directly connected. `PBk(h)` gives the current projection bound at level `h`, thus the upper bound. `C(h)` holds an array of coloured `S(h)`, with `length S(h)` being the number of colourclasses at level `h`, thus the length of `S(h)`, which could be taken as an alternative upper bound.

Implementation Details. The first step resembling step 2 of the presented general scheme (see section 3.4.3) is calculating the upper bounds for the whole set of vertices. Upper bound calculation is done using the function `projectBound[vercs]` implementing projection bounds as suggested by RASCAL.

The `projectBound[]` help function. The parameter `vercs` holds the set of vertices the upper bound has to be calculated for, which are saved in `D(h)(1)` in the general loop. The indices of a modular product matrix vertex code for one bond of the original molecule each, the characteristerics of these bonds are taken from the `mol_types` variables and saved to `atmtypes` and `bndtypes`. A projection bound adds up the frequency of bonds occurring in both molecules for each specific bond type (figure 3.9). Therefore, we create a list of unique types over both molecules using MOE's `uniq` function (see listing 3.16). Each element of `list(1)` contains the elements forming the bond and the bond order, and for each of those elements the frequency determined using MOE's `freq` function is saved in `list(2)` for molecule one and in `list(3)` for molecule two. The last step selects the minima from `list(2)` and `list(3)` for each bond type `list(1)` and returns the sum as an upper bound taken as `PBk` at level `h`.

```
1 list(1) = uniq tr [cat atmtypes, cat bndtypes];
2 list(2) = freq [list(1),tr [atmtypes(1),bndtypes(1)]];
3 list(3) = freq [list(1),tr [atmtypes(2),bndtypes(2)]];
4 return add apt min[list(2),list(3)];
```

Listing 3.16: Counting and return step of `projectBound[]`.

Following the concept of an upper bound, we can leave the current branch and backtrack to an upper level if the highest possible size of MCS is lower than the already obtained proposal

(step 4 in the general scheme in section 3.4.3). It is important to consider the current depth `h` in the tree, as the series of vertices chosen for branching has to be included in the MCS. The correction factor of -1 reflects the highest level starting at 1 although having no vertex chosen. If the backtracking reaches the root level 1 (`h==1`) with having no upper bound larger or the size of the current MCS, the loop and the `branchbound[]` function are exited by returning `maxis`, all found MCS vertex codes.

Otherwise, if a branch with a possible larger MCS has been detected, it has to be tracked down. The following branching step requires a new vertex out of those contained in the branch as its root. This vertex is chosen in the vertex selection part of the MCS algorithm. Our script implements a very simple method derived from the Wood algorithm (see section 3.4.3). It is based on first calculating colourclasses for the elements in question and then taking the element with the most neighbours available within the last colourclass of the set. To reduce the number of time-consuming colourclass calculations, our implementation saves the element for which the calculation has already been done. Before executing the colouring procedure, we first check whether the element at the current level `S_index_current(h)` has remained the same since the last colouring step; this element has been saved in `S_index_creationtime(h)`. If the element has changed, `checkColor[]` is called and the `S_index_creationtime(h)` set to the new value.

The `checkColor[]` help function. The `checkColor[]` function is a wrapper function for the `color[]` function itself. It includes a hash mechanism using the MOE function `tok_hash`, which is applicable to tokens, for saving calculation time if colourclasses have to be calculated for the same set twice. Currently, it only saves the last calculation result, as having the same set twice in a row happens when two subsequently chosen elements have the same neighbours. Of course, the number of saved classifications can be increased easily, but the time saving potential of this hash mechanism has not been evaluated. The last-save mode is heavily used as the `checkColor[]` function is typically called twice with the same arguments, the second time during clique guessing.

The inline-function `color[vertices, neighbourcount]` is called, if a saved classification is not available. This function is implemented according to the DSATUR algorithm proposed in [67]. `vertices` are the set of vertices for which the classification has to be done, `neighbourcount` gives the number of neighbours for each vertex in `vertices` and is used to find the starting element `actelement` of the algorithm. `vertexneighbors_colors` keeps track on the number of classes a vertex' neighbours have. The main part of the function is a loop iterating over all elements and assigning a colour to them. We choose the element with the most neighbours as a starting point.

The variable `colors` holds the assigned colourclass for each vertex, `left` is not zero for all vertices still to colour. First, we select these neighbours of `actelement` contained in `vertices` and determine all classes used for them setting `used`. At the beginning, `used` will be empty — trivially, the `newcolor` can then be set to 1. Otherwise, we create a vector from 1 to the number exceeding the highest class of `used` by 1, thus containing at least 1 (the added) class not already used for the element's neighbours. The simple **`indexof`** call then gives back the first occurrence of a not-used colourclass, which is set as `newcolor`, added to the colourclass list `colors` and zeroed in the `left` list. To keep track on the number of neighbour classes of each vertex, we then uniquely add the class of `actelement` to all neighboured vertices. Then we select that vertex having a non-zero `next` value with the most classes listed in `vertexneighbors_classes` as our next `actelement`.

```
1 local splitter = btoc sort colors;
2 local colorclasses = split[vertices[x_sort colors],splitter];
```

Listing 3.17: Extracting colourclasses out of the `colors` vector.

When no element is left, we extract the colourclasses out of the `colors` vector as shown in listing 3.17. Therefore, we use MOE's **`btoc`** function returning the counts of the class blocks previously sorted using **`sort`**. Then a **`split`** concerning the block size is applied on the elements ordered by a `colors` depending **`x_sort`** extraction from the `vertices` vector. The classes are returned in falling sizes.

After this branching decision, first the current level `h` is updated and then the next deeper level `h+1` is set up.

Therefore, the element is added to the `maxindexlist`, a variable tracking the chosen elements and resultingly the common substructure currently under investigation. Then it is removed from the current set variables `D(h)`, `S(h)` and `C(h)` to keep track on the elements already investigated. Additionally, `k(h)` holding the number of colourclasses is updated if the element chosen causes its class to be empty. As the number of colourclasses can also serve as an additional parameter for the upper bound (see section 3.4.3), we eventually reset the main upper bound `PBk(h)`.

Preparation for the next level includes setting the variable containing the complete available set, `D(h+1)` as well as setting the variable for the set that has to be taken into account specifically, `S(h+1)`, which depends on the connected mode chosen. In general, `D(h+1)` can be derived from `D(h)`, which includes all vertices being neighboured to all elements in the MCS path up to level `h` by simply intersecting `D(h)` with the neighbours of the newly chosen element (see 3.18, line 1). For the disconnected mode (**`else`**-branch of listing 3.18), `S(h+1)` equals `D(h+1)`, thus includes all vertices neighboured to the elements chosen at levels `h-`x, $\forall x \in \{1 \dots (h-1)\}$,

as all members of the clique have to be taken into account. The connected MCS variant (`if` -branch in listing 3.18) requires the new vertex, thus the new bonding, to have one atom in common with one of the already selected bonds, which must be true for both molecules. For each chosen vertex with the most neighbours `maxindex`, the connected information is stored for all neighbours of `maxindex` in the static variable `edgetype(maxindex)` (see listing 3.18, line 3). We have to look for the directly connected vertices in the then-available set `D(h+1)` (see 3.18, line 8), as the variable `cedges` contains all possible connected bonds regardless if they are part of the current clique. If we have a match, the corresponding index of the helper `D(h+1)(2)` is set to 1 and then applied to `D(h+1)(1)` resulting in the connected `S(h+1)`. The variable `D(h+1)(2)` is preserved along the path, thus holding a 1 for all possible directly connected vertices.

```
1  D(h+1) =  D(h) || [indexof[D(h)(1),neighbors(maxindex)]];
2  if connected then
3     local cedges = neighbors(maxindex)| edgetype(maxindex);
4     local i;
5     for i in cedges loop
6        local ind = indexof[i,D(h+1)(1)];
7        if ind then
8           D(h+1)(2)(ind) = 1;
9        endif
10    endloop
11    S(h+1) = D(h+1)(1) | D(h+1)(2);
12 else
13    S(h+1) = D(h+1)(1);
14 endif
```

Listing 3.18: Setting up the next level.

Altogether, this restrictive condition which can be applied if a connected MCS is requested explains why this form can be calculated much faster than a general MCS (compare [65]). Nevertheless, even for the connected type we have to track all possible vertices of the clique as well as to determine the upper bound and the colourclasses for all theoretically available vertices, as the set of directly connected bonds changes according to the current selected element. This part of the `branchbound[]` function does not take advantage of vector languages, but typically `cedges` consists of only a few elements so that the time saving potential is only marginal.

On the next deeper level, the Wood algorithm suggests finding any common clique on basis of the path including the current element by again taking the colourclass scheme, which is also

giving a lower bound. Connected components are coloured in turn, with the initial vertices chosen forming a clique within each connected component [67]. So we are taking the first element of the first colourclass, combine it with the first element of the second colourclass and continue until either one element of each colourclass has been chosen giving an MCS for the current branch, or one element chosen is not a member of the clique initiated by the first element (compare [67]). Variable `Q` is set to the proposed MCS, which is then checked against the longest found up to that point. If the proposal is larger in size or the same size as the current MCS, not already contained in the MCS-set, and no ΔY exchange has happened, it is added to the variable `maxis`. This check is done in the function `noDeltaYExchange[]`, which is relying on extracting the edges from the variable `edges1` and `edges2` and a simple **`freq`** count, which results are then compared (compare [66]).

Different sets of vertices don't necessarily lead to different SMILES representations of MCS and therefore different MCS in a chemical sense, as the same SMILES string might be found more than once within one molecule consisting of other atoms of same type. Additionally, two found MCS can also refer to the same atoms, as the vertices returned always include both molecule's structures. If for instance molecule one contains the MCS twice, molecule two only once, the algorithm will return two MCS cases, but always the same unique MCS for molecule two.

After the break condition of the **`branchbound[]`** main loop has been fulfilled and the `maxis` vector is returned, the main function **`getStructure[]`** converts the raw results into molecule parts and a SMILES string representing the MCS. At first, the vertices are transformed back to the corresponding linegraph nodes representing bonds in the original molecule, for which the **`getCommonFromMP[]`**-function is used. We again apply a **`uniq`** function at this level in order to sort out equivalent pairs of MCS. The remaining MCS structures are then converted into the corrsponding atom lists. For each pair of linegraph edges, we first extract the unique atom numbers taking part in the MCS bonds by using a simple **`get[]`** and later get the atom object keys stored in `akeys`. The system is then ready for SMILES extraction.

SMILES extraction and problems. The simplest version of obtaining a SMILES string is provided via the built-in **`sm_Extract`** function. In order to maintain the status of the atom (e.g. aromatic/non-aromatic), which is necessary for refinding the SMILES string in the molecules later, the SMILES function has to be applied on the original molecule coding the original stati of the atoms. But this unveils a big drawback: SMILES are only containing atoms, not bonds. Resultingly, as the SMILES are necessarily taken from the whole structure, the SMILES do also encode the bonds of the whole structure! As a result, if all six atoms but only five bondings of a phenyl ring are part of the MCS, thus we have an open series of aromatic carbons, which is

valid within our definitions, the **sm_Extract** fails in our sense (compare figure 3.10). To avoid the subsequently wrong MCS representation, a change in the code of the extraction procedures seems to be unevitable — we decide to go via a CTAB (connection table), which allows to explicitly save bonds of a structure and can be adapted in the SVL source code. A CTAB structure is created using **ctab_Extract**[atoms,options] defined in ctab_u.svl. It builds up the atom and bond tables from the atom input data [34, CTAB Connection Table Functions]. In order to set our own bond information and neglect the atom-connected bonds, we have to modify the **ctab_Extract** function and add a parameter leading to the **chwgtab_Extract**[atoms, bonds,options] function which is available in chwgtab.svl. **chwgtab_Extract**[atoms,bonds ,options] is bond-sensitive and only adds the bonds provided with the parameter. Additionally, the element identifier is tweaked to maintain the aromatic status information (see listing 3.19), which is necessary to later refind the MCS structures in the molecules. The second problem is caused by SMILES strings' aromaticity detection, which is originally being based on an extended version of the Hückel rule [31], whereas we use the strings rather as SMARTS search strings containing only parts of the molecule than as SMILES representations. As a result, within our script, "lonely" or "aliphatic" aromatic atoms are acceptable. Additionally, we have to take care of MOE converting "aromatic" atoms not meeting the common criteria of aromaticity into normal aliphatic atoms when creating molecules out of SMILES strings. We also need a similar form of bond extraction when preparing an MCS structure out of a molecule for further MCS determinations in the MLCS part of our script, which can be found in mlcs.svl and section 3.4.3.

```
1  //original bond creating procedure in ctab_u.svl
2  local xA = stretch [x_id atoms, aBondCount atoms];
3  local xB = indexof [cat aBonds atoms, atoms];
4  local deg = s_add [notnot xB, aBondCount atoms];
5
6  //modified bond creating procedure in chwgtab.svl
7  local xA = stretch [x_id atoms, app length bonds];
8  local xB = indexof [cat bonds, atoms];
9  local deg = s_add [notnot xB, app length bonds];
10
11 //original element identifier function
12 local el = aElement atoms;
13
14 //modified element identifier function
15 local el = aElement atoms;
```

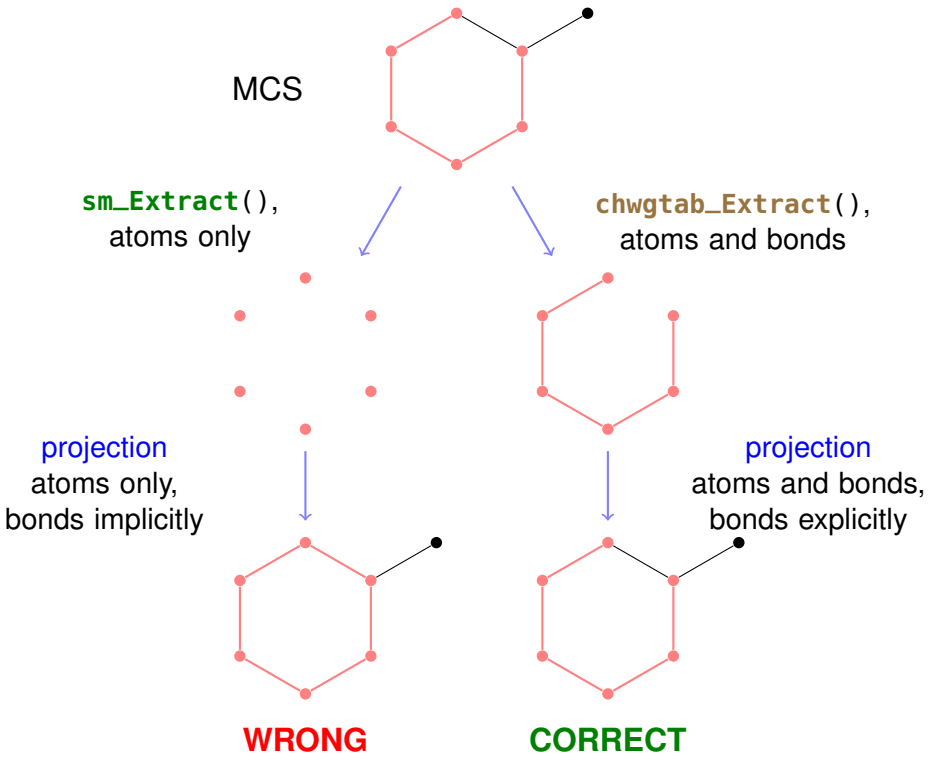


Figure 3.10.: The projection of an MCS back onto the molecule might return a wrong result if `sm_Extract`, the SMILES extraction procedure, is used. This is caused by SMILES being based on atoms only. CTABs allow to save the included bonds.

```
16 el | aInHRing atoms = (tolower el)| aInHRing atoms;
```

Listing 3.19: CTAB bond generation procedures in the original ctab_u.svl and the modified chwgtab.svl.

The parameter `bonds` expects a bond representation equal to the **`aBonds`** function result, which return vector consists of a list of bonded neighbours for each atom (compare [34, Atom Fundamental Properties]). We create this vector by iterating over the list of bonds, adding the first atom of the bond to the bond list of the second and vice versa. Then the modified function **`chwgtab_Extract[]`** is called, the resulting individualized CTAB structure can be converted to a SMILES string using the standard **`sm_ExtractFromCTAB[]`** MOE function. As this section of the **`getStructure[]`** function is only executed once per call, the time savings of a vector oriented calculation would only be marginal.

From Two Molecules to a Database

The two molecules comparison now has to be extended to cover the whole database. In principle, it is not possible to found an MLCS algorithm as described in section 3.4.3 on a simple MCS calculation algorithm only returning all MCS structures of the first two input molecules, as the risk of the second or x^{th} largest graph of two molecules being in fact the largest over the whole dataset is eminent. To cope with, Bayada et al. introduce SUPLIMIT [56], and later, *large* [57], both MCS algorithms being capable of returning a set of common structures larger than a specified threshold, which is strongly influencing the number of structures returned. This lower number is obtained by simply executing a typical MCS function, LCSA and *largest*, returning only the maximum common structures $R11$ ... $R1x$ on two structures $G1$ and $G2 \in G$, then comparing one result $R1i \in R$ with the third structure $G3$, continuing with $R2k$ and $G4$, until all $Gx \in G$ have been done. The result $R(x-1)1$ then gives one common subset as an estimate within $x-1$ comparisons, which length is set as threshold [56, 57].

We follow a very similar approach by extending our MCS implementation to return as many common substructures larger than a supplied limit as possible, which can be done by keeping the variable defining the current maximum size at the value of the threshold [57]. As a result, each new proposal potentially bigger than the limit is seen as a new MCS and therefore added to the return vector `maxis`. That procedure implies that calculation time rises, therefore we restrict these MCS retrievals to connected variants as already stated. Nonetheless, the algorithm could easily be extended to a disconnected MLCS, as the underlying MCS algorithm is also capable of returning disconnected fragments.

Of course, many of the returned fragments are just subparts of, or even identical to, other fragments, and therefore would repeatedly be iterated over. To reduce this risk, we save all

unsuccessful MLCS subparts, thus all potential MLCS structures that have failed. A structure has failed if the size of the MLCS under investigation has turned to zero before it is proved to be included in all molecules, checked either by using SMARTS search or the MCS algorithm. Subsequently, we check each new challenge if it is part of one of these failed fragments before tracking it down using SMARTS search and continue only if it has not yet been investigated.

Before starting the actual algorithm, the two molecules to start with have to be determined. Surprisingly, Bayada et al. give no clue whether they perform a certain molecule selection in their first algorithm [56], although it would be very important: The less similar the starting molecules are, the smaller is the resulting MCS. A smaller MCS then reduces the number of vertices and thus the number of possibilities the MCS algorithm has to check in all following calls, in the end the time needed to get results for all subsequent comparisons can dramatically be reduced if the right molecules are chosen (compare [57]).

The improved Bayada version [57] then includes a basic selection method by sorting the list of graphs according to size, starting with the two smallest molecules. Apart from that, we also offer the option of taking the least "similar" molecules. In general, molecular similarity analysis, dating back to the beginning 1990s, is based on the similarity property principle. So it claims that "similar molecules should have similar biological properties" [70]. In many cases, molecular similarity has to be quantified, resulting in two main steps: First, a set of attributes describing the molecules is needed. It ranges from computational expensive quantum-mechanical descriptions over 2D-/3D-properties to simple atom and bond counts [71]. Very often these properties are coded in fingerprints, which are a bit string representation of the selected descriptors [72]. Second, a calculation method is required to compare both attribute sets. This numerical value can be calculated in different ways including Hamming, Euclidean or Soergel distances as well as Tanimoto or Dice coefficients [71].

A similarity measure suited as an MCS estimation should completely focus on structural features. In our terms, that refers to atom type equivalence as well as bond multiplicity. Additionally, a guess about the absolute size of similar structural parts and not a relative measure should be returned. That is important as the calculation time typically rises dramatically for bigger structures. The Bessonov bound, already introduced in section 3.4.3, reflects that quite well. It returns an absolute number estimating the upper bound for MCS calculation [66], which is also our similarity measure.

The Bessonov bound is also calculated for each molecule compared to itself, giving the number of a molecule's bonds as the upper bound. If that number turns out out to be the smallest "similarity" measure over the dataset, the whole molecule is taken as the first MCS proposal.

It is a reasonable assumption that one whole molecule's structure, if it is smaller than the other molecules of the database, is close to the MLCS.

At any case, the molecule selection necessary after the first comparison will be done to the least similar over the whole dataset and not to the next smallest. Additionally, we use the relatively fast (when compared to MCS) SMARTS search on all molecules left within the search tree after each MCS determination to reduce the molecules left in the search tree by MCS calculation. In this way, we try to eliminate molecules already including the whole current MLCS proposal and therefore all subsequent substructures quickly. Of course, the SMILES restriction already mentioned applies here, too, but it only results in false negatives, thus returning a "not included" although the SMILES string is part of the molecule. That prolongs calculation time, but has to be accepted. It also might result in different SMILES representations of the same MCS, which are later needed as we have to refind the MLCS in all molecules during visualization using a fast method like SMARTS search.

Due to many compromises during implementation, but also due to the in general high computing effort, for some datasets or very large molecules calculation time can be beyond acceptable numbers. As we use the MLCS just for improved visualization of data, it might also be fine for many users if we detect not the real MLCS but a structure similar to it in a shorter period of time. We offer three steps in MLCS calculation, with the first two being an estimation which might be by far smaller than the correct one.

The fastest variant just performs the Bayada threshold calculation, thus a series of simple MCS calls and returns this estimation as result. The second variant is a special form of the real MLCS, as the MCS algorithm in that case returns all disconnected MCS on the first two molecules, which cover all equal parts thus giving a suitable starting point. The disconnected parts of the MCS are then split up as we intend to only get one connected MLCS, and each fragment is taken as a set of common structures like in [56, 57], which is then processed as described in section 3.4.3. To enable a correct split up, we have to determine the equivalent parts of the disconnected MCS for both molecules. This is not as trivial as it seems, as SMILES representations of the same structure can be different. We have a three-steps procedure to detect equivalent parts, the first step being based on SMILES strings, the second on a hash consisting of bonds and atoms properties and the third being another MCS call, as the real equivalence can only be checked with the MCS function itself, which then delivers the whole input molecules as results. Nonetheless, this procedure is just an approximation, as it does not return a complete set of equivalent nodes. Problems especially arise if one atom of degree ≥ 3 is potentially part of two substructures, with one compassing two bonds. At the same time, the third bond cannot be assigned to the second structure, as it then would be connected with the first. As a result, although the third bond in principle is also equivalent, it doesn't get detected.

Implementation — An MLCS

The function `calcMCSoverDataSet[database,binmode,level,batchmode]`, contained in the file mlcs.svl, triggers the MLCS calculation. It takes the path to the database for which the MLCS has to be determined as well as various options as parameters. `binmode` refers to the selection method applied to get the first two molecules, `level` gives the level of estimation. We start with reading the molecules to consider and prepare the segmentation necessary to later perform a Bessonov bound. This step is similar to the procedure `projectBound[]` in the MCS algorithm (see section 3.4.3).

In the next step, `getLeastSimilar[]` determines the two molecules to start with influenced by `binmode`. If set to one, we use the already introduced Bessonov similarity measure. The similarity measure matrices are set up, which is done in the function `getDataSetSimilarity[]` via `stretch` and `resize` commands. To avoid a `for ... loop` construction we use MOE's `apt` function, then for each pair of molecules their Bessonov bound is calculated in the function `calcSimilarity[mol1,mol2]`. Afterwards, the smallest number is searched, by looking for the containing row using `x_min app min` on the results. `app min` returns a vector containing the minimum value for each row, `x_min` subsequently gives row index. The column index is then obtained by issuing an `x_min` on the minimum-containing row.

As the molecules are sorted according to size and MOE's `x_min` commands return the first occurrence of the minimum [34, Min/Max Functions], it is also ensured that we start with the smallest molecules if more than one set has the same minimum Bessonov bound.

If `binmode` is set to 0, the smallest molecules are used, which are extracted by `first` and `second`, as the molecules are already sorted according to size.

The first two molecules give the parameters needed to call `backtracking[]`, which contains the backtracking part of the algorithm. We declare some variables, `MCSArray` holding the current MLCS proposal, `level` to keep track on the current search depth and `currentMCS` and its length for saving the current MLCS. `S(level)` saves the set to consider at a given level `level`. As there exists the possibility of `getLeastSimilar[]` returning one molecule, we have to care about that special case where the first MCS result is simply set to the molecule itself. It occurs if both database keys are the same, if not, the MCS-calculation-appropriate MCS function is called, which is a connected `getMCS[]` for the lowest, a disconnected `getMCS[]` for the special medium and a connected `getCS[]` for the real MCS detecting procedure (see section 3.4.3).

To continue, the `MCSArray` variable has to be filled, which is only simple for the one-molecule special case. Normally, the raw results have to be examined and pre-processed concerning disconnected parts and their equivalence over both molecules and duplicate SMILES strings.

Therefore, the function `processMCSArray[entrykey1,entrykey2,erg]` is called, having both database keys of the molecules and the MCS result as parameters. The `for...loop` wrapping main parts of this function iterates over each single result, where at first the SMILES strings of both molecules are extracted into `mcssmiles`. SMILES strings indicate a not completely connected structure by putting a "." between disconnected parts [34], giving the possibility to detect them easily using `strpos`. Of course, the handling of disconnected structures is only important at the medium MCS level.

For all other levels, we can simply add the current result to the local variable `MCSProposal` consisting of four vectors: A "done" flag which indicates if the SMILES has already been dealt with in the backtracking algorithm (1), the database keys of the molecules where the SMILES string has been extracted from (2), the SMILES string itself (3) and the length of it (4). To save calculation time in the backtracking part, we only add a result to the `MCSProposal` if it is new and not already included at the current level using the `MCSCheck[]` function. That function returns the result of a simple string comparison, different SMILES representations of the same MCS are not detected. Nevertheless, a modification of that function including a higher level equivalence determination can be implemented easily in a future version.

Disconnected MCS results as the special case have to be first matched with the corresponding part in the second molecules' MCS. At first, all subparts of both molecules are split, then a second loop is set up implementing the sub-part matching. The general intention is to iterate over the parts of molecule one and assign each part the equivalent MCS part index of molecule two in up to three steps with rising computational cost: The simplest assignment can be done if a SMILES string equivalence is given. If the SMILES representations are different, we use a specific `hash[]` function combining the MCS determining criteria atom and bond type to a string, which is used for equivalence search. We accept an assignment resulting from step 2, if and only if exactly one equivalent hash has been found, as other cases might indicate that the hashing is not unique. Again, at this point performance is lost, as the hash search returns more than one match if the same part is just contained twice within the disconnected MCS resulting in an in principle unnecessary third step process. This third and last step executed if still no equivalence has been found performs a `getMCS[]` call, which should return the whole MCS parts in question if they correspond, as the method is the same as that which has originally given the disconnected MCS. If all steps fail, which might happen e.g. after molecule modifications during runtime, the script exits giving an error, normally the new proposal is added to `MCSProposal`.

Within `backtracking[]`, the first-level `MCSArray` is then set. If it is empty, an MLCS is not existent and the script exits returning an empty MLCS string. Otherwise, the two initial molecules

are then removed from the set `S` and the function enters its main loop consisting of three major parts: The first deals about backtracking and stopping criteria, the second performs the SMARTS search to exclude as many molecules as possible and the third finally determines on the next steps that have to be performed.

The backtracking step itself includes a loop having two important `if...then` conditional structures. The first sets up the global break of the backtracking loop, which is reached if the largest remaining proposal at the first level is smaller than the current MLCS or all proposals have been investigated leaving no possibility left. The second includes the backtracking. We can only track back if the largest structure left at the current level is smaller than the current MLCS, similar to the break criterion. If the backtracking is possible, we additionally know that all structures of the current level have failed. Subsequently, all proposals can be added to the `failed` vector containing unsuccessful structures, which is done with the `addToFailed[]` function. Then, the actual backtracking is performed and the level reduced. At any case, we have to set the new proposal to work with. Here, we just choose the biggest, but only if it is not part of the `failed` vector, which is checked with `checkSMILES[]`.

The two helping functions `addToFailed[]` and `checkSMILES[]` keep track on unsuccessful structures. That prevents the backtracking step from iterating over the same partial structure again and again, which especially happens if there doesn't exist an MLCS. An MCS proposal SMILES can be added to the `failed` vector if its MLCS is

- empty, thus it contains no substructure included within all database molecules
- smaller than the current MLCS, thus there is no chance of getting a bigger one

Condition checks are implemented using the helping functions with the first possibility occurring after a `getMCS[]` step returns an empty string and the second if the current level contains no structure bigger than the current MLCS. `addToFailed[]` iterates over `checks`, which basically resembles `MCSArray` but contains length information for each single structure. The extension of the original `MCSArray` as well as the optional selection of a specific substructure (called with `idx` parameter if just one SMILES out of the current level has failed) is done at the very beginning of the function using `stretch` and `uniq` to get rid of duplicate SMILES. In principle, `addToFailed` tries to keep the `failed` vector as small as possible, as the `checkSMILES` function going over each element of `failed` is called once per iteration. It has to distinguish between two cases when extending the vector: The structure to add can be larger or smaller than the biggest that already failed, which case applies is determined via a mask, `biggermask`. In the first case, we have to check whether the vector holds structures which are included in the new structure, thus can be removed from the vector. In the second we only have to add the new structure if it is not already contained in our vector of structures. For both cases, an eventual addition is

decided in dependence on the result of the SMARTS match function, **sm_Match[]**. Because of the SMILES representation difficulties, we refer to the whole molecule — including the drawback of **sm_Match[]** searching within the whole connected structure, thus the molecule. This is caused by the fact that just the starting points of the search can be provided, but not the atoms which should be included in the search (compare listing 3.20). Starting with MOE 2008.10, a specific function **sm_MatchAll[]** would be available which allows to restrict the atoms to include, but usage was avoided for two reasons: On the one hand, our script should be downwards compatible to MOE 2007.09, on the other hand extensive tests unveiled errors within the function **sm_MatchAll[]** when handling aromatic non-ring atoms, which can occur due to our CTAB modifications.

```
1  for i=1, length checks(3),1 loop
2  [...]
3   local querymask = sm_Match [checks(3)(i),failsatoms];
4   if anytrue querymask then
5    local checkatoms = failsatoms | querymask;
6    local matchlist = sm_MatchAtoms[token checks(3)(i),checkatoms];
7    local splitter = app length matchlist;
8    included = anytrue app andE split[freq[cat matchlist,failsatoms],
        splitter];
9   endif
10 [...]
11 endloop
```

Listing 3.20: Determining whether a SMILES string is already included in `failed`.

To cope with the **sm_Match[]** definition problem, we have to ensure that all atoms returned by **sm_Match[]** are included in our MCS atoms. In listing 3.20, we check whether the SMILES string is contained in `failsatoms`, one of the structures in `failed`, for each string in `checks(3)`. Only if a positive result is returned for at least one query atom `failsatoms` in `querymask`, we investigate the matching atom(s) closer. Therefore, we call **sm_MatchAtoms[]** giving all matching atoms for each `checkatoms`. Line 8 in listing 3.20 then returns a frequency count for each `matchlist` atom in the set of MCS atoms, only if all atoms are contained with frequency > 0 for at least one of the sets, `included` is set to one and the new SMILES string does not have to be added. The additional **split** and **cat** commands are necessary as **freq** only operates on flat vectors [34].

The second case dealing with the new string being bigger than the already included is handled similarly in terms of SMARTS search, but here we look for the `failed` strings within the atoms of the new SMILES string. If it is contained, the old strings are deleted from `failed`.

In principle, `checkSMILES[]` operates the same way using a similar search method, but it returns just a one if the query aroms are already included in the `failed` vector or a zero if not.

After the new SMILES string to track was determined via `backtracking[]`, a `SMILESSearch []` is performed on all molecules `S(level)` left at the current level. The function returns a one if the MCS SMILES is included in the specific molecule or a zero if not. It is easy to build up `S(level+1)`, which only includes those molecules not containing the MCS SMILES at `level`.

The third part of `backtracking[]` in principle deals with two cases: If `S(level+1)` is empty, the structure connected to `level` is contained in all molecules of the database, thus an MLCS. If that structure exceeds the current MLCS (which should be the case as the script would already have backtracked during part one), we reset `currentMCS` to the whole current path, which is necessary as it might include different representations of the current MLCS derived from different levels and molecules. Additionally, the "done"-flag is set to zero indicating that this structure has been investigated at `level`, and the script loops to part one looking for a new candidate.

If there are still structures left in `S(level+1)`, we enter the next deeper level and try to shorten the proposed MCS structure by comparing it to the least similar molecule left, which again is calculated using `calcSimilarity`. We only get a flat vector as a result, the new molecule to compare with the current MCS can therefore directly be extracted with `x_min`. Before triggering the determination level-specific `getMCS[]` method for the new molecule and the current MCS proposal, we have to determine the bonds belonging to the SMILES representation of the MCS. Therefore, we use the `sm_BuildParse[]` MOE function, which converts a SMILES string to a CTAB. At this point, the script relies on `sm_BuildParse[]` returning the CTAB atoms in the same order as they were provided in the SMILES string, as we definitely need the equivalence of atoms between molecule, SMILES and CTAB. In tests, MOE always behaved the like (see section 4.1.2) — nevertheless, documentation does not explicitly guarantee it [34, SMILES Conversion], in contrast to `sm_MatchAtoms[]` used to search the MCS atoms within the molecule, where the order is kept [34, SMARTS Functions].

We hand over a minimum MCS size saved in `lengthCurrentMCS` to the MCS calculating function to improve calculation time. If the returned and processed MCS is empty, the current MCS proposal can be added to `failed`. If not, the set `S(level)` is reduced by the molecule already checked, and the `backtracking[]` returns to step one.

In the end, the loop returns the MLCS as a path of MCS results, which have to be post-processed. During the backtracking step, the whole MLCS path from level 1 to n has been

saved, as along this path an MCS representation for all database molecules has been found, either by `SMILESSearch[]` or by `getMCS[]`. We have to retain all necessary MLCS SMILES variants. At first, one MLCS has to be found, which is equal to the smallest SMILES in `currentMCS` and consists of two representations, which can be the same. The other SMILES strings might be bigger in size, as they result from higher levels where less database structures included them. All MCS structures are iterated over, and a `sm_Match[]` is performed with the already determined MLCS representations as query strings. If the query string is contained, we can remove the SMILES string, if not, we have found a different form of the same MLCS. The next step therefore is to reduce the larger SMILES string to an equivalent, alternative MLCS string, which is done using `getMCS[]`. In the end, we get a vector of various SMILES strings coding the same MLCS for all molecules of the dataset, which is returned as the result of `calcMCSoverDataSet[]`.

3.4.4. Drawing the Molecule's Contribution to the MOE Window

The most important visualization task is displaying the molecule's contribution in the MOE Window, which is handled using the four cascaded functions `show[]`, `PLS_model_show[]`, `PLS_model_drawDescr[]` and `drawView[]`.

In general, each user-triggered event modifying visualization, which includes a change of an option or adding a molecule as well as selecting the next or previous molecule out of the database, redraws all objects in the MOE Window. On the one hand, this allows for an easy option setting and simplifies tracking, but on the other hand all graphics have to be redetermined and recalculated for all molecules. Especially when displaying four molecules with the wiresphere option enabled, this might lead to some delay on slow computers.

The first function, `show[]`, keeps primarily track on the basics of visualization, the molecule graphs which have to be drawn influencing calculation and viewing options. To keep this section of the script simple, `show[]` is called every time the script has to modify one molecule of the visualized content with the exception of molecule closing, which is handled directly. Resultingly, the parameters of `show[]` include the MOE FIT model as a result of a PLS-QSAR run, the molecule which has to be changed, the multiple view number the changed molecule applies to and the database key of the molecule.

At first, we have to determine whether `show[]` is called to add a new molecule, indicated by an increase in the number of molecules visualized, or whether one of the already displayed slots has to be updated with a new molecule. This is determined via `number` which codes for the slot to change and `aktEntry` setting the database key of the new molecule. In the second case,

the old molecule and its graphics objects are destroyed via **removeMolecule[]**, followed by the creation of the new one done in either case via the built-in MOE function **mol_Create** and the settings for visualization, which in our case currently only compass the bond type (set to "cylinder" using **aSetBondLook[]**). Then the molecule positioning function **setMolPositions []** is called (see section 3.4.5). At this point, we could differentiate between different QSAR-model-types and adapt calculation procedure and the like, but here, we simply call the only possible **PLS_model_show[]** which provides the PLS-type-QSAR model.

PLS_model_show[] controls descriptor calculation with the function **Calc_Descriptors[]** and additionally performs the z-transformation $z = \frac{x-\mu}{\sigma}$ over the molecule to add or change. Then the results of transformation and descriptor calculation are stored using the `aktMol_*`-variables. Currently, our script is not able to transform over the set of displayed molecules or even the whole database, but only over one single structure. After finishing these steps, the corresponding assigning function, **PLS_model_drawDescr[]** is called.

This function optionally accepts a subset of available descriptors as parameter to allow for displaying only parts of QSAR-equations. In this case it extracts the corresponding values out of the variables `aktMol_*`, summing them up to one single displayable value per atom. The function distinguishes between the z-transformed and the normal variant. For that reason, all user modifications in the MOE Window where no molecules are added or changed (e.g. different options, visualization modes, displaying of hydrogen values) result in a call of **PLS_model_drawDescr[]**. The function then calls **drawView[]**, which sets the values, the colours, the objects and the atoms part of the MLCS for each molecule. It is the main integration function of the system (see section 3.5.2 for details).

3.4.5. A Pseudo-Multiple-View Mode for Comparison

The script allows for visualizing one compound. As a result, an inter-molecule comparison of contributions is hardly possible, especially for complex molecules: The user first has to view molecule one, remember the general picture of colour-codings, switch to molecule two and compare it with the memorized properties from molecule one. One solution supporting the user's efforts in comparing different molecules is a simultaneous depiction of more than one molecule, as an example by splitting the screen or opening a new graphics window.

Tripos SYBYL. Tripos SYBYL offers four display areas into which molecules or group of molecules, each represented by a so-called molecule area, can be loaded independently. Additionally, various view modes are available, ranging from superimposing all four areas in

the center of the screen to a quartered screen mode, where each display area has its own section of the whole graphics screen. Each molecule area can be hidden or adapted in visualization style separately, allowing for flexibly adapted individual view modes. Most importantly, the user can move or rotate each display area and even each molecule separately using the "Mouse Focus" tool or a special key combination to set the focus. SYBYL does not only rotate the molecule structure, but also includes all graphics objects associated with the currently focused atoms, as an example surfaces. Resultingly, the molecules including their graphics objects like wirespheres can be oriented easily as the user prefers (compare [73]).

MOE. In contrast to SYBYL, MOE does not implement a multiple view environment, although a special split mode in principle would be available via Render — Stereo — Left-Right. This view mode just serves as a stereo view, so the content of one side is "mirrored" and the sides cannot be addressed separately. As a result, it is of no use for our intention. Of course, it is possible to load more than one molecule to the so-called MOE Window, but these molecules are not independent of each other. There doesn't exist an easy option to arrange molecules within the window, and when using the standard dials at the bottom, MOE moves the camera position looking at the atom system and not the molecules themselves [34, MOE Window]. Nevertheless, it is possible to rotate or shift an individual molecule by pressing <SHIFT>+<ALT> during dialling, but in this case, only the atoms and bonds are included. Other graphics objects like surfaces or the wirespheres used as an activity marker introduced in section 3.4.1, and even text labels are ignored, making the whole system uninterpretable.

Workaround. Due to the necessity of a comparison mode, we decide to provide a very basic multiple-view mode, which is only "virtual" as it doesn't separate the molecules from each other, but allows for shifting and rotating molecules including graphics objects attached to them. It also offers some basic distribution options for up to four compounds. Additionally, we implement a script-specific tool for switching through the database, which also helps the user to keep control over the displayed molecules. As a result, it is possible to display up to four compounds simultaneously, re-orientate molecules to the user's requirements without losing touch with the attached graphics objects. Still, this is just a basic workaround, which does not have independent work areas resulting in possibly wrong calculation of 3D-descriptors and the like, as the molecules can act out influence on each other. It neither provides shortcuts nor options to easily select and move a molecule, which still has to be done using MOE's shortcut methods pressing <SHIFT>+<ALT>.

Basically, the concept relies on one watchdog task for each presented molecule, which controls the atom positions in very short time intervals. We set this period to 0.2s, and if any position

coordinate of one atom differs more than 0.01 from the previous check, the attached wirespheres and labels are moved for the same distance. In MOE, each graphics object is assigned a key after it has been created using `GCreate 'title'` (compare section 3.4.1). This key can be used to modify properties, as an example positions using `GTranslate[]`, which expects a graphics key as well as a translation vector as parameters [34, Graphics Objects Functions]. To keep track on the molecules and objects, some `static` variables are needed which are accessible throughout the whole programme file. `molKeys` holds the database keys of molecules currently displayed, `molecules` the molecules themselves. Another variable is needed to keep track on the objects, `runningChildTasks(i)` saving the keys of all child tasks associated with the i^{th} molecule currently displayed. The task creation function, `goFollow[atoms, obj_keys]`, is called every time textual labels or wirespheres are created (functions `wiresphereAtoms[]` and `drawView[]`), with the atoms to be watched and the keys to the graphic objects to be moved as parameters. After determining the initial atom positions via `aPos`, the main parent task is split using `task_fork` (see listing 3.21). This MOE function returns both the ID of the parent/child task and the type of the current task, in this case "" for the parent task and "child" for the child. So from this line on, two separate tasks are continuing with the script. Subsequently, an `if .... then` structure allows for differentiating between the tasks — if the second item given back is equal to "child", which is only true for the child task, the script enters the watching function. It consists of a simple `loop` and compares old and the positions of the atoms to watch. The parent task continues within the `else` branch, returning the ID of the child task.

```
1  function goFollow[atoms, gokeys]
2  local oldpos = aPos atoms;
3  local newtask = task_fork[master:'parent'];
4  if second newtask == 'child' then
5   loop
6    local newpos = aPos atoms;
7    if anytrue (abs(newpos - oldpos) > 0.01) then
8     local pos_div = tr (newpos-oldpos);
9     gokeys = cat gokeys;
10    local trans = app cat apt append [gokeys,pos_div];
11    app GTranslate trans;
12    oldpos = newpos;
13   endif
14   sleep 0.2;
15  endloop
16  exit[];
17 else
18  return first newtask;
```

```
19 endif
20 endfunction
```

Listing 3.21: The task splitting function `goFollow[]`.

If a molecule is deleted from the area, we kill the associated child tasks and remove all graphic objects as well as the molecule itself. This is done by the function `removeMolecule[number]` and its helper function `removeViewOptions[]`.

Another problem of a multiple view mode is the positioning of the individual molecules. By default, MOE arranges all molecules in the center of the MOE Window producing an overlay. To simulate different drawing areas in order to avoid graphical interference, the function `setMolPositions[]` is called every time the display window is redrawn or a molecule is closed. This function distinguishes between a single molecule, where the molecule's position remains default, and a two-, three- and four-molecules case. For two molecules, we arrange the items on the left and the right side for two reasons: On the one hand, the navigation bar displayed at the top blocks horizontal screen space, on the other hand, typical computer displays' screens have changed from 4:3 to 16:10, thus giving more horizontal space. For the three molecules view, we arrange the molecules in three columns. If four molecules have to be displayed, they are arranged within a 2×2 matrix.

Before adjusting the molecules' positions, we have to center them in the middle of the screen, which is necessary if the distribution method before has already spread the items. We simply take the arithmetic average over all atoms of each molecule, which then represents the deviation of the molecules' middle points from the real center having in MOE Window the coordinates $[0, 0, 0]$. Then the atoms are translated using `aSetPos[]`. After the centering step, the molecules are redistributed. For two in parallel, we get the maximum x-coordinate of the first as well as the minimum of the second, shifting the first molecule this maximum value to the left (subtracting), the second to the right (adding). The three molecules view takes also both the minimum and maximum values of the third molecule into account, which remains in the center. Then the first molecule is shifted by the summed up maximum x-coordinate plus the minimum x-coordinate of the third molecule to the left, the second similarly to the right. To obtain the matrix of four molecules, the y-coordinates are additionally taken into account. The first is moved its minimum y-value and maximum x-value to the top and left, respectively, the second its minimum y-value and minimum x-value to the top and right, and so on. Theoretically, additional view options could be specified for five or more molecules, they simply have to be added via an `elseif` to the `setMolPositions[]` function.

At any case, we additionally put a one-unit-wide space between all molecules to additionally separate them. The associated objects which are already in place during molecules translation

are automatically translated by the watchdog.

3.4.6. Dealing with Hydrogen Atoms

As already mentioned (compare section 3.2.3), hydrogens often have no special role within QSAR-equations. As only a few descriptors return reasonable values when calculated for hydrogen atoms and many assign a value to the whole hydride displayed as the value of the heavy atom in our script, the overall contribution of them is very small in most cases. Additionally, as many descriptors only take into account directly connected atoms, the values given to the standard hydrogen atom bonded to a carbon is often the same for nearly all hydrogens over the dataset. Nevertheless, the number of hydrogen atoms within one molecule can be quite high. Associating each of those hydrogens a separate colour as well as a separate text label would lead to an unnecessarily complicated view of the molecule. Furthermore, additional calculation resources have to be spent.

To support the user in concentrating on important molecule parts, our script does not display values and labels for hydrogens connected to a carbon atom by standard, although the values for it are calculated correctly. Within `drawView[]`, the values for H atoms are removed from the `aggvalues` vector holding the results for the atoms. Hydrogens not bonded to a carbon remain, they are determined using `aBonds` returning a key list of element bonds (see listing 3.22).

It might nonetheless be important to obtain a full view including hydrogen atoms in some cases, therefore we offer the possibility to skip the removal step. The hydrogen's display state is determined via the static variable `HON`, which is set via an option in the graphical user interface (GUI) (see section 3.5.1).

```
1 if not HON==1 then
2  aggvalues = aggvalues | (aElement notmcs_atoms <> 'H' or app anytrue (
       aElement (aBonds notmcs_atoms) <> 'C'));
3  notmcs_atoms = notmcs_atoms | (aElement notmcs_atoms <> 'H' or app
       anytrue (aElement (aBonds notmcs_atoms) <> 'C'));
4 endif
```

Listing 3.22: Removal of hydrogens from the result vector.

As a result, when adding all partial contributions over the molecule, a user will not get a sum equal to those of the equation as the hydrogen values are missing, which has to be borne in mind when working with our script.

3.5. Parts Integration and User Interface

The various script parts introduced in sections 3.3 and 3.4 have to be combined to one coherent piece of software. Additionally, we add a graphical user interface which makes using the script easy.

3.5.1. Graphical User Interface

A big part of our script is dealing with visualizing chemical information and requires a graphical display to draw molecules and graphical objects, which in our case is provided by the user interactive MOE Window. Within MOE, methods are accessible via a graphical user interface (GUI) to a high extent. A GUI is a user interface which can largely be used by a mouse. It supplies visual elements, e.g. intuitive icons or menus to access its methods and functions, which enables unskilled users to work with the software, too [15]. To follow this general MOE concept of "an interactive, windows-based chemical computing and molecular modelling tool" [34, Introducing MOE] we also implement a GUI for our script.

MOE GUI Concepts

MOE offers the easy to handle SVL Window Toolkit which is also vector-oriented. It compasses user interface components, denoted as widgets, as well as a function library. A widget offers a set of attributes that modifies behaviour, design or displayed data. The attributes can be defined during creation time of the widget within its vector or modified later which is done by calling `WindowSetAttr[]` for attributes and `WindowSetData[]` to change the data, in both cases referring to window and widget name [34, Window Toolkit Reference].

After a window has been defined within one vector `x`, first the window has to be created using the MOE function `WindowCreate x` returning the key `wkey`, then it has to be displayed with `WindowShow wkey`. To wait for a return value from the window, the function `WindowWait wkey` puts a window into a wait state. Many user interface components allow to set an "onTrigger" attribute, which controls behaviour of the script after this component has been activated, e.g. by clicking (button widget), changing (option widget) or pressing <ENTER> when it is on focus (text widget). If the trigger is enabled and a user action is noticed, the `WindowWait` function exits and returns a vector of size 2, the first part containing the current data values of all widgets defined on the window, the second giving the name of the element which was triggered [34, Window Toolkit Reference].

Resultingly, a loop can be constructed which permanently accepts user inputs, calls a specific function and waits for input again, until a certain trigger is activated or anything else causes the terminating condition to turn true [34, Window Toolkit Reference]. Our script contains two loops where different input data is processed, one is active during the setup phase, the second when the main window and the molecules are displayed. As an example, listing 3.23 shows a part of the main window loop. `navWindow` holds the key to the navigation window, "prev" and "next" are names of button widgets, name is a widget attribute. They are returned by **`WindowWait`** as the second vector of `command` (line 2, listing 3.23), and then checked in multiple `if ... then` conditional expressions.

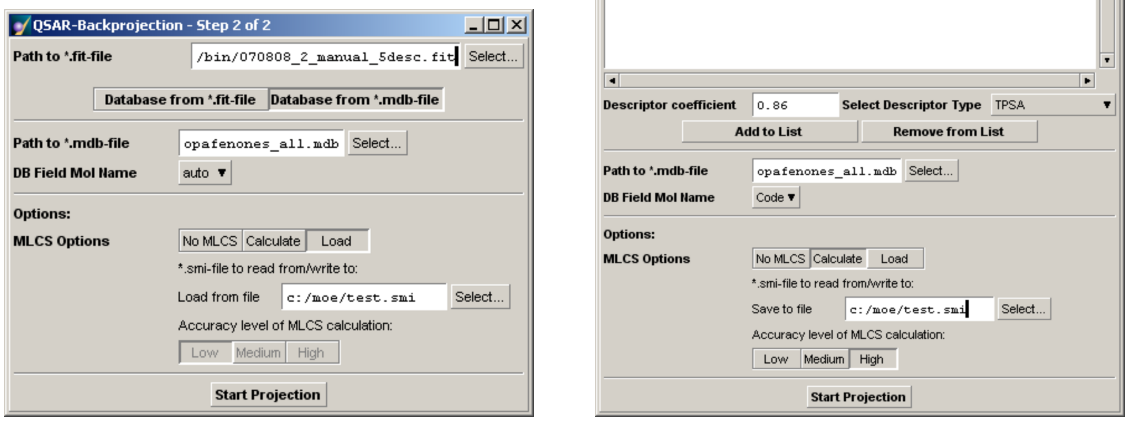
```
1 loop
2  local command = WindowWait navWindow;
3  //prev molecule
4  if command[2] == 'prev' then
5  [...] //execute 'prev' function
6  endif
7  //next molecule
8  if command[2] == 'next' then
9  [...] //execute 'prev' function
10 endif
11 [...] //other triggers are processed
12 until command[2] == 'quit'
13 endloop
```

Listing 3.23: An exemplary GUI loop, part of the main window loop.

A GUI for the Visualization Script

Apart from the MLCS part, which provides a small progress bar, all GUI functionalities are concentrated in the main file, qsar_backprojection.svl. We can distinguish between two sections of the programme (see section 3.5.2), which is also reflected in the GUI organization: First, we present two input windows retrieving necessary information about the QSAR-equation and the database to be used. Second, a main window in "prompt" style, which is integrated within the MOE Window, allows for setting options, controlling molecules to display and navigating through the database.

After the first presented window is created in the function **`presentStartWindow[]`**, either **`presentOpenWindow[]`** or **`presentInputWindow[]`** are called, depending on whether the user



(a) The *.fit-file selection setup dialog.

(b) The QSAR-equation setup dialog.

Figure 3.11.: Various GUI dialogs alleviating the visualization script's usage.

intends to provide a *.fit-file or to enter his own QSAR-equation. Both windows share a common part for general options on MLCS, which is obtained from the `commonStartWindow[]` function. As a GUI in MOE is simply defined as a vector, we can easily combine the common part using the `append[]` function (figure 3.11).

Another commonly used function, `presentOpenFileWindow[]`, returns a MOE FilePrompt window, a built-in function drawing the typical "Open File..." dialogs, where multiple options like extension masks can be specified.

The window presented to load a *.fit-file supports entering the path via a FilePrompt, a radio button switches between taking the database file out of the *.fit-file, available which is because the *.fit-file additionally saves the database it has been created on, and a user supplied file. It should be taken into account that the *.fit-file contains an absolute path to the database file, so neither the *.fit-file nor the *.mdb-file should have been renamed or moved since the creation of the QSAR-fit. For the user database mode, a "DB Field Mol Name", which specifies the database field the molecule descriptions are generated from, can be defined. The dropdown option widget therefore is filled with suitable database field names potentially containing molecule descriptions, namely all "molecule"- and "text"-typed database fields.

```
1 if isnull molecule_name_field or molecule_name_field==0 then
2  molecule_name_field = token first molFields;
3 endif
4
```

```
5 if (db_FieldType [mdbKey,molecule_name_field])=='molecule' then
6  names = (tr (cat apt db_ReadFields[mdbKey,entries,molecule_name_field]
       ))(1);
7 else
8  names = cat apt db_ReadFields[mdbKey,entries,molecule_name_field];
9 endif
10
11 names = app token names;
12 (names | names=='') = (app totok entries) | names=='';
```

Listing 3.24: Extracting the molecule's name out of the database.

If the name field remains "auto", molecule names are taken from the first molecule field saved in `molFields` (listing 3.24). As names are used to refer to the molecule via the list selection tool, empty values are set to the database entry key of the structure to avoid unset names.

The name and database selection part is also available in the input window, but instead of specifying a path to an already existent file, a listbox with all defined descriptors is shown. Below, a text input allows to specify the coefficient of the descriptor to add next, which can be selected from an option widget on the right. The available descriptors are extracted from the main descriptor holder variable `descriptors` (see section 3.2.3). The button "Add to List" finally adds the descriptor if not already existent. The button "Remove from List" removes the currently selected descriptor from the list. Of course, the resulting implicit QSAR-equation lacks a constant value, as that is not backprojected.

The general part common to both windows defines various MLCS options. A radio button set defines whether an MLCS has to be calculated or loaded from a file. The "Load from file" text serves two functions, it can either specify the path to a *.smi-file holding the MLCS smiles keys (or any SMILES strings the user wants to be highlighted) or optionally save the calculated MLCS to a *.smi-file, if "Calculate" is selected and a filename entered.

Additionally, an MLCS level can be chosen. We define three options with the first returning a quick MLCS estimation and the third giving the real MLCS in terms of our definitions (see section 3.4.3).

A click on "Start Projection" finally triggers the MLCS calculation if selected and then draws the first molecule to the MOE Window.

In general, it has to be noted in general that the trigger defined on a text field, which e.g. updates an option widget, does not fire unless the user hits <ENTER> within the field, which is due to

MOE's trigger events for a text field. That is defined as "return pressed within the text area" [34, Text Widget].

The visualization is processed using the MOE Window, where the first molecule of the database is already drawn after the calculations have finished. The GUI primarily serves as a navigation tool used to control the molecules currently shown and is therefore only small in size. In contrast to integrated MOE functions, we implement the main navigation window not as a separate window but as a "prompter". This is a special shell created within another window denoted by the `location` attribute, it has no command buttons and in principle can be seen as one large `Hbox`, which is one MOE widget grouping its content horizontally. It is recommended to keep the size of a prompter window small. Additionally, the behaviour of a prompter is different [34, Window Toolkit Reference] in terms of

- closing: <ESC> always closes the prompter window, thus the script.
- stacking: Prompters stack, and only the last is visible.

Although the actual screen size is reduced by the usage of a prompter, we prefer the easy navigation without having to switch the active window over a bigger screen size. Nevertheless, as space is limited, the navigation GUI is quite compressed and oriented horizontally. All in all, the minimal vertical screen resolution is 1024 pixels. Unfortunately, MOE does only provide a "pager" element allowing for defining different pages with sets of controls, with one active, but the pager widget does not predefine the page switching routines, which would be included in a traditional "tab" environment. To simulate the tabbing controls, three buttons are available in the upper left corner, which activate the main options : "Main", the standard navigation page, "Options" showing the option control sets and "Quit", which quits the script. The displayed page can be switched by changing the `page` value of the widget (see listing 3.25).

```
1 if command[2] == 'page2' then
2  WindowSetAttr[navWindow,
3   [panel: [page:2]]
4  ];
5 endif
```

Listing 3.25: Switching the displayed page.

The main page also includes a list of currently displayed molecules in a listbox widget, which allows to select a molecule for removal or to add one to the navigation commands from the navigation bar in the middle section. An "Add Molecule" command triggers the "add" branch of the window waiting loop, which increases the number of molecules, sets the new molecule to the first of the database and then initiates a redraw of the MOE Window by calling `show[]`.

Similarly, the “Close Molecule” command is processed, `nraff` always holds the number of the currently affected molecule.

The middle section of the main window includes the navigation bar with different navigation options, which are always applied to the highlighted molecule of the listbox: An option widget allows to directly select a molecule, the names are taken from the “DB Field Mol Names” field. Below, four buttons let the user select the first, the previous, the next or the last molecule of the database. The positions are linked to the order returned by **`db_Fields[]`** for the specific database and might therefore change after a reopening of the script. All triggers are catched in the window wait loop and call **`show[]`** after the affected molecule has been determined.

The “Details” button stacks a small atom detail information “prompter” type window over the main navigation window. While the normal view mode gives an overview over a certain descriptor set and the whole molecule, the atom details view gives an overview over the whole available descriptor set for a certain atom. A crosshair mouse pointer allows for selecting a specific atom, which detail information is then shown in the listbox. The displayed values include all descriptors, their original recalculation value for the specific atom, the factor extracted for the descriptor from the QSAR-equation as well as the resulting final result for the atom and the z-transformed value obtained from a z-transformation over all atoms of the molecule. The text set within the listbox is formatted in the function **`calcInformationFormat[]`** which uses **`swrite[]`** to cut off decimals and trim the input to the right size.

The right section of the main navigation window is filled with a listbox with all descriptors of the current model. By default, the option “whole model” is active, so all descriptors are included in the recalculation. The “multiSelect” attribute of this listbox widget is set to multi select, so one or more descriptors can be viewed individually. If all descriptors are selected, the “whole model” entry is automatically highlighted again. A change in the listbox triggers a `PLS_Model_drawDescr[]` with an updated descriptor set. During this function, the listbox entries are highlighted according to the current descriptors.

The second main page offers various options to be selected by the user. This includes mode of contribution representation, which can be switched to wiresphere or colour mode, display of MLCS if available and mode of hydrogen visualization. If the wiresphere option is selected, the scaling factor can be modified to influence the size of the wireframes.

3.5.2. Ensembling

The main script consists of two parts: On the one hand a setup part, where the user has to provide necessary options and files, on the other hand the visualization part in the MOE Win-

dow allowing the user to navigate through the selected database and retrieve information about displayed descriptors and atoms.

Resultingly, we have two loops within the script waiting for user input. The first keeps track on the setup part, the main loop is located in `ReadModel[]`, with many trigger options calling either `show[]` whenever one molecule to display is changed or `PLS_Model_DrawDescr[]` resulting in a redraw of the molecules, with the display option set by `drawView[]`.

The function `drawView[]`, which is also the last step called during the visualization function chain, can be seen as the main integration function of the script, where all partial results are combined to a final representation of the PLS-QSAR equation. It is called after the descriptor values of the molecule to change have been determined and first removes all existing objects (function `removeViewOptions[]`). Then it determines the atoms of the current molecule which belong to the MLCS.

The Multiple Largest Common Substructure

Within the script, two static variables are needed to track the current MLCS options set. Basically, if the user requires an MLCS to be calculated, `MCSCALCON` is set to a non-zero value causing the script to either read an already saved MLCS from a database or to trigger the calculation. This is done after the user has set up all necessary options and before the PLS-QSAR model is read from the file and displayed in the function `ReadModel[]`.

A value of 2 indicates the file-mode, where we read MLCS-SMILES linewise from a file specified in the GUI. We use `fread[]` function to read the line and let `sm_MatchSyntax[]` check whether the input is a valid SMILES string.

If set to 1, `MCSCALCON` indicates that we have to get an MLCS. Therefore, the MLCS calculation file mlcs.svl which should reside in the same directory as qsar_backprojection.svl is loaded and the MLCS calculation function `calcMCSoverDataSet[]` is called. Optionally, the results are saved to a file which later can be reread. The script provides some fallbacks to avoid an accidental overwriting of an already existent file. In either case, the array of MLCS-SMILES strings is saved to the static variable `mcs`.

The actual refinding of an MLCS structure within a molecule to display is performed in the function `drawView[]` itself via the fast and effective `sm_MatchAtoms[]` MOE function, returning for each atom of the molecule whether it is a starting point for the MLCS-SMILES string and, if so, the atoms matching the string. We simply take the first match. Especially if the MLCS is a very generic structure, as an example a phenyl group, this might lead to unexpected results if a molecule contains an MLCS more than once, as it might be highlighted in the wrong

section of the molecule. A real determination of the right position is difficult and could for instance be implemented using a similarity search within the MLCS neighbourhood. Some simple but effective solution could be provided in future versions, as an example a switch allowing to change the MLCS position according to the variants returned by `sm_MatchAtoms[]` or a comparison of assigned descriptor values with molecules including the MLCS structure just once.

Visualization Objects

In dependence on the static variable `HON`, `drawView[]` decides on hydrogen removal (see section 3.4.6 for details). Then the colours are determined using `calculateColors[]` as described in section 3.4.1. The static variable `viewoption`, also set via the GUI, is read to select the visualization mode, either by colouring the atoms via `colorAtoms[]` or, if the wirespheres are requested, by calling `wiresphereAtoms[]` which then creates the objects with respect to the current scaling factor. Additionally, the required text labels are created using the function `label[]`.

4. Testing and Examplary Applications of the New SVL Script

We give a short overview on tests performed to ensure proper functionality of the script and provide a brief introduction to an application test case which covers some of the scipt's features and gives an example of possible usage.

4.1. Testing the SVL Script

Apart from continuous white box testing performed during coding and assembling of the modules, we also performed black box tests based on specification based testing on the most important modules of the script, which are the descriptor calculation and the MLCS part. Therefore, we check various datasets trying to compass most of the possible problems that might arise.

4.1.1. Proof of Correct Descriptor Calculation

An exact validation of the whole script is difficult — but descriptor recalculation can easily be proven to work exactly as the original MOE calculation. As descriptors are just split up to fragmental parts on atom level, a combination of that fragments, thus the sum of values, has to be equal to the descriptor value calculated for the whole molecule. We provide an own script descriptor_proof.svl enabling users to compare values originally calculated within MOE versus the summed up recalculated values. The main function `QSAR_Backprojection_Proof[]` expects just one or more databases and an output file to write the results to. It loops over each provided database and gets both the original MOE values via the central calculation routines `QuaSAR_Descriptor_CalcInfo[]` and `QuaSAR_Descriptor_Calc[]` and the backprojection calculation values via `QSAR_Backprojection_Calc_Descriptors[]`. This function, which is globally defined in qsar_backprojection.svl, directly calls the calculation routine for the molecule and descriptors specified, `Calc_Descriptors[]`.

Results of both routines are subsequently compared. If the difference is zero, the script writes out an "OK" statement for the certain descriptor and molecule into the output file, as in these cases the recalculation obviously succeeds. It has to be noted that some differences might occur due to roundoff errors [74], although MOE uses IEEE-754 double-precision numbers to internally store the data [34, Fundamental SVL Concepts]. Therefore, we also accept values which differ less than $\frac{1}{100000}\%$ of the original calculated value and treat them as zero. Theoretically, IEEE-754 double-precision arithmetic should give approximately 16 decimal digits of accuracy [74], nonetheless, we consider this deviation small enough to cover systematic calculation errors, as it resembles the approximate single-precision accuracy (7 digits, [74]) for a descriptor value of 1. To get an overview over the results quickly, a database-wide and whole file-wide acceptance statement is also written, which is set to "NOT OK" if anything goes wrong.

Special Cases. Some descriptors can't be proven that easily. To calculate logS (see section 3.3.2), MOE adds a constant value of 0.26062, which of course has to be reproduced. For that reason, the script includes a "correction"-column, where absolute correction values can be defined to retrieve the correct number. Furthermore, the density descriptor (see section 3.3.8) can be calculated well for each atom individually, but as density is normalized, a simple sum-up and division doesn't return the correct value. For that reason, proof of the density descriptor is skipped altogether, its values are set to zero and the result to "NA".

Results

Using the script, value proving can be done quickly. Altogether, we check six databases, one containing 79 different propafenone-type ABCB1 inhibitors (see section 4.2) and one with 101 insuline receptor agonists which both serve as typical QSAR databases including only a few scaffolds. In order to check correct descriptor calculation for various structures, the third database contains 286 hERG channel blockers with quite different scaffolds and the fourth database covers diversity consisting of 1000 sample structures derived from the ChemDiv screening library [75]. The ChemDiv sample structures are retrieved using MOE's "Diverse Subset" algorithm, with MACCS structural keys fingerprint as molecule descriptor and Tanimoto Coefficient as diversity criterium. The 1000 most diverse molecules were extracted and compiled to the database.

For all four databases and for all descriptors, results show that the summed up recalculated descriptor values equal the MOE calculated values within the given range of accuracy.

For a fifth database, the NCI-60 data set with 1343 natural product type structures, one recalculation fails. For the component NCI638736, a chlorotris(chinoline-8-olato)titanium(IV)-trihydrochloride, recalculation of apol-descriptor value is -1.334 units below the original MOE calculation (84.009 versus 82.675). This is caused by NCI638736 being a titanium-complex, where the complexed metal atom is visualized with 6 covalent bondings to three hydroxy groups. Thus the oxygens have a ionization state of +1. As our apol algorithm accepts only heavy atoms as input and adds implicit hydrogens by **aHCount**, two of the hydroxy-hydrogens are missing in comparison to MOE's method, where the hydrogens are added by **Add_H**-method. As a hydrogen gets assigned a value of approx. 0.6668, both give the missing 1.334 units. Another molecule causing problems is H_2, where again apol gives back a result 1.334 too low (1.334 versus 0). Nevertheless, for the 219 MOE descriptors where recalculation is available, only 17 give back a non-zero value for the H_2 molecule.

It is expected that similar problems will occur for other metal complexes as well. Nonetheless, a sixth dataset mainly consisting of indanon and tetralon structures, which also contained a Fe-complex, proved to be no problem.

A general solution concerning the apol errors might be to change the recalculation to equal the original calculation, which would be possible if we take `molecule` as a starting point for the function instead of the already separated atoms and apply the **Add_H** procedure as MOE does.

4.1.2. MLCS and MCS algorithms

Tests of the MLCS and MCS modules compass both correctness, thus returning a structure equal to one possible M(L)CS according to our definitions, and speed, which is important due to the high computational efforts necessary to obtain the structures.

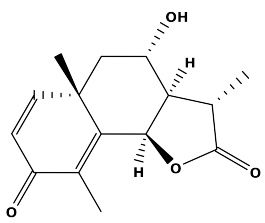
Correctness

The functionality of the MLCS algorithm is proven on the five datasets introduced in section 4.1.1 and six additional datasets, which are partly taken from literature and partly compiled from in-house datasets used for QSAR-analyses. For each, we obtain an MLCS and visually recheck the proposed structure with all molecules or the literature provided MLCS, respectively.

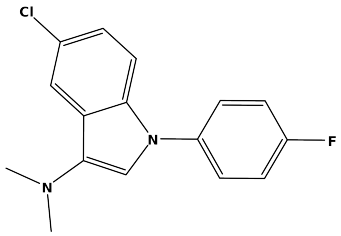
The propafenone dataset results in an MLCS of size 13. It contains the aromatic system in the center, but only one of its commonly bound two substituent moieties can be taken, as parts of the dataset are meta- or para-substituted in contrast to the lead structure propafenone, which is

ortho-substituted. One of the substituents includes the nitrogen moiety, the second is only three atoms in size when considering the small structures GPV0017, GPV0385 or GPV0389. The script therefore correctly decides to include the first one. It is extended up to the inclusion of the nitrogen atom, which is additionally methyl-substituted. The size of the nitrogen substitution is limited by GPV0240, where the nitrogen is substituted with a phenyl system, thus has only one non-aromatic bond left. On the other hand, in some structures the nitrogen is embedded into a morpholino ring with no aromatic bond at all. As a result, we conclude that the retrieved structure is maximal for our 79 propafenones.

The hERG testset turns out to have no MLCS, which the algorithm notices at the first comparison in all accuracy levels when set to the similarity-determining mode ("binmode 1"), therefore the speed remains the same. This first MCS retrieving step compasses the dataset molecules artemisin and compound 15 of [76]. It can be seen that both molecules share no bond in common according to our definitions (see figure 4.1): Both have different heteroatoms, and the only non-aromatic bondings including a carbon in compoung 15 are in the dimethylaminogroup, thus N—C bondings.

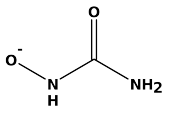


(a) Artemisin

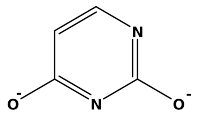


(b) Compound 15[76]

Figure 4.1.: The figures show the first structures which are compared in the hERG-dataset. The molecules share not even one bond.



(a) N-hydroxurea



(b) 2,4- pyrimidinediol

Figure 4.2.: The NCI-60 structures leading to the MLCS sized zero.

The same happens when retrieving the MLCS of the NCI-60 dataset. Again, we get an MLCS result of size zero, in this case this is caused by the substances NSC32065 N-hydroxyurea (figure 4.2a) and NSC3970 2,4-pyrimidinediol (figure 4.2b). The similarity method again proposes these molecules for the MLCS check in the first step. On the one hand, NSC3970 has, beside its aromatic core, only two bonds, which are interpreted in our context as C—O single bonds. On the other hand, the urea derivate is neither aromatic nor has it a C—O single bond (figure 4.2).

Additionally, we create a sixth dataset containing 14 selected macrolides, which are entered into MOE via the "Molecule Builder" tool. This dataset dates back into 1979 [59] and was

used as an example dataset for MLCS calculation for both Varkony and Bayada algorithms. The macrolides consist of 22 to 33 non-hydrogen atoms, five of them have a 12- and nine a 14-membered ring. The sugar components are not taken into account. According to their multiple definitions, Varkony et al. obtain various common substructures, the largest being a fragment of 12 bonds and 13 atoms around the common lactone group [59]. This MLCS and its mapping on the dataset is shown in figure 4.3. Takahashi et al. and Bayada et al. describe the same structure in [58] and [56] respectively, and an extended disconnected version is shown in [57]. A comparison of the MLCS structures obtained unveils that our script is also able to retrieve the correct connected MLCS of 12 bonds and 13 atoms. Only the fast estimation variant returns a structure one bond too small.

The seventh test dataset holds 3 structures, the narcotics meperidine, morphine and methadone and was also used by Takahashi and Bayada. Although they differ in scaffolds, an MLCS of 14 bonds can be obtained. It covers one aromatic system, substituted by a 4-dimethylaminobutyl moiety. Our script retrieves a different version, but nonetheless returns an in our terms correct MLCS of the same size as Bayada reports [56]. This example shows that in many cases there exists more than one equivalent MLCS, although only one is retrieved when applying our script.

Another four datasets are taken from QSAR-studies on the P-gp and the $GABA_A$-receptor performed in our group. One β-carboline dataset of 40 compounds compasses mainly various [9H]-pyrido[3,4-b]indole-3-carboxylate esters with 18 to 33 heavy atoms, partially with positions 4, 6 and 7 as additional points of substitutions. The β-carbolines dataset unsurprisingly returns the β-carboline core as a common structure. Visual inspection of the molecules in the dataset unveils that many molecules are only mono-substituted at position 3. At the same time, one compound has a position 4 and 6 substitution moiety, leaving only the β-carboline core common to all structures of the dataset. Additionally, this core must be the biggest partial structure, as it contains 13 out of 18 heavy atoms of the smallest compound.

The ninth datasets holds 46 benzophenone-like P-gp inhibiting structures with 23 to 38 heavy atoms. A second moiety common to many structures is the 3-amino-propan-1,2-diol linker often substituted to one phenyl system of the benzophenone structure. The MLCS turns out to be a phenyl ring. Its possible extension to a benzyl moiety is prohibited, as in some structures the only phenyl ring is attached to a nitrogen atom giving a C—N single bond. Although unspecific for the dataset and thus maybe not very useful during backprojection and analysis, it is therefore the MLCS of the dataset.

A dataset of 20 structures of P-gp inhibitors, mainly having an indanon or tetralon moiety, is also checked. The compounds consist of 21 to 42 heavy atoms. In this case, a 1-phenyl-

prop-1-en moiety is common to the whole dataset. When looking at the smallest molecule, a 1-(2-methoxyphenyl)-3-(2,4,6-trimethylphenyl)-prop-2-en-1-on (GB33), it is clear that the proposed MLCS could only be elongated by a keto or a phenyl moiety on the aryl part or a methyl group on the phenyl part. Enlarging the structure on the phenyl ring is not possible, as the dataset also contains one compound having only one phenyl ring, which is only substituted with a methoxy group (GB38). The keto group can't be added to the MLCS either, as it is not included in all structures (e.g. not in GB38). The algorithm therefore detects the correct MLCS.

We also extract the MLCS out of a dataset compassing of 33 chromane analogues, consisting of 20 to 31 heavy atoms. The chromane moiety of the MLCS is substituted with a 6-nitrile, an isopropylamino at position 4, a dimethyl at position 2 and an oxygen, which is part of various moieties (e.g. lactame or hydroxy), at position 3. Resultingly, the MLCS is close to the smallest molecule in the dataset, where the isopropylamino group and the oxygen in position 3 are part of a lactame with six members, which has only one additional oxygen. As other molecules lack a carboxy moiety and only have a hydroxy substitution at position 3, the MLCS is proven to be the largest common structure and correct, as our algorithm by definition ignores oxidation numbers of atoms. A test of the MLCS using our script shows that it is included in all molecules.

Speed Considerations

As we decide to implement exact MCS and MLCS algorithms, thus NP-complete problems, there exists a certain unpredictable limitation of input molecule size. Nevertheless, drugs typically have a relatively small molecular weight, with some authors stating a peak of 355 g/mol and the mid-50% ranging from 285 to 427 g/mol [53], thus having around 30 heavy atoms.

To measure performance, we calculate the MLCS of the various datasets introduced in section 4.1.1 on a Centrino® Pro notebook with a T7500 Intel® Core™ 2 Duo processor at 2.2 GHz using MOE 2008.10. Due to design limits of the script, only one core is used. For each of the eleven datasets, the MLCS is retrieved three times for each MLCS level and mode, then for each the average value is calculated. CPU time is determined with the built-in MOE function `cpuclock[]`, which returns the number of seconds the CPU spent on the current process [34, Time Functions]. In principle, this measure is not suited for I/O functionalities, but they are not the time critical part of our script. Therefore, the time function should be sufficient for an approximate performance measurement. Nonetheless, as this is not a comprehensive result and some kind of estimation only, we additionally terminate all other programmes running on the PC at the same time to keep influences as low as possible.

When evaluating the results obtained after applying different calculation accuracy levels and methods offered by our implementation, we can clearly show that in most cases the lowest estimation level returns a thoroughly good MLCS estimation in a very short time, which can be used for backprojection purposes (see table 4.1).

Especially for datasets where only a few comparisons are performed, the time needed to finish the similarity calculation prolongs calculation time in mode 1 operations. As the insulinmimetika, the propafenones and the macrolides dataset suggests, the time spent on this similarity calculation is invested in obtaining a better initial MCS — thus leading to slightly faster overall processing time when considering average complex datasets, which require more than one comparison (see table 4.1).

Over all datasets, the above introduced mode of medium accuracy (Accuracy Level 1) turns out to be an alternative only in some special cases, as it returns in four cases not the real MLCS. On the other hand, the prescreening method (Accuracy Level 0) performs well, it gives a wrong result in only three cases.

As far as speed of calculation is concerned, a sign-test including the raw results of table 4.1 shows that we can reject H_0 of level 0 and 1 having the same speed with a probability of error of 0.055 (two-side), H_0 of level 0 and 2 having the same speed with a probability of error of 0.055 and H_0 of level 1 and 2 having the same speed with a probability of error of 0.285. Wilcoxon signed-rank test gives similar results, H_0 of level 0 and 1 having the same speed can be rejected at a significance level of $p = 0.05$ with a probability of error of approximately 0.00016 (two-side), between level 0 and 2 with a probability of error of approximately 0.00024. Between level 1 and 2, we can't reject H_0 (probability of error of approximately 0.16758).

We therefore conclude that level 0 is significantly faster than levels 1 and 2, with less wrong results than level 1. Between levels 1 and 2, level 2 shows better results in terms of correctness (3 versus 0 errors), while in terms of speed we can't reject the H_0 of level 1 and 2 calculating equally fast.

Between the different modes 0 and 1, for a given H_0 of both modes calculating at the same speed the Wilcoxon signed-rank test results in a non-rejection of the hypothesis at a significance level of $p = 0.05$, as the probability of error would be approximately 0.23. We therefore accept both modes as being equivalent.

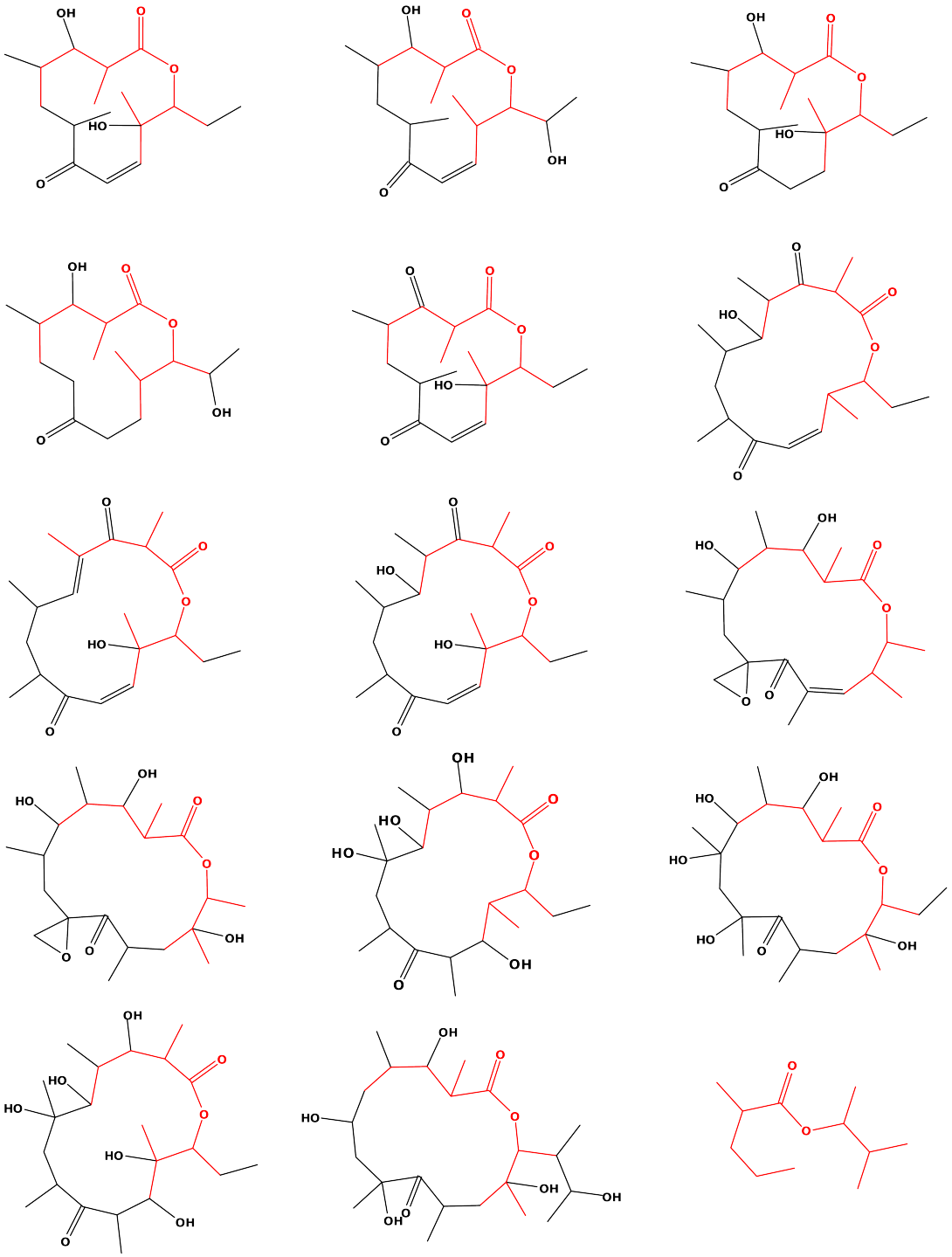


Figure 4.3.: The macrolides dataset used by Varkony, Takahashi and Bayada with the MLCS retrieved by our script highlighted in red. Bottom right: MLCS only.

Dataset	Accuracy											
	Level 0				Level 1				Level 2			
	Mode 0		Mode 1		Mode 0		Mode 1		Mode 0		Mode 1	
	t	$\|MLCS\|$	t	$\|MLCS\|$	t	$\|MLCS\|$	t	$\|MLCS\|$	t	$\|MLCS\|$	t	$\|MLCS\|$
hERG Dataset	0.79	0	3.55	0	1.55	0	3.59	0	1.97	0	3.58	0
ChemDiv 1000	0.38	0	33.8	0	0.44	0	32.2	0	0.47	0	32.1	0
NCI-60 1343	1.69	0	81.3	0	1.66	0	92.6	0	1.65	0	91.9	0
Macrolides 14	20.8	11	21.9	11	319	12	310	12	1291	12	672	12
Narcotics	0.59	14	0.96	14	5.66	14	2.39	14	2.80	14	1.90	14
Propafenones	2.10	13	3.89	13	6.25	13	21.6	13	6.14	13	7.43	13
Insulinmimetika	1.71	10	4.07	10	10.1	6	111	6	454	10	18.1	10
β-Carbolines	0.73	15	1.00	15	4.36	15	20.7	15	1.66	15	2.22	15
Benzophenone-like	5.36	4	5.60	6	27.7	6	14.1	6	15.5	6	12.3	6
Indanone/Tetralone-like	2.32	9	0.58	9	9.44	7	1.57	8	4.67	9	2.24	9
Chromane-analogues	0.40	20	0.31	20	0.97	20	0.54	20	0.91	20	0.47	20

Table 4.1.: Speed comparison of various datasets using different levels of accuracy. Level 0 is derived from the Bayada threshold calculation, level 1 based on the disconnected MCS and level 2 should give the actual MLCS. MLCS columns display the size of the retrieved structure in bonds, moieties smaller than the actual solution are highlighted in red. Times in seconds. Machine: one core of an Intel® Core™ 2 Duo T7500@2.2GHz laptop with 2 GB RAM using MOE 2008.10. Data are average of three runs each.

In table 4.2, we list the average CPU time in comparison to the results obtained from Varkony, Takahashi and Bayada. To begin with, there might be different settings and different steps of preparations distorting the result. We also don't reimplement the various algorithms, but rely on the values published by their authors, so the calculation times can't be compared exactly due to development in calculation power and different computers used.

As the comparison table shows, our implementation turns out to be by far the slowest, nonetheless the overall times seem to be reasonable low for practical usage. We calculate our MLCS from database to MLCS structure, thus covering all steps and additionally ensure that the result can be refound in all structure using a SMARTS search algorithm (compare section 3.4.3).

Dataset	SVL script	Varkony [59]	Takahashi [58]	Bayada [56]	Bayada [57]
Macrolides	672	213*	430	17	4
Narcotics	1.9	N/A	338	0.7–1.3	N/A

Table 4.2.: CPU time comparison sheet of various datasets. Times in seconds. For the own algorithm, the exact calculation method (level 2, mode 1) is chosen. * measured in 1987 by [58]

The `sm_BuildParse[]` Problem

We already explained the necessity of parameterizing the MCS functions with both bonds and atoms (see section 3.4.3). The built-in MOE `sm_BuildParse[]` [34, SMILES Conversion] function's purpose is to extract the CTAB structure out of a SMILES string. As the CTAB structure also encodes the correct bonds between the atoms, it is the only possibility of obtaining a correct atom-bond representation of a SMILES string. CTAB saves the bond type as well as the numbers of atoms within the CTAB structure that are directly connected. The second requirement is a correct atom set including the aromaticity of atoms as defined in the creating SMILES, which can only be extracted from the basic molecule structure the SMILES string is extracted from. That is caused by MOE determining aromaticity status only in dependence on the created structure, but not on the SMILES string it is based on when using `sm_Build[]`.

As a result, the parameters of the MCS calculation functions are created by both a CTAB structure and an `sm_MatchAtoms[]` on the basic molecule, the latter within the wrapping function `getMoleculesMCSAtoms[]`. It is inherent that the order of atoms is of extreme importance and has to be in both cases predictable and the same as in the SMILES string, as CTAB references binding atoms by index. For the `sm_MatchAtoms[]` function, MOE guarantees the atoms to be returned in the same order as in the SMARTS search string [34, SMARTS Functions]. Unfortunately, the programme manual does not ensure this for the `sm_BuildParse[]` counterpart,

although that seems to be the case. Unfortunately, many of the SMILES function are not part of the SVL libraries but implemented in native C, so the sourcecode is not publicly available.

To keep the risk of an unexpected result of `sm_BuildParse[]` and resultingly a definitive malfunction of the MLCS script as low as possible, we introduce the test script sm_BuildParse.svl. The function `check_sm_BuildParse[]` has three parameters, a database, a file to write the results to and an optional namefield to take within the database. For each database element, we first build the molecule and extract a SMILES string with `sm_ExtractUnique[]`. As the correct order in the return vector of `sm_MatchAtoms[]` is guaranteed, a comparison between the SMILES and the resulting CTAB structure is sufficient.

The comparison itself takes into account the atom elements only and checks the whole-molecule element string of the CTAB with the specially prepared extracted SMILES string (see listing 4.1). If the extraction procedures don't return ordered results, the positions of heteroatoms change, resulting in an inequality between two strings.

```
1 local smilesmod = string smiles;
2 smilesmod = smilesmod | smilesmod<>"H";
3 smilesmod = toupper smilesmod;
4 smilesmod =  token (smilesmod | isalpha smilesmod);
```

Listing 4.1: Preparing the modified SMILES string later compared with the CTAB element list.

To prepare the SMILES string, we eliminate all "H" letters at first. That does not only include hydrogens, but also other elements with codes starting with a capital "H", namely helium, hafnium, mercury and hassium. Nevertheless, as they have a two-letter code, the second remains as an indicator within the modified SMILES string. In general, the exclusion of hydrogens is necessary as the SMILES extraction procedure includes indicated hydrogens, whereas the CTAB only takes into account atoms currently displayed. Then, as aromatic atoms are represented by lowercase letters in SMILES while aromaticity is not part of the atom properties in CTAB, we have to uppercase the SMILES string. In the last step, we only take alphanumeric letters by applying `isalpha` function. The CTAB atom list has to be processed similarly, we also have to remove the capital "H". We then perform an exact string comparison.

Results. We test the extraction behaviour of the function using datasets already introduced and then introspect the written textfiles. For all six tested datasets including the propafenone set, the insulinmimetika set, the NCI-60, the ChemDiv 1000 diverse and the macrolides set defined by Varkony, the results of the comparison step showed that `sm_BuildParse[]` returns the atoms within the CTAB structure in the same order as provided with the SMILES string.

We therefore conclude the `sm_BuildParse[]` to return the atoms in the same order as provided in practically relevant cases.

4.2. Examplary Application – Backprojection of a QSAR-dataset of P-gp-Inhibiting Propafenones

The following proof-of-concept study on P-gp (P-glycoprotein)-inhibiting substances shows the applicability of our script in general. We outline the types of data necessary to perform an analysis and show its benefits and limitations regarding descriptor backprojection.

4.2.1. Introduction and Data

Since the detection of P-gp in 1976 by Juliano et al. [77] and its role in vincristine resistance and how to overcome it with verapamil in 1981 by Tsuruo et al. [78], it has been commonly known that substrates of P-gp are often subject to multidrug resistance in cancer patients. Classical multidrug resistance, which occurs if a cell is resistant to more than one structurally and mechanistically unrelated substances, is generally dependent on polyspecific efflux pumps of the ATP-binding cassette (ABC) transporters [79] and one main cause for rapid resistance development during cancer therapy [80]. Resultingly, inhibition of these transporters, among them ABCB1 or P-gp, ABCC1 and ABCG2 is believed to restore drug sensitivity in affected cells [81]. McDevitt et al. list three generations of P-gp-inhibitors, ranging from verapamil, which Tsuruo et al. discovered to enhance cytotoxicity of vincristine and vinblastine in 1981 [78] to the third-generation Ontogen, a diarylimidazole [82]. Nonetheless, only a few substances remain in development, among them elacridar, tariquidar, dofequidar and CBT-1™[83], and no substance is on the market by July 2009.

In 1995, a series of substances related to the class Ic antiarrhythmic drug propafenone was shown to be potential inhibitors of ABCB1 by Chiba et al. [84]. Subsequently, the number of substances has been substantially enlarged, and various SAR studies have been conducted within our group (overview in [80]).

Klopman et al. identified various biophores relevant for activity using their MULTICASE programme, which was specifically developed to investigate noncongeneric datasets. MULTICASE detects substructures which are statistically significantly dominant in active structures [85]. Many of these important biophores are also contained in propafenone and most of its

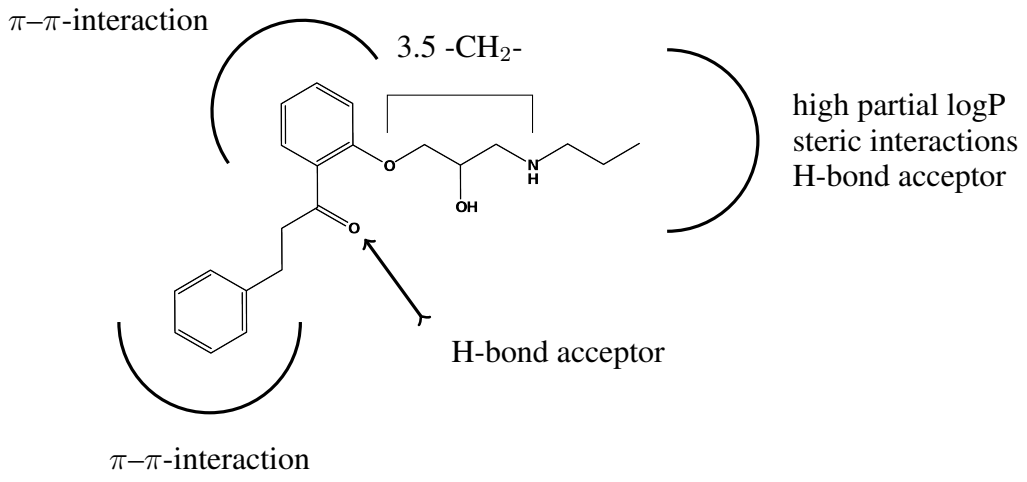


Figure 4.4.: Summary of the SAR results on propafenone-type P-gp inhibitors [80].

derivatives, as examples two aromatic rings, a basic nitrogen atom or a carbonyl group. Problems might arise due to the high flexibility and, pharmacologically, their antiarrhythmic potential [80].

The many 2D-QSAR studies already performed and published by our group suppose that [80, 86]

- activity is strongly related to the compound's overall lipophilicity within a homologous series,
- the carbonyl group is important as an H-bond acceptor,
- substituents on the nitrogen need to be lipophilic, but not too large in volume,
- the nitrogen acts as H-bond acceptor, it need not be chargable,
- the central aromatic ring and the nitrogen are ideally 3-4 methylene groups apart,
- the influence from substituents on the aromatic ring is mainly due to their σ-values (electronic rather than lipophilic contribution).

Our dataset comprises 79 propafenone-like structures. Structural modifications on the central aromatic system include ortho-, meta- and para-substituted compounds, further variations exist in the area of the second aromatic ring and the nitrogen atom moiety, which includes both alicylic and aliphatic substituents. The whole set of structures is available in appendix B. Chiba et al. derived the $-\log EC_{50}$-values in daunomycin and rhodamine 123 efflux studies as described in [87]. The set of compounds covers 5 log-units of activity ($-2.4801 \leq -\log EC_{50} \leq 2.2518$).

4.2.2. QSAR-equation

Before we can visualize a PLS-QSAR-equation, it has to be generated. As a special requirement, we should only use built-in MOE descriptors we are capable of backtracking (compare appendix A). For propafenone-like P-gp-inhibitors, our group has already published various 2D-QSAR models [87, 88] on which we base our proposals. We finally obtain the following unscaled model:

$$
\begin{aligned}
-\log EC_{50} \;=\; & -3.21 \\
& -0.02 * \text{PEOE\_VSA+1} \\
& -0.01 * \text{PEOE\_VSA-0} \\
& -0.01 * \text{PEOE\_VSA\_POS} \\
& -0.01 * \text{SMR\_VSA5} \\
& -0.01 * \text{SlogP\_VSA9} \\
& +0.76 * \text{chi0v\_C}
\end{aligned}
$$

Basic Model Data	
R^2	0.83
$RMSE$	0.37
XR^2	0.79
$XRMSE$	0.41

Table 4.3.: QSAR parameters.

Five out of six descriptors cover various parts of the compounds' VSA (see section 3.3.3), the sixth descriptor is a Hall and Kier χ-index of order zero (see section 3.3.4), which is a connectivity descriptor. The MOE calculated model shown in table 4.3 gives an R^2 of 0.83 with 6 descriptors and 79 substances with a root mean square error $RMSE$ of 0.37, the leave-one-out cross validation scheme of MOE shows an XR^2 of 0.79 with a cross validated root mean square error $XRMSE$ of 0.41. Further checks on the model quality have not been performed.

In contrast to other examples and the findings of [80], lipophilicity is not directly included in the equation. At first sight, the χ-index seems to carry many of the lipophilicity features, as both descriptors correlate with R^2 =0.83 (chi0v_C to SlogP in MOE). On the per-atom basis, the correlation is down to R^2 =0.39 (atoms taken from four molecules only, GPV0005, GPV0012, GPV0017 and GPV0789), thus significantly lower. As a result, the chi0v_C descriptor might hold different properties on the here important per-atom basis. Additionally, the examples demonstrate that despite a high correlation of two descriptors when considering the whole molecule, there can be notable differences concerning atoms.

The relative importance of descriptors, defined as the absolute normalized coefficients of the descriptors devided by the largest, states the importance of the chi0v_C descriptor, followed by SMR_VSA5 contributing approximately by the half. Other descriptors are contributing less (see table 4.4).

Relative Descriptor Importance	
chi0v_C	1.00
PEOE_VSA-0	0.14
PEOE_VSA+1	0.20
PEOE_VSA_POS	0.20
SlogP_VSA9	0.23
SMR_VSA5	0.46

Table 4.4.: Relative descriptor importance in the QSAR model.

The predicted versus $-\log EC_{50}$ correlation plot shown in figure 4.5 as well as the Z-score give a measure of the compound's predicted value's absolute deviation from its $-\log EC_{50}$ data. The Z-score is defined as the absolute difference of the predicted versus the $-\log EC_{50}$ divided by the square root of the mean square error ($RMSE$= residual mean square error) of the dataset [34, QuaSAR Model]. The plot indicates three compounds as outliers, which are defined as substances having a Z-score ≥ 2, thus a difference between predicted and $-\log EC_{50}$ values larger than twice the standard deviation of the dataset. Two of the three outliers, GPV0046 and GPV0576 are at the edge of the space covered by the equation, the third, GPV0031 has a Z-score of only 2.0001 placing it at the border of the definition.

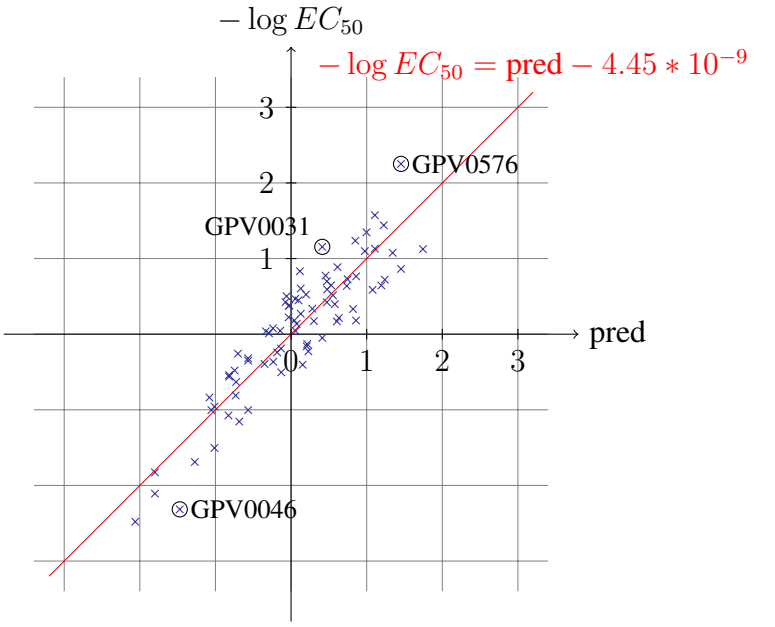


Figure 4.5.: Predicted versus actual $-\log EC_{50}$-values. Outliers with Z-score (see text) ≥ 2 are highlighted.

4.2.3. Backprojection and Results

The propafenone dataset and its respective PLS-QSAR-equation are backprojected onto the compounds' atoms using the SVL script introduced in section 3. We load the saved *.fit-file into the script, point to the *.mdb-file and set the MLCS options to "Calculate" and "High".

When comparing compounds, it is important to keep in mind the predictive quality, as it is the basis of our visualization. Despite a sufficiently low $RMSE$, the difference between the most over- (GPV0046) and underpredicted (GPV0576) compounds is 1.64 log-units of activity — an error which is also included in the atom visualization.

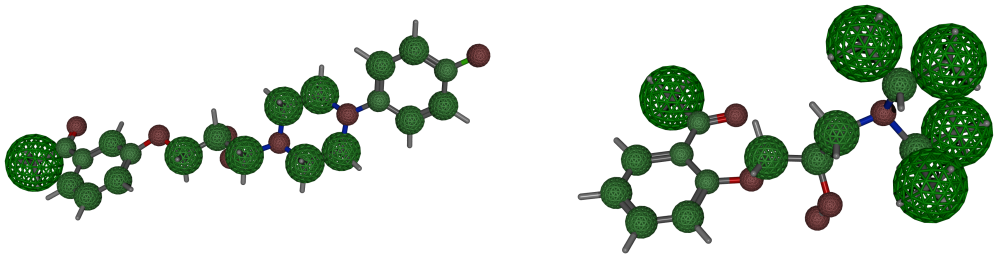
The visualization demonstrates some known features of ABCB1-inhibitors already addressed in section 4.2.1. The positive contribution of chi0v_C is shown, as well as the influence of lipophilicity in certain parts of the molecules. On the contrary, the importance of a strong H-bond-acceptor is accounted for only partially in the QSAR-equation.

Descriptors

Apart from visualizing whole QSAR-equations, our script can also be used to illustrate effects of single as well as freely definable combinations of descriptors. This should help the chemist to decide whether non-intuitive descriptors code the features they expect them to do. Additionally, by displaying the pattern of contributions, the script might support the detection of artefacts. For descriptors with known effects, the correctness of our script can be proven.

chi0v_C. Analysis of the single descriptor using the z-transform-mode shows the negative contribution of heteroatoms, which is as expected as these atoms are ignored in the only C-atom-dependent chi0v_C (see section 3.3.4). The then-negative value results from the transformation of the zero contribution of the normal mode, which of course is negative as all other contributions are only positive.

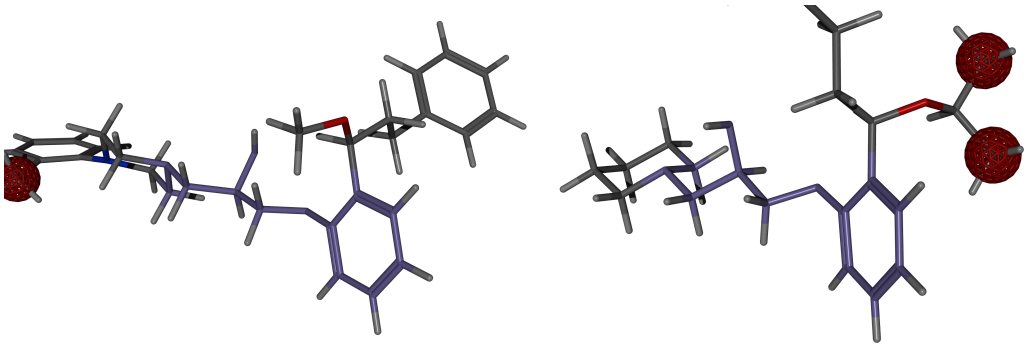
For the carbon atoms, a comparison of various atom types clearly demonstrates that aromatic carbons contribute less than aliphatic carbons, which is especially visible for substituted aromatic atoms. Terminal carbons on the other hand are especially positive for the equation value (see figure 4.6), which is shown well with the diisopropyl-substituted nitrogen of GPV0788 (figure 4.6b).



(a) GPV0386 **(b)** GPV0788

Figure 4.6.: Two example molecules showing the contribution of the chi0v_C descriptor only. The molecules are displayed using the z-transformed wiresphere mode, the scaling is 0.5. The high contribution of terminal aliphatic carbons is clearly visible.

SlogP_VSA9. This descriptor, part of Labute's binned VSA-descriptors (section 3.3.3), occurs with a negative constant in the equation. As the VSA can only be positive, the untransformed values for all atoms belonging to this certain VSA bin are negative, either. For the case of SlogP_VSA9, the bin compasses only atoms classified as very lipophilic. For our molecules, terminal aliphatic carbons as well as the aromatic carbon substituted with an ether or amine moiety meet the criteria, the terminal carbons do only so if they are not directly bound to a nitrogen or oxygen atom (e.g. methoxy group in GPV0792 versus isopropyloxy group in GPV0264, see figure 4.7). Another atom type present in this bin is the aliphatic carbon of a benzyl-substituted nitrogen (e.g. GPV0789).



(a) GPV0792, methoxy moiety **(b)** GPV0264, isopropyloxy moiety

Figure 4.7.: The SlogP and therefore the atoms the binned descriptors compass is dependent on the neighbourship. Terminal carbons are only part of the SlogP_VSA9 descriptor if they are not bound to a nitrogen- or oxygen atom. Visualization only shows the SlogP_VSA9 descriptor, scaling factor is 2.

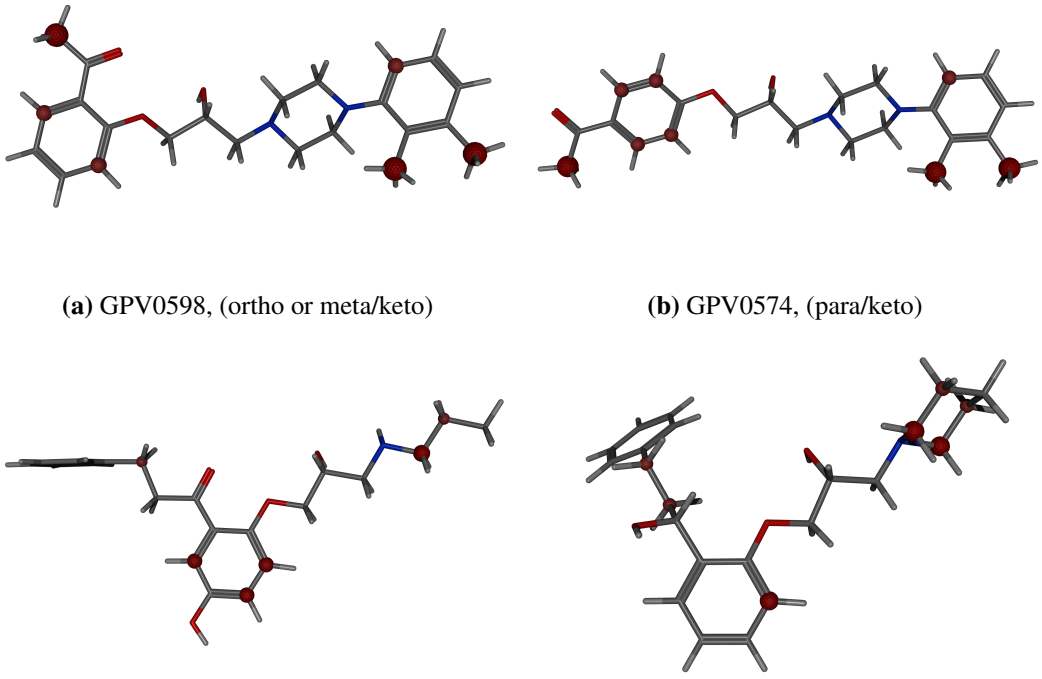
As expected, the value of aromatic carbons, which ideal VSA surfaces interfere with more neighbour atoms than those of terminal aliphatic carbons, is much smaller (contribution values -0.06 versus -0.35), again confirming the correct workflow of the script.

Additionally, the flourine present in some molecules (e.g. GPV0390, GPV0787) is part of that descriptor and gives a descriptor-specific contribution of -0.16.

SMR_VSA5. Again, the principal values of atoms falling into that bin of SMR descriptors can only be positive, as they equal the atom's VSAs. The atoms belonging to that descriptor have medium estimated MR values between 0.44 and 0.485. Most of the unsubstituted aromatic carbons fall into that group, but also the not directly N-bonded unsubstituted carbons of the piperidine moieties. A backprojection of the SMR descriptor which was performed additionally out of scope of the QSAR-equation shows that the directly bonded carbons typically have a higher MR value. As the VSA value of the atoms coded in this descriptor is similar, the resulting absolute contributions to the equation are also very similar.

PEOE_VSA_POS. This descriptor covers all molecules carrying a positive partial charge, but it takes into account the VSA value of these atoms and not the magnitude of the charge. Resultingly, it includes all hydrogens as well as all carbons connected to a heteroatom assigned a higher electronegativity. Because of the large amount of atoms, an interpretation is difficult. As shown in section 4.2.3, it is the only way our QSAR-model can distinguish between keto and hydroxy groups.

PEOE_VSA-0. In many cases, the PEOE_VSA-0 descriptor, which codes a very low negative partial charge, includes aromatic carbons which have a heteroatom-substitution in "ortho" position. Some carbons substituted with a phenyl moiety also belong to that descriptor. Because of the "ortho"-substitution dependency, this descriptor is able to distinguish between the ortho-, meta- and para-substituted central aromatic ring throughout our dataset. Additionally, it is dependent on the substitution moiety. As this central system of the molecules is part of the MLCS, we additionally have to switch off the depiction of the common structure. Then we identify four classes within our dataset: First, the ortho/meta-substituted-keto moiety as the classical propafenone, where two aromatic carbons belong to that descriptor (e.g. GPV0019, GPV0317 or GPV0598, figure 4.8a). Second, in the para-substituted-keto moiety, four aromatic carbons are part of PEOE_VSA-0 (e.g. GPV0574, figure 4.8b), while the ortho-substituted-keto moiety with an additional methoxy group (only GPV0129, figure 4.8c) has 3 members. Furthermore, within the ortho-substituted-hydroxy moiety, only one carbon of the aromatic ring belongs to the descriptor, e.g. as it is the case with GPV0088 (see figure 4.8d).



(a) GPV0598, (ortho or meta/keto)

(b) GPV0574, (para/keto)

(c) GPV0129, (ortho/keto + methoxy)

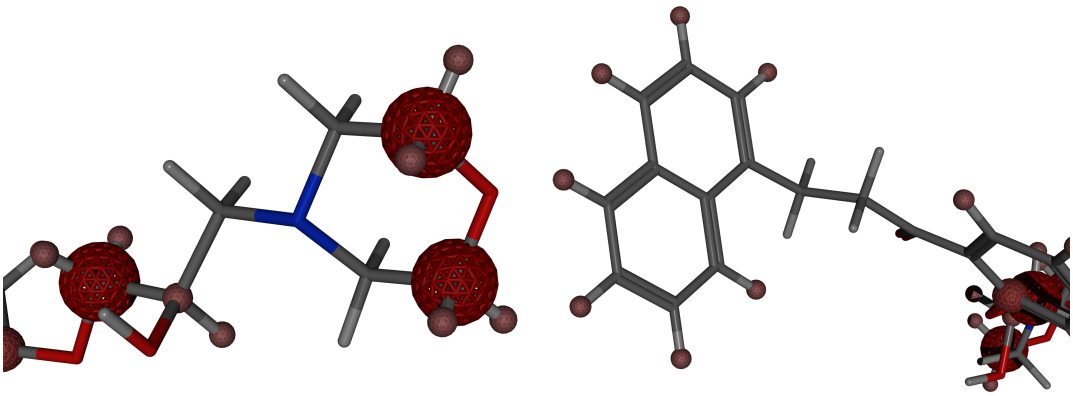
(d) GPV0088, (ortho/hydroxy)

Figure 4.8.: The four identified different atom patterns belonging to PEOE_VSA-0 descriptor. It would allow to distinguish between several substitution moieties on the central aromatic system. The figures show the untransformed PEOE_VSA-0 descriptor at a scaling of 2.

PEOE_VSA+1. Similar to the PEOE_VSA_POS descriptor, the PEOE_VSA+1 also includes atoms with a positive charge, but only these which values fall into the descriptor's bin. Among them there are many hydrogen atoms, especially those connected to an aromatic ring (e.g. figure 4.9b) and those bound to carbons substituted with an oxygen atom. Additionally, carbon atoms having a bond to an oxygen often belong to the descriptor, too. A typical example showing this pattern is the morpholino moiety present in many structures, e.g. GPV0391 (figure 4.9a) or the naphtyl moiety of e.g. GPV0374 (figure 4.9b). Of course, the negative contribution of the carbons is larger than that of a hydrogen, as the carbon's VSA values typically are bigger.

Lipophilicity

It is general accepted that higher lipophilicity correlates with higher activity [81, 80]. Although lipophilicity is not directly encoded in our equation, a comparison of compounds having different lipophilic substituents should reveal which descriptors are accounting for these features. We compare GPV0017, GPV0012, GPV0005 and GPV0180, which are identical apart from

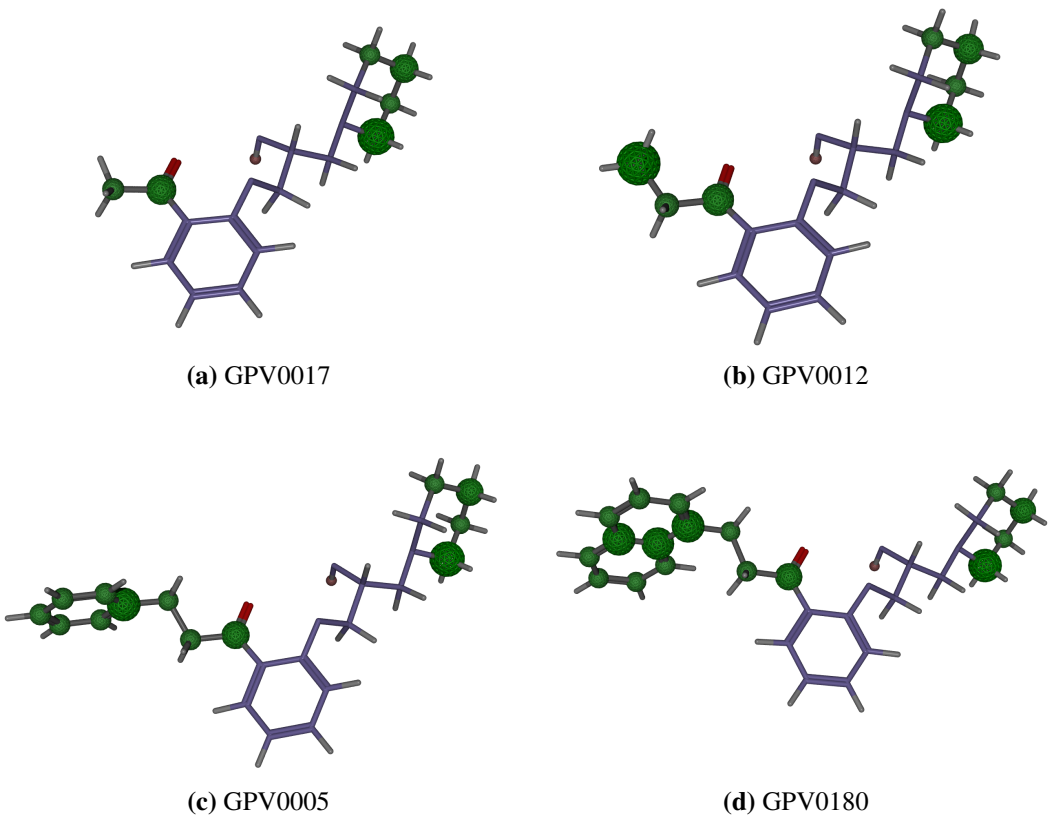


(a) GPV0391, morpholino moiety **(b)** GPV0374, naphtyl moiety

Figure 4.9.: Two typical examples of atoms part of the PEOE_VSA+1-descriptor. The value and thus the contribution is dependent on the VSA of the atom. This visualization includes hydrogens, scaling factor is 2.

different ortho substituents at the central aromatic system. These are increasingly lipophilic (estimated SLogP values of 2.11, 2.50, 3.73 and 4.88). The errors in prediction vary from +0.49 (overpredicted, GPV0017) to -0.25 (underpredicted, GPV0005), thus have an acceptable low amount of variance. GPV0017 has a 1-oxoethyl moiety, GPV0012 a 1-oxopropyl, GPV0005 a 1-oxo-3-phenylpropyl and GPV0180 a 3-naphtyl-1-oxophenyl moiety. As expected from [80], $-\log EC_{50}$ values increase throughout the row. The combined visualization of the four compounds shown in figure 4.10 immediately indicates the positive contribution of the lipophilic atoms. A closer investigation shows that the positive contribution is due to the chi0v_C descriptor, which obviously carries the lipophilic properties. Most of the atoms in the aromatic system of GPV0180 and GPV0005 are additionally contributing via the SMR_VSA5 descriptor, which has a negative factor in the equation and thus reduces the chi0v_C values. Resultingly, an aromatic carbon at this position contributes 0.36 log-units if its degree is larger than two and 0.22 log-units if its degree is two. The aliphatic carbons also have a positive value due their chi0v_C values, possibly reduced by SMR_VSA5. In this case, only aliphatic carbons with degree two belong to that descriptor, the carbons at the end of the chain and the carbonyl-carbon don't, as both have a higher SMR value. Resultingly, they contribute stronger to the equation.

A second subset consisting of GPV0788, GPV0795, GPV0009 and GPV0376 demonstrates the same. Again, the ether moiety of the central aromatic ring is a constant, in this case the amine is substituted with two isopropyl moieties. The second substituents on the aromatic ring are identical to that investigated before. GPV0788 has a 1-oxoethyl moiety, GPV0795 a 1-oxopropyl, GPV0009 a 1-oxo-3-phenylpropyl and GPV0376 a 3-naphtyl-1-oxopropyl moiety. Resultingly lipophilicity as well as the predicted activity is increasing throughout the example



(a) GPV0017 **(b)** GPV0012

(c) GPV0005 **(d)** GPV0180

Figure 4.10.: Overview over four compounds having different substituents showing the influence of lipophilicity increasing from (a) to (d). The figures show the results of the whole equation.

compounds. Nonetheless, the measured $-\log EC_{50}$ does not exactly follow this suggested rule in this case, as the most lipophilic compound is by far less active than expected (see table 4.5). A look on the visualization unveals the same scores for the varying substituent as obtained for the first set, with a constant contribution for the constant parts of the molecules. In this case, the visualization clearly shows that the PLS-QSAR equation does not take into account the specifities of these compounds. The same substituents are given the same contribution all over the dataset.

Another important conclusion in [80] refers to the substituent on the nitrogen atom, which is said to must be lipophilic but not too voluminous. On a set of six molecules (GPV0002, GPV0005, GPV0009, GPV0048, GPV0240 and GPV0242), we try to check via visualization whether the QSAR-equation represents that rule. Apart from the nitrogen substituent, the rest of the molecules is identical. Again, we immediately see the positive contribution of lipophilic atoms due to the chi0v_C descriptor, which influence is bigger than that of any other descriptor. We can't identify any descriptor accounting for volume in that case.

Compound	Substituent	SLogP-value	predicted	$-\log EC_{50}$	Error
GPV0788	1-oxoethyl	2.75	−1.05	−1.00	0.05
GPV0795	1-oxopropyl	3.14	−0.73	−0.63	0.10
GPV0009	1-oxo-3-phenylpropyl	4.36	−0.07	0.42	0.49
GPV0376	3-naphtyl-1-oxopropyl	5.51	0.82	0.33	−0.49

Table 4.5.: Important characteristics of propafenone-like structures taken for the second lipophilicity example. SLogP is a MOE descriptor estimating lipophilicity, see section 3.3.2.

H bond acceptor strength

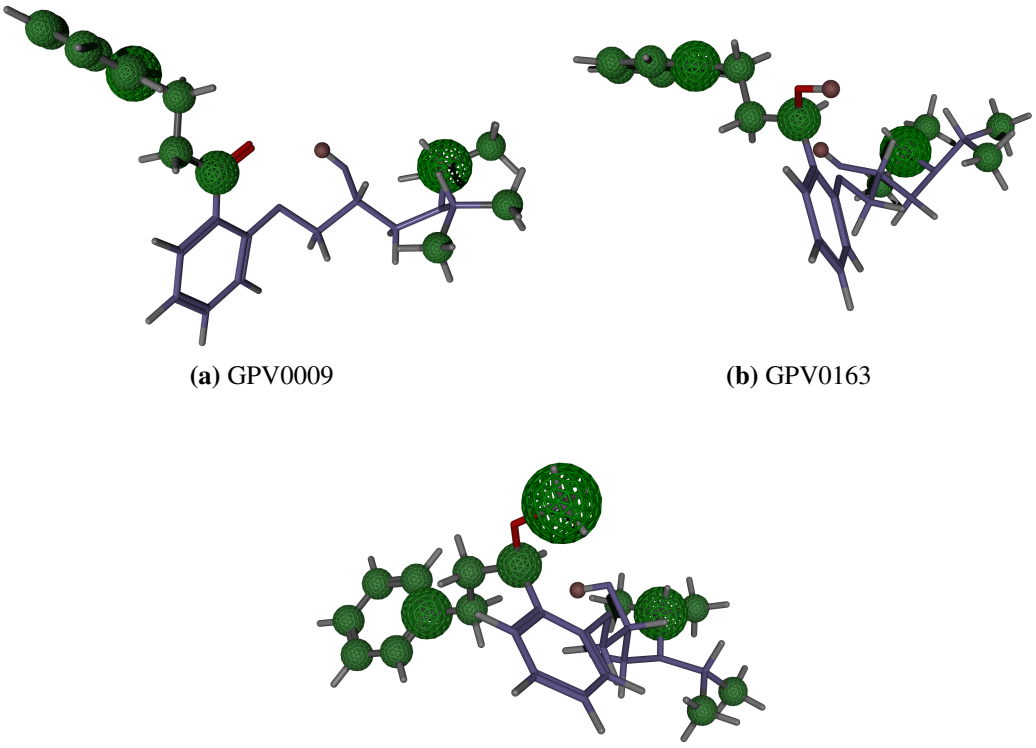
One accepted reason for reduced activity is decreased hydrogen acceptor strength at the carbonyl group of propafenone (see figure 4.4) [80, 86]. The dataset compasses three moieties at that position, apart from the carbonyl group, some compounds have a hydroxy group, which is in three cases etherified. The hydrogen acceptor strength of these moieties varies, Chiba et al. mention it to be lower for etherified and hydroxy groups [87]. Some other set of 403 experimental data classifies all three moieties as strong hydrogen bond acceptors, with ethers covering an Abraham parameter *B* value range of 0.26–0.76, ketones 0.47–0.56 and alcohols 0.47–0.60 [89]. Compounds GPV0009, GPV0163 and GPV0164 are equal apart from GPV0009 having a carbonyl moiety, GPV0163 a hydroxy group and GPV0164 being a methylether. According to the concept of hydrogen bond acceptor strength, the activity of these compounds is different, with GPV0009 being the most active (see table 4.6).

Compound	Substituent	predicted	$-\log EC_{50}$	Error
GPV0009	keto	−0.07	0.42	0.49
GPV0163	hydroxy	−0.18	−0.24	−0.06
GPV0164	methoxy	0.04	0.18	0.14
GPV0005	keto	−0.03	0.22	0.26
GPV0088	hydroxy	−0.14	0.05	0.19

Table 4.6.: Important characteristics of propafenone-like structures taken for the H-bond acceptor strength example.

We therefore compare the descriptor visualization of the three compounds. Surprisingly, the oxygen at this position is not contributing at all and ignored in the PLS-QSAR equation (see figure 4.11). The minor differences in predicted activity are only due to the methyl group, which is contributing extremely positively due to a high chi0v_C-value (0.76; it is only reduced by a negative partial contribution of PEOE_VSA_POS of -0.25 resulting in an overall value of 0.51) and the hydrogen of the hydroxy group, which is contributing negatively due to a PEOE_VSA_POS of -0.10. The same results can be found when comparing GPV0005

(keto) and GPV0088 (hydroxy), although in this case the difference in activity is represented slightly better by the hydrogen contribution (again -0.10). Resultingly, the hydrogen bond acceptor moiety is not represented in any descriptor of the QSAR-equation and thus leads to bad predictive results concerning the hydrogen bond acceptor and its strength.



(a) GPV0009 **(b)** GPV0163

(c) GPV0164

Figure 4.11.: Overview over three compounds having a different oxygen functionality with variable H-bond acceptor strength, which is strongest in (a). The figures show the results of the whole equation's backprojection.

The nitrogen atom

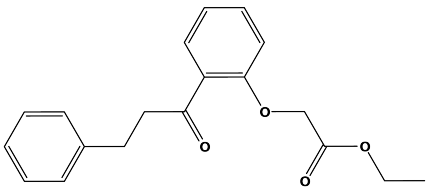
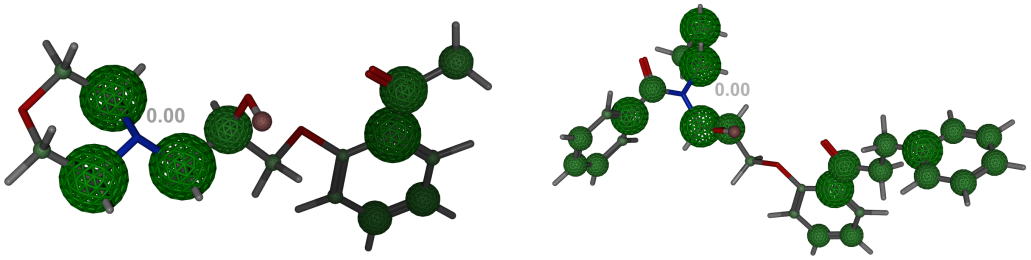


Figure 4.12.: GPV0570 (ester) shows G-gp inhibitory activity, so an N-atom is not absolutely necessary [88].

In most P-gp modulators, a nitrogen atom is present [85], later Ecker et al. concluded that it need not be basic, as some propafenone-type amides (e.g. GPV0366, $-\log EC_{50}$ -0.23) and one ester (GPV0570, not included in the test dataset, figure 4.12) also showed activity although they are not protonable in aqueous solution [88]. Thus, the H-bond acceptor strength of the nitrogen atom is quantitatively correlated to the inhibitory potential of the compound [86].

Resultingly, as the nitrogen atom or at least its hydrogen bond acceptor capability is definitely important in P-gp inhibitory activity, the atom should contribute positively to the whole PLS-QSAR equation. The nitrogen atom is preserved over the dataset and therefore part of the MLCS. We set the option "Maximum Common Substructure" on the option page within our GUI to *"false"* to visualize all atoms. A scan through the database shows that the nitrogen atom is completely ignored within the QSAR-equation, both as basic nitrogen atom and as amide (see figure 4.13).

Within that setting, the nitrogen atom is of no relevance for the equation, as a differentiation between the compounds is impossible due to it being included within all structures. Nonetheless, descriptor values typically can and should separate between amine and amide nitrogen atoms, as well as cover the pure existence of heteroatoms. Visualization clearly shows that the chemical space our QSAR-equation covers is limited to structures having the same nitrogen moiety as our structures, which is important if taken for external predictions.



(a) GPV0049 **(b)** GPV0366

Figure 4.13.: One molecule with a morpholino cycle (GPV0049) and the compound with the amide moiety (GPV0366). In both cases, the nitrogen group is assigned a 0.00 value. In this case, the MLCS is not shown.

5. Conclusion and Outlook

Using the new SVL script for MOE, it is possible to easily visualize PLS-QSAR equations, as long as they contain only suitable descriptors. These descriptors have to be fragment-based, with a fragment size of one atom. The multiple visualization options allow the user to optimally fine-tune the graphic display mode, giving a good overview on the contributions of atoms to the equation.

Concerning speed and correctness, test runs with nine different QSAR- as well as two diverse datasets don't unveil any problem with MLCS determination or descriptor recalculation, apart from errors with metal complexes. The proof-of-concept example demonstrates the abilities with a propafenone-like ABCB1-inhibitory dataset where SARs are known well. Many of the substructural features already identified can be reproduced, as an example the π-π interactions of the core aromatic system as well as the general positive contribution of highly lipophilic substituents, as parallel visualizations of congeneric sub-series within the dataset demonstrate. In the case of hydrogen bond acceptor strength, the tool is not able to point out its importance at the carbonyl group moiety. Surprisingly, the oxygen is not contributing in any way to the equation's overall value at all, it is simply ignored. The same goes for the nitrogen atom, which zero values can be shown after switching of the MLCS option. Ideally, these results are included in an QSAR-equation improvement.

Additionally, the implemented features only cover the basic needs, for an application within a production environment, especially the integration of additional descriptors would be necessary. As an example, the capabilities could be extended to some 3D-descriptors, which in some cases are also fragment-based. Additionally, to allow a general usage of fragment-based descriptors and remove the restriction of one-atom-fragments, an abstraction of the visualization options from the atom scheme would be obligatory, maybe using a contribution layer technique.

The MLCS algorithm, although sufficient for typical QSAR-datasets, should be improved in terms of speed. Improvements might include features introduced in RASCAL, e.g. horizontal pruning [66]. Provided it offers a competitible calculation time, it could also be forked from the backprojection application and distributed with a suitable GUI as a useful tool on its own. Additionally, as the definition of a "common" substructure varies, a set of fine-tuning options

should be offered. Concerning the multiple-view mode, problems might arise in terms of mutual influence of the molecule, as a workaround solution currently is the only option for a multiple view. Here, a distinct dual display mode of MOE as it exists in SYBYL would be needed. As this is a major change within the core files of MOE, this modification can't be done using the SVL scripting language.

For a general conclusion on the script's ability to identify important substructural features of both high positive and negative contribution, an extensive application to many more QSAR equations is needed. Furthermore, the results would have to be analysed more thoroughly than examplified in the proof-of-concept study. At any case, the script can be used to get an overview over features covered in the equation quickly. But as the script does not interprete but only visualizes, nothing can be shown which is not represented in the underlying QSAR model.

A. List of Available MOE Descriptors

The following table lists all 2D-descriptors of MOE 2007.09 and their availability in the script. Section 3.2 provides further details.

Descriptor Name	Details	File in MOE 2007.09	Availability?
SlogP_VSA0	section 3.3.3	q_slogp.svl	Y
SlogP_VSA1	section 3.3.3	q_slogp.svl	Y
SlogP_VSA2	section 3.3.3	q_slogp.svl	Y
SlogP_VSA3	section 3.3.3	q_slogp.svl	Y
SlogP_VSA4	section 3.3.3	q_slogp.svl	Y
SlogP_VSA5	section 3.3.3	q_slogp.svl	Y
SlogP_VSA6	section 3.3.3	q_slogp.svl	Y
SlogP_VSA7	section 3.3.3	q_slogp.svl	Y
SlogP_VSA8	section 3.3.3	q_slogp.svl	Y
SlogP_VSA9	section 3.3.3	q_slogp.svl	Y
SMR_VSA0	section 3.3.3	q_slogp.svl	Y
SMR_VSA1	section 3.3.3	q_slogp.svl	Y
SMR_VSA2	section 3.3.3	q_slogp.svl	Y
SMR_VSA3	section 3.3.3	q_slogp.svl	Y
SMR_VSA4	section 3.3.3	q_slogp.svl	Y
SMR_VSA5	section 3.3.3	q_slogp.svl	Y
SMR_VSA6	section 3.3.3	q_slogp.svl	Y
SMR_VSA7	section 3.3.3	q_slogp.svl	Y
PEOE_VSA-6	section 3.3.3	q_charge.svl	Y
PEOE_VSA-5	section 3.3.3	q_charge.svl	Y
PEOE_VSA-4	section 3.3.3	q_charge.svl	Y
PEOE_VSA-3	section 3.3.3	q_charge.svl	Y

PEOE_VSA-2	section 3.3.3	q_charge.svl	Y
PEOE_VSA-1	section 3.3.3	q_charge.svl	Y
PEOE_VSA-0	section 3.3.3	q_charge.svl	Y
PEOE_VSA+0	section 3.3.3	q_charge.svl	Y
PEOE_VSA+1	section 3.3.3	q_charge.svl	Y
PEOE_VSA+2	section 3.3.3	q_charge.svl	Y
PEOE_VSA+3	section 3.3.3	q_charge.svl	Y
PEOE_VSA+4	section 3.3.3	q_charge.svl	Y
PEOE_VSA+5	section 3.3.3	q_charge.svl	Y
PEOE_VSA+6	section 3.3.3	q_charge.svl	Y
TPSA	section 3.3.6	q_surf.svl	Y
SMR	section 3.3.2	q_slogp.svl	Y
mr	section 3.3.2	q_mref.svl	Y
SlogP	section 3.3.2	q_slogp.svl	Y
logPo/w	section 3.3.2	q_logp.svl	N
logS	section 3.3.2	q_logs.svl	Y
Kier1	section 3.3.4	q_kappa.svl	N
Kier2	section 3.3.4	q_kappa.svl	N
Kier3	section 3.3.4	q_kappa.svl	N
KierA1	section 3.3.4	q_kappa.svl	N
KierA2	section 3.3.4	q_kappa.svl	N
KierA3	section 3.3.4	q_kappa.svl	N
KierFlex	section 3.3.4	q_kappa.svl	N
chi0v	section 3.3.4	q_btop.svl	Y
chi0v_C	section 3.3.4	q_btop.svl	Y
chi1v	section 3.3.4	q_btop.svl	N
chi1v_C	section 3.3.4	q_btop.svl	N
BCUT_PEOE_0	section 3.3.7	q_bcut.svl	N
BCUT_PEOE_1	section 3.3.7	q_bcut.svl	N
BCUT_PEOE_2	section 3.3.7	q_bcut.svl	N
BCUT_PEOE_3	section 3.3.7	q_bcut.svl	N
BCUT_SMR_0	section 3.3.7	q_bcut.svl	N

BCUT_SMR_1	section 3.3.7	q_bcut.svl	N
BCUT_SMR_2	section 3.3.7	q_bcut.svl	N
BCUT_SMR_3	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_0	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_1	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_2	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_3	section 3.3.7	q_bcut.svl	N
GCUT_PEOE_0	section 3.3.7	q_bcut.svl	N
GCUT_PEOE_1	section 3.3.7	q_bcut.svl	N
GCUT_PEOE_2	section 3.3.7	q_bcut.svl	N
GCUT_PEOE_3	section 3.3.7	q_bcut.svl	N
GCUT_SMR_0	section 3.3.7	q_bcut.svl	N
GCUT_SMR_1	section 3.3.7	q_bcut.svl	N
GCUT_SMR_2	section 3.3.7	q_bcut.svl	N
GCUT_SMR_3	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_0	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_1	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_2	section 3.3.7	q_bcut.svl	N
GCUT_SLOGP_3	section 3.3.7	q_bcut.svl	N
diameter	section 3.3.7	q_adjm.svl	N
petitjean	section 3.3.7	q_adjm.svl	N
petitjeanSC	section 3.3.7	q_adjm.svl	N
radius	section 3.3.7	q_adjm.svl	N
VDistEq	section 3.3.7	q_adjm.svl	N
VDistMa	section 3.3.7	q_adjm.svl	N
VAdjEq	section 3.3.7	q_btop.svl	N
VAdjMa	section 3.3.7	q_btop.svl	N
wienerPath	section 3.3.7	q_adjm.svl	N
wienerPol	section 3.3.7	q_adjm.svl	N
vsa_acc	section 3.3.8	q_vdw2d.svl	Y
vsa_acid	section 3.3.8	q_vdw2d.svl	Y
vsa_base	section 3.3.8	q_vdw2d.svl	Y

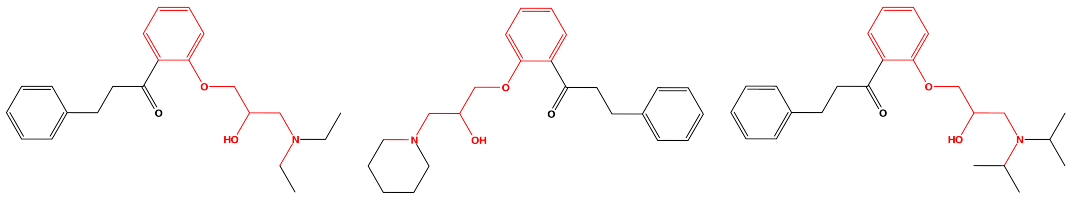
vsa_don	section 3.3.8	q_vdw2d.svl	Y
vsa_hyd	section 3.3.8	q_vdw2d.svl	Y
vsa_other	section 3.3.8	q_vdw2d.svl	Y
vsa_pol	section 3.3.8	q_vdw2d.svl	Y
vsa_area	section 3.3.8	q_vdw2d.svl	Y
vsa_vol	section 3.3.8	q_vdw2d.svl	Y
density	section 3.3.8	q_vdw2d.svl	N
a_aro	section 3.3.9	q_btop.svl	Y
a_count	section 3.3.9	q_btop.svl	Y
a_IC	section 3.3.9	q_btop.svl	N
a_ICM	section 3.3.9	q_btop.svl	N
a_nH	section 3.3.9	q_btop.svl	Y
b_1rotN	section 3.3.9	q_btop.svl	N
b_1rotR	section 3.3.9	q_btop.svl	N
b_ar	section 3.3.9	q_btop.svl	N
b_count	section 3.3.9	q_btop.svl	N
b_double	section 3.3.9	q_btop.svl	N
b_rotN	section 3.3.9	q_btop.svl	N
b_rotR	section 3.3.9	q_btop.svl	N
b_single	section 3.3.9	q_btop.svl	N
b_triple	section 3.3.9	q_btop.svl	N
b_heavy	section 3.3.9	q_btop.svl	N
a_heavy	section 3.3.9	q_btop.svl	Y
a_nB	section 3.3.9	q_btop.svl	Y
a_nBr	section 3.3.9	q_btop.svl	Y
a_nC	section 3.3.9	q_btop.svl	Y
a_nCl	section 3.3.9	q_btop.svl	Y
a_nF	section 3.3.9	q_btop.svl	Y
a_nI	section 3.3.9	q_btop.svl	Y
a_nN	section 3.3.9	q_btop.svl	Y
a_nO	section 3.3.9	q_btop.svl	Y
a_nP	section 3.3.9	q_btop.svl	Y

a_nS	section 3.3.9	q_btop.svl	Y
a_acc	section 3.3.9	q_vdw2d.svl	Y
a_acid	section 3.3.9	q_vdw2d.svl	Y
a_base	section 3.3.9	q_vdw2d.svl	Y
a_don	section 3.3.9	q_vdw2d.svl	Y
a_hyd	section 3.3.9	q_vdw2d.svl	Y
Weight	section 3.3.9	q_btop.svl	Y
FCharge	section 3.3.9	q_btop.svl	Y
nmol	section 3.3.9	q_btop.svl	N
rings	section 3.3.9	q_btop.svl	N
reactive	section 3.3.9	q_btop.svl	N
chiral	section 3.3.9	q_btop.svl	Y
chiral_u	section 3.3.9	q_btop.svl	Y
apol	section 3.3.2	q_mref.svl	Y
bpol	section 3.3.2	q_mref.svl	N
PEOE_PC+	section 3.3.1	q_charge.svl	Y
PEOE_PC-	section 3.3.1	q_charge.svl	Y
PEOE_RPC+	section 3.3.1	q_charge.svl	N
PEOE_RPC-	section 3.3.1	q_charge.svl	N
PEOE_VSA_FHYD	section 3.3.1	q_charge.svl	Y
PEOE_VSA_FNEG	section 3.3.1	q_charge.svl	Y
PEOE_VSA_FPNEG	section 3.3.1	q_charge.svl	Y
PEOE_VSA_FPOL	section 3.3.1	q_charge.svl	Y
PEOE_VSA_FPOS	section 3.3.1	q_charge.svl	Y
PEOE_VSA_FPPOS	section 3.3.1	q_charge.svl	Y
PEOE_VSA_HYD	section 3.3.1	q_charge.svl	Y
PEOE_VSA_NEG	section 3.3.1	q_charge.svl	Y
PEOE_VSA_PNEG	section 3.3.1	q_charge.svl	Y
PEOE_VSA_POL	section 3.3.1	q_charge.svl	Y
PEOE_VSA_POS	section 3.3.1	q_charge.svl	Y
PEOE_VSA_PPOS	section 3.3.1	q_charge.svl	Y
PC+	section 3.3.1	q_charge.svl	Y

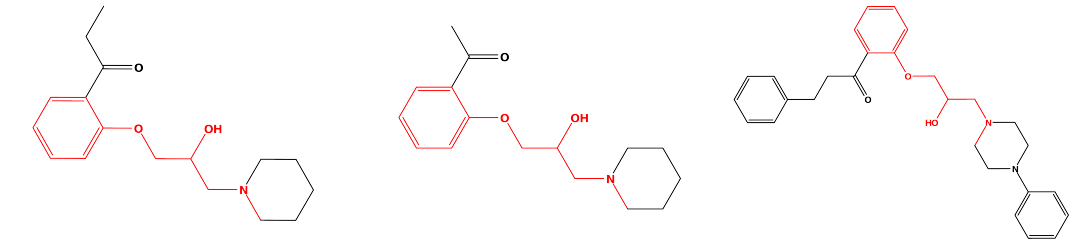
PC-	section 3.3.1	q_charge.svl	Y
RPC+	section 3.3.1	q_charge.svl	N
RPC-	section 3.3.1	q_charge.svl	N
Q_PC+	section 3.3.1	q_charge.svl	Y
Q_PC-	section 3.3.1	q_charge.svl	Y
Q_RPC+	section 3.3.1	q_charge.svl	N
Q_RPC-	section 3.3.1	q_charge.svl	N
Q_VSA_FHYD	section 3.3.1	q_charge.svl	Y
Q_VSA_FNEG	section 3.3.1	q_charge.svl	Y
Q_VSA_FPNEG	section 3.3.1	q_charge.svl	Y
Q_VSA_FPOL	section 3.3.1	q_charge.svl	Y
Q_VSA_FPOS	section 3.3.1	q_charge.svl	Y
Q_VSA_FPPOS	section 3.3.1	q_charge.svl	Y
Q_VSA_HYD	section 3.3.1	q_charge.svl	Y
Q_VSA_NEG	section 3.3.1	q_charge.svl	Y
Q_VSA_PNEG	section 3.3.1	q_charge.svl	Y
Q_VSA_POL	section 3.3.1	q_charge.svl	Y
Q_VSA_POS	section 3.3.1	q_charge.svl	Y
Q_VSA_PPOS	section 3.3.1	q_charge.svl	Y
lip_don	section 3.3.10	q_btop.svl	Y
lip_acc	section 3.3.10	q_btop.svl	Y
lip_violation	section 3.3.10	q_btop.svl	N
lip_druglike	section 3.3.10	q_btop.svl	N
opr_nrot	section 3.3.10	q_btop.svl	N
opr_brigid	section 3.3.10	q_btop.svl	N
opr_nring	section 3.3.10	q_btop.svl	N
opr_violation	section 3.3.10	q_btop.svl	N
opr_leadlike	section 3.3.10	q_btop.svl	N

B. The Propafenones Dataset

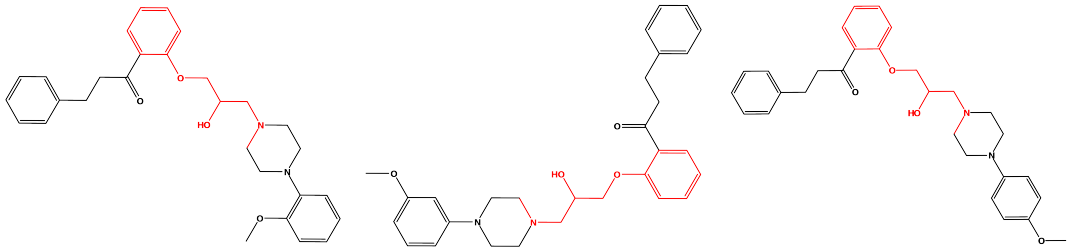
The propafenones-like test dataset compasses the following compounds. The MLCS has been highlighted in red.



(1) GPV0002 **(2)** GPV0005 **(3)** GPV0009

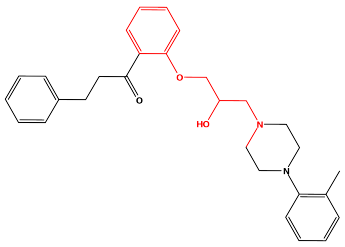


(4) GPV0012 **(5)** GPV0017 **(6)** GPV0019

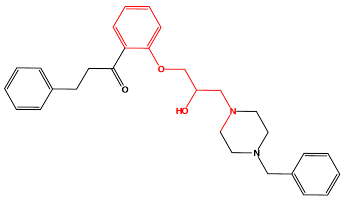


(7) GPV0021 **(8)** GPV0023 **(9)** GPV0025

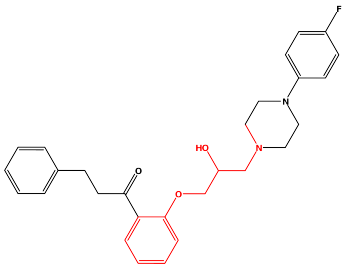
Figure B.1.: List of propafenone-like compounds in the test-dataset.



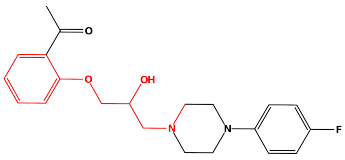
(10) GPV0027



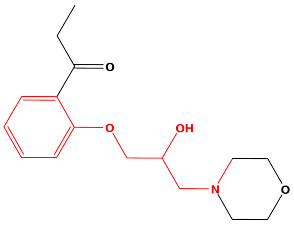
(11) GPV0029



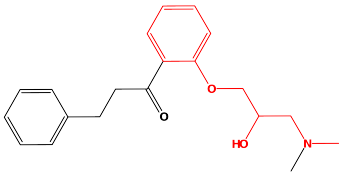
(12) GPV0031



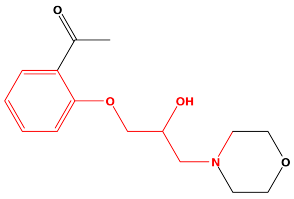
(13) GPV0045



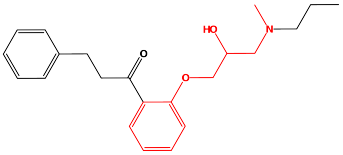
(14) GPV0046



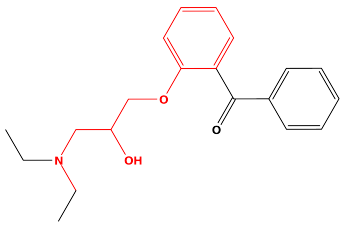
(15) GPV0048



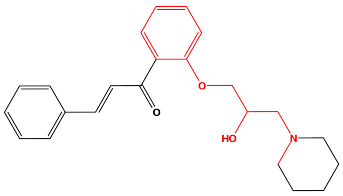
(16) GPV0049



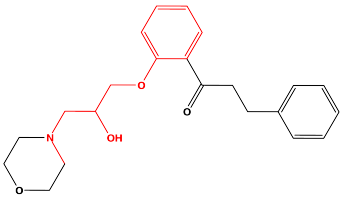
(17) GPV0050



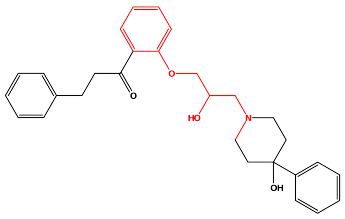
(18) GPV0051



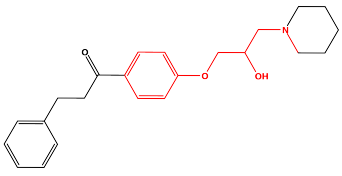
(19) GPV0056



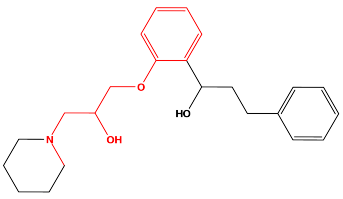
(20) GPV0057



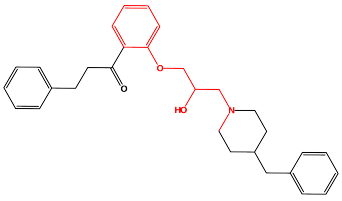
(21) GPV0062



(22) GPV0073

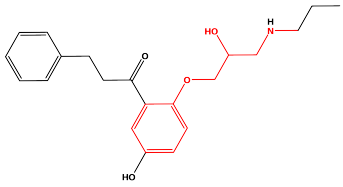


(23) GPV0088

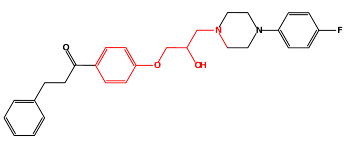


(24) GPV0128

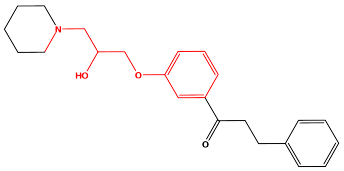
List of propafenone-like compounds in the test-dataset (cont.).



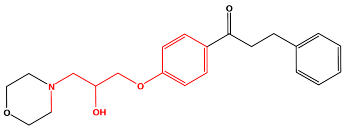
(25) GPV0129



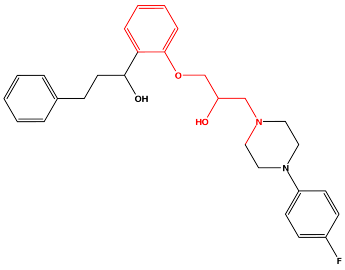
(26) GPV0134



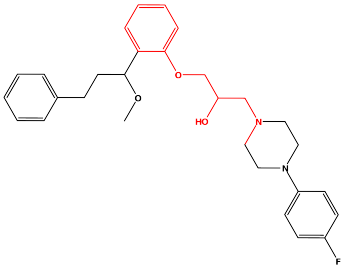
(27) GPV0135



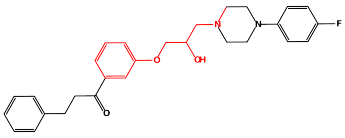
(28) GPV0149



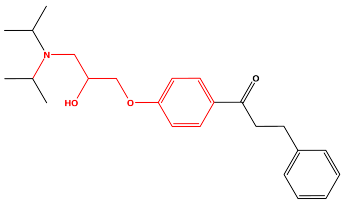
(29) GPV0155



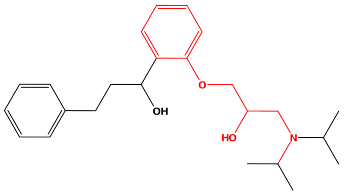
(30) GPV0156



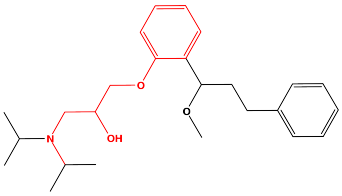
(31) GPV0157



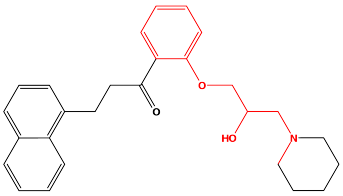
(32) GPV0159



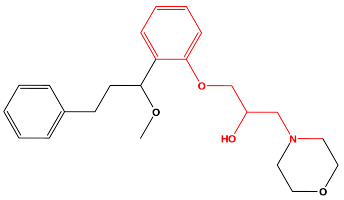
(33) GPV0163



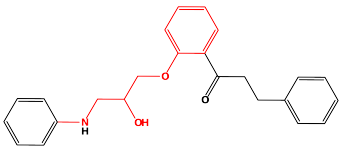
(34) GPV0164



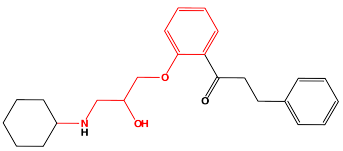
(35) GPV0180



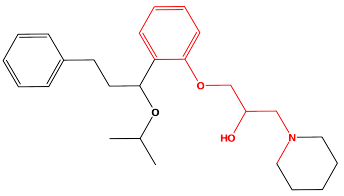
(36) GPV0227



(37) GPV0240

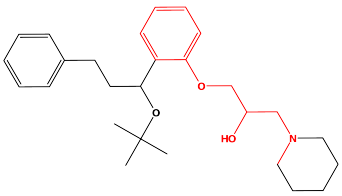


(38) GPV0242

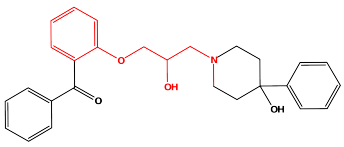


(39) GPV0264

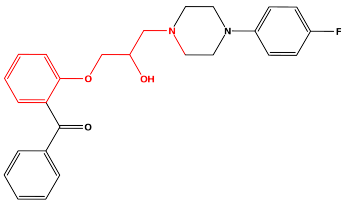
List of propafenone-like compounds in the test-dataset (cont.).



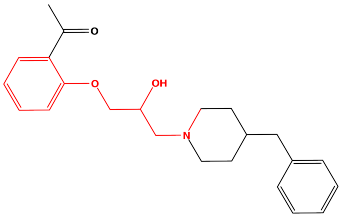
(40) GPV0265



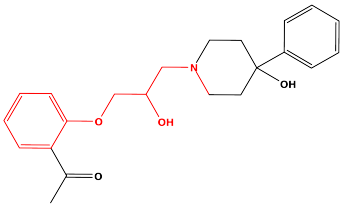
(41) GPV0317



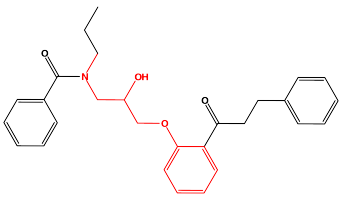
(42) GPV0319



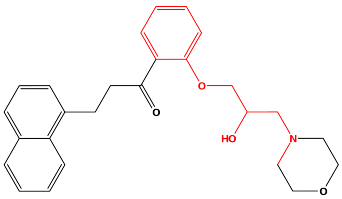
(43) GPV0321



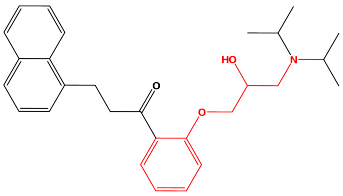
(44) GPV0323



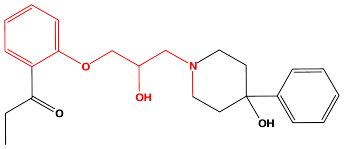
(45) GPV0366



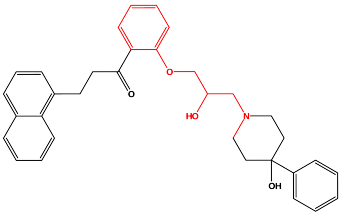
(46) GPV0374



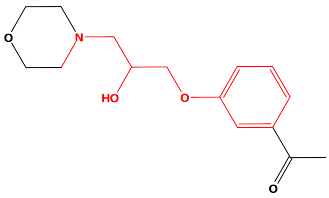
(47) GPV0376



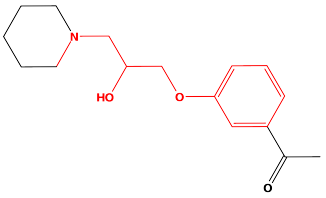
(48) GPV0381



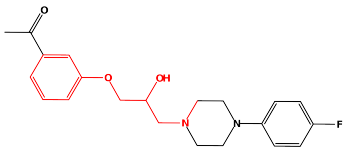
(49) GPV0382



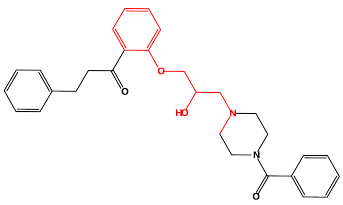
(50) GPV0384



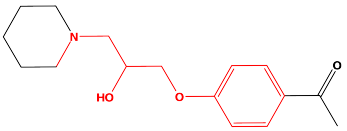
(51) GPV0385



(52) GPV0386

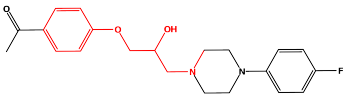


(53) GPV0388

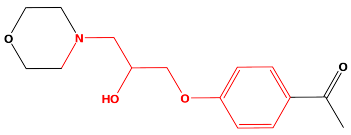


(54) GPV0389

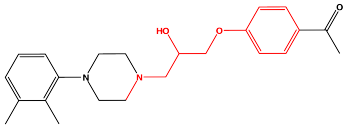
List of propafenone-like compounds in the test-dataset (cont.).



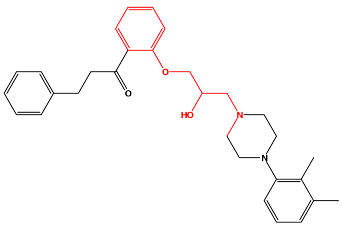
(55) GPV0390



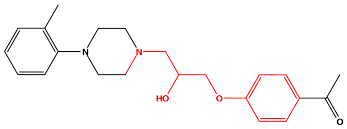
(56) GPV0391



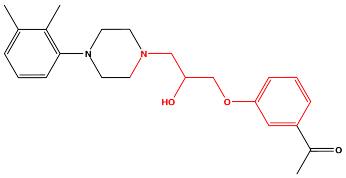
(57) GPV0574



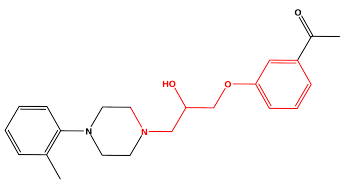
(58) GPV0576



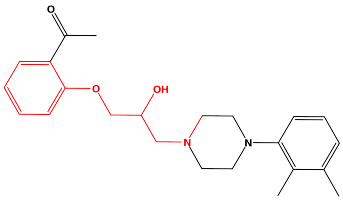
(59) GPV0577



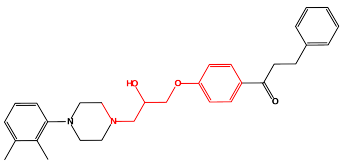
(60) GPV0579



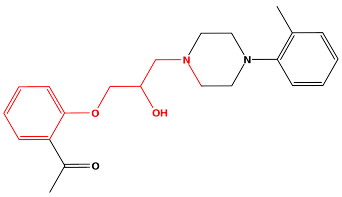
(61) GPV0596



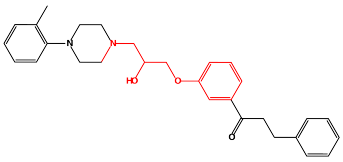
(62) GPV0598



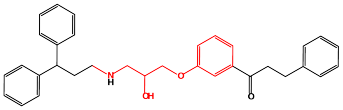
(63) GPV0600



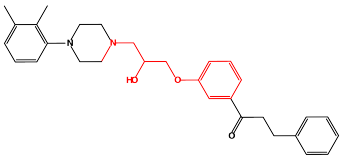
(64) GPV0626



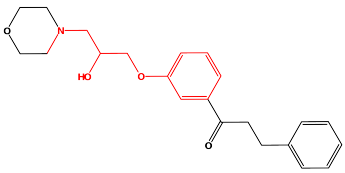
(65) GPV0647



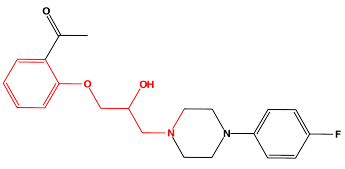
(66) GPV0649



(67) GPV0651

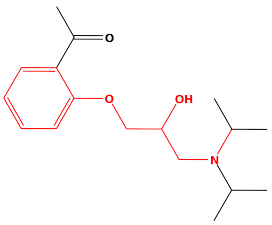


(68) GPV0653

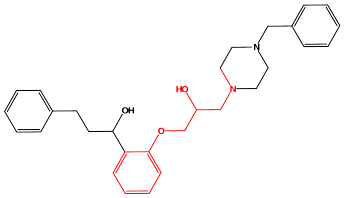


(69) GPV0787

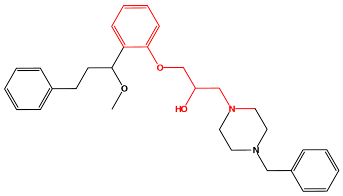
List of propafenone-like compounds in the test-dataset (cont.).



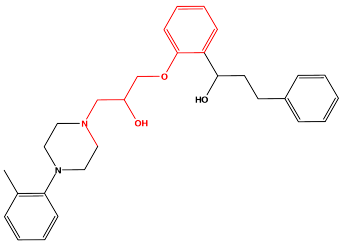
(70) GPV0788



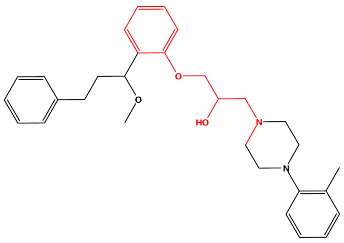
(71) GPV0789



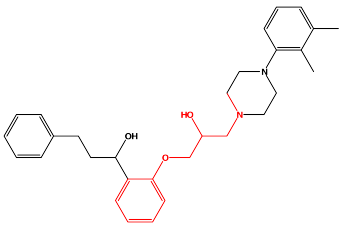
(72) GPV0790



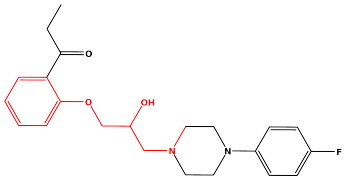
(73) GPV0791



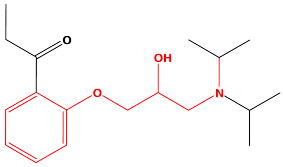
(74) GPV0792



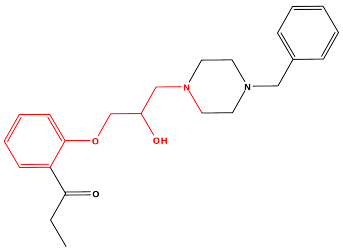
(75) GPV0793



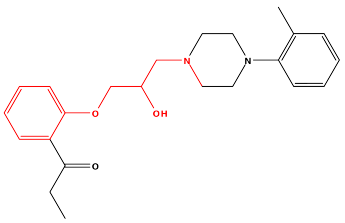
(76) GPV0794



(77) GPV0795



(78) GPV0796



(79) GPV0797

List of propafenone-like compounds in the test-dataset (cont.).

Bibliography

[1] C. Hansch, P. P. Maloney, T. Fujita, and R. M. Muir, "Correlation of Biological Activity of Phenoxyacetic Acids with Hammett Substituent Constants and Partition Coefficients," *Nature*, vol. 194, pp. 178–180, Apr 1962.

[2] R. Guha, "On the interpretation and interpretability of quantitative structure-activity relationship models.," *J Comput Aided Mol Des*, vol. 22, pp. 857–871, Dec 2008.

[3] R. A. Lewis, "A general method for exploiting QSAR models in lead optimization.," *J Med Chem*, vol. 48, pp. 1638–1648, Mar 2005.

[4] V. K. Gombar, "In silico Screens and Tools for Drug Discovery," in *In-Silico ADMET, Conference Documentation*, 2007.

[5] Chemical Computing Group Inc., Montreal, Canada, "Molecular Operating Environment (MOE) 2008.10," 2008.

[6] L. Salum and A. Andricopulo, "Fragment-based QSAR: perspectives in drug design.," *Mol Divers*, vol. 13, pp. 277–285, Aug 2009.

[7] Q.-S. Du, R.-B. Huang, Y.-T. Wei, Z.-W. Pang, L.-Q. Du, and K.-C. Chou, "Fragment-based quantitative structure-activity relationship (FB-QSAR) for fragment-based drug design.," *J Comput Chem*, vol. 30, pp. 295–304, Jan 2009.

[8] N. S. Zefirov and V. A. Palyulin, "Fragmental approach in QSPR.," *J Chem Inf Comput Sci*, vol. 42, no. 5, pp. 1112–1122, 2002.

[9] D. R. Lowis, "HQSAR: A New, Highly Predictive QSAR Technique," Tripos Technical Notes Volume 1 Number 5, Tripos, Inc., 1699 South Hanley Road, St. Louis, MO, USA, Oct 1997.

[10] Tripos™, Inc., "Sybyl 8.0," Nov 2007. Tripos, Inc., 1699 South Hanley Road, St. Louis, MO, USA.

[11] Tripos™, Inc., *HQSAR™ Manual*. Tripos, Inc., 1699 South Hanley Road, St. Louis, MO, USA, SYBYL 7.3 ed., 2006.

[12] W. Tong, D. R. Lowis, R. Perkins, Y. Chen, W. J. Welsh, D. W. Goddette, T. W. Heritage, and D. M. Sheehan, "Evaluation of quantitative structure-activity relationship methods for large-scale prediction of chemicals binding to the estrogen receptor.," *J Chem Inf Comput Sci*, vol. 38, pp. 669–677, Jul 1998.

[13] Chemical Computing Group, Inc., "SVL Exchange." http://www.chemcomp.com.

[14] C. Williams, *Reverse Fingerprinting: Mutual Information Based Activity Labeling and Scoring (MIBALS)*. Chemical Computing Group, Inc., 2008. Retrieved from http://www.chemcomp.com on Feb 12, 2009.

[15] Bibliographisches Institut & F. A. Brockhaus AG, "Brockhaus Enzyklopädie Online." http://www.brockhaus-enzyklopaedie.de/, 2009. Retrieved on April 2, 2009. In German.

[16] B. Buttingsrud, E. Ryeng, R. D. King, and B. K. Alsberg, "Representation of molecular structure using quantum topology with inductive logic programming in structure-activity relationships," *J Comput Aided Mol Des*, vol. 20, pp. 361–373, 2006.

[17] R. D. King, S. Muggleton, R. A. Lewis, and M. J. E. Sternberg, "Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase," *Proc Natl Acad Sci USA*, vol. 89, pp. 11322–11326, 1992.

[18] A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King, "Theories for mutagenicity: a study in first-order and feature-based induction," *Artificial Intelligence*, vol. 85, pp. 277–299, 1996.

[19] S. H. Muggleton and C. Feng, "Efficient induction of logic programs.," in *Proceedings of the 1st Conference on Algorithmic Learning Theory, Tokyo*, pp. 368–381, 1990.

[20] N. Marchand-Geneste, K. A. Watson, B. K. Alsberg, and R. D. King, "New approach to pharmacophore mapping and QSAR analysis using inductive logic programming. Application to thermolysin inhibitors and glycogen phosphorylase B inhibitors.," *J Med Chem*, vol. 45, pp. 399–409, Jan 2002.

[21] P. Gedeck and P. Willett, "Visual and computational analysis of structure-activity relationships in high-throughput screening data.," *Curr Opin Chem Biol*, vol. 5, pp. 389–395, Aug 2001.

[22] R. P. Sheridan and M. D. Miller, "A Method for Visualizing Recurrent Topological Substructures in Sets of Active Molecules," *J Chem Inf Comput Sci*, vol. 38, pp. 915–924, 1998.

[23] Tripos™, Inc., "Distill: Determine and Visualize Structure-Activity Relationships.," 1699 South Hanley Road, St. Louis, MO, USA, 2005. Distill information brochure. Retrieved from http://www.tripos.com/data/SYBYL/Distill_072505.pdf on May 19, 2009.

[24] Tripos™, Inc., *Distill™ Manual*. 1699 South Hanley Road, St. Louis, MO, USA, 2007.

[25] A. Schuffenhauer, P. Ertl, S. Roggo, S. Wetzel, M. A. Koch, and H. Waldmann, "The scaffold tree–visualization of the scaffold universe by hierarchical scaffold classification.," *J Chem Inf Model*, vol. 47, no. 1, pp. 47–58, 2007.

[26] A. M. Clark and P. Labute, "Detection and assignment of common scaffolds in project databases of lead molecules.," *J Med Chem*, vol. 52, pp. 469–483, Jan 2009.

[27] D. T. Stanton, "On the physical interpretation of QSAR models.," *J Chem Inf Comput Sci*, vol. 43, no. 5, pp. 1423–1433, 2003.

[28] Minitab, Inc., State College PA 16801-3008, USA, *Minitab® 15 Help*. Minitab Inc., 2007.

[29] D. G. Garson, *Statnotes: Topics in Multivariate Analysis*, ch. Partial Least Squares Regression (PLS). Published online, 2009. Retrieved from http://faculty.chass.ncsu.edu/garson/pa765/statnote.htm on May 19, 2009 .

[30] Minitab, Inc., State College PA 16801-3008, USA, "Minitab® 15," 2007.

[31] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *J Chem Inf Comput Sci*, vol. 28, pp. 31–36, May 1988.

[32] C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, and E. Willighagen, "The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics," *J Chem Inf Comput Sci*, vol. 43, pp. 493–500, Feb 2003.

[33] CDK Developers, "Chemical development kit online documentation." http://cdk.sourceforge.net. Retrieved on March 1, 2009.

[34] Chemical Computing Group Inc., Montreal, Canada, *Molecular Operating Environment (MOE) - Manual*, 2007.

[35] E. Estrada, G. Patlewicz, and Y. Gutierrez, "From knowledge generation to knowledge archive. A general strategy using TOPS-MODE with DEREK to formulate new alerts for skin sensitization.," *J Chem Inf Comput Sci*, vol. 44, no. 2, pp. 688–698, 2004.

[36] L. H. Hall and L. B. Kier, *Reviews in Computational Chemistry*, ch. 9: The Molecular Connectivity Chi Indexes and Kappa Shape Indexes in Structure-Property Modeling, pp. 367–422. VCH Publishers, Inc., 1991. Edited by K. B. Lipkowitz and D. B. Boyd.

[37] Chemical Computing Group Inc., Montreal, Canada, "Molecular Operating Environment (MOE) 2007.09," 2007.

[38] J. Gasteiger and M. Marsili, "Iterative Partial Equalization of Orbital Electronegativity - a Rapid Access to Atomic Charges," *Tetrahedron*, vol. 36, pp. 3219 –3228, 1980.

[39] R. Mannhold and K. Dross, "Calculation Procedures for Molecular Lipophilicity: a Comparative Study," *Quant Struct-Act Relat*, vol. 15, no. 5, pp. 403–409, 1996.

[40] S. A. Wildman and G. M. Crippen, "Prediction of Physicochemical Parameters by Atomic Contributions," *J Chem Inf Comput Sci*, vol. 39, no. 5, pp. 868–873, 1999.

[41] R. Mannhold, G. I. Poda, C. Ostermann, and I. V. Tetko, "Calculation of molecular lipophilicity: State-of-the-art and comparison of log p methods on more than 96,000 compounds.," *J Pharm Sci*, vol. 98, pp. 861–893, Mar 2009.

[42] T. J. Hou, K. Xia, W. Zhang, and X. J. Xu, "ADME evaluation in drug discovery. 4. Prediction of aqueous solubility based on atom contribution approach.," *J Chem Inf Comput Sci*, vol. 44, no. 1, pp. 266–275, 2004.

[43] P. Labute, "A widely applicable set of descriptors.," *J Mol Graph Model*, vol. 18, no. 4-5, pp. 464–477, 2000.

[44] T. A. Halgren, "Merck Molecular Force Field. I. Basis, Form, Scope, Parameterization, and Performance of MMFF94," *J Comput Chem*, vol. 17, no. 5–6, pp. 490–519, 1996.

[45] L. B. Kier and L. H. Hall, "An Electrotopological State Index for Atoms in Molecules," *Pharm Res*, vol. 7, no. 8, pp. 801–809, 1990.

[46] L. H. Hall and L. B. Kier, "Electrotopological State Indices for Atom Types: A Novel Combination of Electronic, Topological, and Valence State Information," *J Chem Inf Comput Sci*, vol. 35, pp. 1039–1045, 1995.

[47] P. Ertl, B. Rohde, and P. Selzer, "Fast calculation of molecular polar surface area as a sum of fragment-based contributions and its application to the prediction of drug transport properties.," *J Med Chem*, vol. 43, pp. 3714–3717, Oct 2000.

[48] Derwent Publications Ltd., "World Drug Index Database WDI97, distributed by Daylight Chemical Information Systems, Inc.," 1997.

[49] F. R. Burden, "Molecular Identification Number for Substructure Searches," *J Chem Inf Comput Sci*, vol. 29, no. 3, pp. 225–227, 1989.

[50] R. S. Pearlman and K. Smith, "Novel Software Tools for Chemical Diversity," *Perspectives in Drug Discovery and Design*, vol. 9,10,11, pp. 339–353, 1998.

[51] M. Petitjean, "Applications of the Radius-Diameter Diagram to the Classification of Topological and Geometrical Shapes of Chemical Compounds," *J Chem Inf Comput Sci*, vol. 32, no. 4, pp. 331–337, 1992.

[52] A. Balaban, "Highly Discriminating distance-based topological index.," *Chemical Physics Letters*, Jul 1982.

[53] T. I. Oprea, "Property distribution of drug-related chemical databases," *J Comput Aided Mol Des*, vol. 14, pp. 251–264, 2000.

[54] Chemical Computing Group Inc., Montreal, Canada, *Reverse Fingerprinting: Mutual Information Based Activity Labeling and Scoring (MIBALS)*.

[55] Persistence of Vision Raytracer Pty. Ltd., "POV-Ray™ for Windows." http://www.povray.org, 1991–2006.

[56] D. M. Bayada, R. W. Simpson, A. P. Johnson, and C. Laurenco, "An algorithm for the multiple common subgraph problem," *J Chem Inf Comput Sci*, vol. 32, no. 6, pp. 680–685, 1992.

[57] D. M. Bayada and A. P. Johnson, *Molecular Similarity and Reactivity: From Quantum Chemical to Phenomenological Approaches*, vol. 14 of *Understanding Chemical Reactivity*, ch. Detection of the Largest Patterns Common to a Set of Molecules, Using 2D Graphs and 3D Skeletons, pp. 243–265. Kluwer Academic Publishers, 1995. Edited by R. Carbó.

[58] Y. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki, "Recognition of Largest Common Structural Fragment among a Variety of Chemical Structures," *Analytical Sciences*, vol. 3, pp. 23–28, Feb 1987.

[59] T. H. Varkony, Y. Shiloach, and D. H. Smith, "Computer-Assisted Examination of Chemical Compounds for Structural Similarities," *J Chem Inf Comput Sci*, vol. 19, no. 2, pp. 104–111, 1979.

[60] H. Bunke and A. Kandel, "Mean and maximum common subgraph of two graphs," *Pattern Recognition Letters*, vol. 21, pp. 163–168, 2000.

[61] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *J Comput Aided Mol Des*, vol. 16, pp. 521–533, 2002.

[62] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, pp. 341–352, Dec 1972.

[63] J. J. McGregor and P. Willett, "Use of a maximum common subgraph algorithm in the automatic identification of ostensible bond changes occurring in chemical reactions," *J Chem Inf Comput Sci*, vol. 21, no. 3, pp. 137–140, 1981.

[64] Y. Cao, T. Jiang, and T. Girke, "A maximum common substructure-based algorithm for searching and predicting drug-like compounds," *Bioinformatics*, vol. 24, no. 13, pp. i366–i374, 2008. An open access journal. Retrieved on September 24, 2008.

[65] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs," *Theoretical Computer Science*, vol. 250, pp. 1–30, 2001.

[66] J. W. Raymond, E. J. Gardiner, and P. Willett, "RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[67] D. R. Wood, "An algorithm for finding a maximum clique in a graph," *Operations Research Letters*, vol. 21, pp. 211–217, 1997.

[68] L. G. Mitten, "Branch-And-Bound Methods: General Formulation and Properties," *Operations Research*, vol. 18, pp. 24–34, Jan–Feb 1970.

[69] J. W. Raymond, E. J. Gardiner, and P. Willett, "Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm.," *J Chem Inf Comput Sci*, vol. 42, no. 2, pp. 305–316, 2002.

[70] M. A. Johnson and G. M. Maggiora, eds., *Concepts and applications of molecular similarity.* A Wiley-Interscience publication., 1990.

[71] P. Willett, J. M. Barnard, and G. M. Downs, "Chemical similarity searching," *J Chem Inf Comput Sci*, vol. 38, pp. 983–996, 1998.

[72] J. Auer and J. Bajorath, *Bioinformatics, Volume II: Structure, Function and Applications*, vol. 453, ch. Molecular Similarity Concepts and Search Calculations, pp. 327–347. Humana Press, a part of Springer Science+Business Media, 2008. Edited by J. M. Keith.

[73] Tripos™, Inc., *Graphics Manual*. Tripos, Inc., 1699 South Hanley Road, St. Louis, MO, USA, SYBYL 8.0 ed., 2007.

[74] J. Goodman, *Principles of Scientific Computing*. Published online on http://www.math.nyu.edu/faculty/shelley/Classes/SciComp/GoodmanBook.pdf, 2006. Retrieved on April 14, 2009.

[75] ChemDiv, Inc.

[76] R. A. Pearlstein, R. J. Vaz, J. Kang, X.-L. Chen, M. Preobrazhenskaya, A. E. Shchekotikhin, A. M. Korolev, L. N. Lysenkova, O. V. Miroshnikova, J. Hendrix, and D. Rampe, "Characterization of HERG potassium channel inhibition using CoMSiA 3D QSAR and homology modeling approaches.," *Bioorg Med Chem Lett*, vol. 13, pp. 1829–1835, May 2003.

[77] R. L. Juliano and V. Ling, "A surface glycoprotein modulating drug permeability in Chinese hamster ovary cell mutants," *Biochimica et Biophysica Acta (BBA) - Biomembranes*, vol. 455, pp. 152–162, Nov 1976.

[78] T. Tsuruo, H. Iida, S. Tsukagoshi, and Y. Sakurai, "Overcoming of Vincristine Resistance in P388 Leukemia *in Vivo* and *in Vitro* through Enhanced Cytotoxicity of Vincristine and Vinblastine by Verapamil," *Cancer Research*, vol. 41, pp. 1967–1972, May 1981.

[79] M. M. Gottesmann, T. Fojo, and S. E. Bates, "Multidrug Resistance in Cancer: Role of ATP-Dependent Transporters," *Nature Reviews in Cancer*, vol. 2, pp. 48–58, Jan 2002.

[80] G. F. Ecker and P. Chiba, "Development of modulators of multidrug resistance: A pharmacoinformatic approach," *Pure Appl. Chem.*, vol. 76, no. 5, pp. 997–1005, 2004.

[81] G. F. Ecker, T. Stockner, and P. Chiba, "Computational models for prediction of interactions with ABC-transporters," *Drug Discovery Today*, vol. 13, pp. 311–317, Apr 2008.

[82] C. A. McDevitt and R. Callaghan, "How can we best use structural information on P-glycoprotein to design inhibitors?," *Pharmacology & Therapeutics*, vol. 113, pp. 429–441, 2007.

[83] R. W. Robey, S. Shukla, E. M. Finley, R. K. Oldham, D. Barnett, S. V. Ambudkar, T. Fojo, and S. E. Bates, "Inhibition of P-glycoprotein (ABCB1)- and multidrug resistance-associated protein (ABCC1)-mediated transport by the orally administered inhibitor, CBT-1®," *Biochemical Pharmacology*, vol. 75, pp. 1302–1312, 2008.

[84] P. Chiba, S. Burghofer, E. Richter, B. Tell, A. Moser, and G. F. Ecker, "Synthesis, Pharmacologic Activity, and Structure-Activity Relationships of a Series of Propafenone-Related Modulators of Multidrug Resistance," *J Med Chem*, vol. 38, no. 14, pp. 2789–2793, 1995.

[85] G. Klopman, L. M. Shi, and A. Ramu, "Quantitative Structure-Activity Relationship of Multidrug Resistance Reversal Agents," *Molecular Pharmacology*, vol. 52, pp. 323–334, 1997.

[86] G. F. Ecker and P. Chiba, *Computational Toxicology: Risk Assessment for Pharmaceutical and Environmental Chemicals*, ch. QSAR Studies on Drug Transporters Involved in Toxicology, pp. 295–314. John Wiley & Sons, Inc., 2007. Edited by S. Ekins.

[87] P. Chiba, G. F. Ecker, D. Schmid, J. Drach, B. Tell, S. Goldenberg, and V. Gekeler, "Structural Requirements for Activity of Propafenone-type Modulators in P-Glycoprotein-Mediated Multidrug Resistance," *Molecular Pharmacology*, vol. 49, pp. 1122–1130, 1996.

[88] G. Ecker, M. Huber, D. Schmid, and P. Chiba, "The importance of a nitrogen atom in modulators of multidrug resistance.," *Mol Pharmacol*, vol. 56, pp. 791–796, Oct 1999.

[89] J. Schwöbel, R.-U. Ebert, R. Kühne, and G. Schüürmann, "Prediction of the Intrinsic Hydrogen Bond Acceptor Strength of Organic Compounds by Local Molecular Parameters," *J Chem Inf Model*, vol. 49, pp. 956–962, Apr 2009.

C. Abstract

QSAR or QSPR equations are mainly intended to predict physical properties or biological activity of chemical compounds, but they also contain further information which supports unveiling the underlying SAR of an equation, thus giving reasons for one molecule being active and a second not.

Many approaches aiming at an extraction of substructural information out of model-based QSARs, created by various methods, have already been introduced, among them RQSPR by Gombar, which is included in the "visdom" tool set. It projects back contribution values to a molecule's atoms. Inspired by this workflow, we present a new application which allows a user to visualize contributions of a molecule's atoms to a predefined PLS-QSAR equation. Implemented in MOE (Molecular Operating Environment) SVL scripting language, it can both use MOE's display options and MOE's integrated window toolkit, which enables a user-friendly GUI application. The script's backprojecting capability is restricted to so-called fragment based descriptors, thus descriptors calculated from values assigned to submolecular parts. The fragment size has to be one atom, nonetheless an amazing 57% of all MOE implemented 2D-descriptors are available, as well as the E-state descriptor set by Hall and Kier.

An MLCS (multiple largest common substructure) feature highlighting atoms and bonds common to all molecules in the dataset, various display options and a special multiple-view mode additionally improve the visualization and make it highly customizable.

A proof-of-concept study using a QSAR dataset of 79 propafenone-like ABCB1 inhibitors demonstrates the script's basic appliance. It can be shown that some well-known structural features of the propafenones contributing to activity are preserved in the examplary PLS-QSAR equation, e.g. lipophilicity or π-π interactions. For other moieties already identified by multiple SAR studies on propafenones, e.g. hydrogen bond acceptor capabilites, results are ambiguous.

D. Zusammenfassung

QSAR- und QSPR-Modelle werden vor allem zur Vorhersage von Moleküleigenschaften oder von biologischer Aktivität verwendet. Oft enthalten sie zusätzliche Informationen über die zugrundeliegende Struktur-Aktivitätsbeziehung und unterstützen deren Aufklärung.

Eine der vielen publizierten Methoden, die einen Einblick in substrukturelle Information aus modellbasierten QSAR-Systemen erlauben, ist RQSPR von Gombar, das im Programm "visdom" enthalten ist. Es projiziert die Beiträge einzelner Atome eines Moleküls zur QSAR-Gleichung graphisch zurück auf das jeweilige Atom. Dadurch inspiriert, stellt diese Arbeit ein neues Programm vor, das die Visualisierung von Atombeiträgen anhand einer vordefinierten PLS-QSAR Gleichung erlaubt. Dank der Implementierung in SVL, der Skriptsprache von MOE (Molecular Operating Environment), können sowohl die graphischen Möglichkeiten als auch der Window Tool Kit der Applikation genutzt werden, wodurch eine anwenderfreundliche graphische Benutzerschnittstelle (GUI, graphical user interface) ermöglicht wird. Allerdings ist der Einsatz des Skripts auf sogenannte fragmentbasierte Deskriptoren beschränkt. Deren Gesamtwert setzt sich aus mehreren Einzelwerten, die jeweils einen Teil des Moleküls, ein Fragment, beschreiben, zusammen. In unserem Spezialfall muss die Fragmentgröße genau ein Atom betragen, trotzdem können rund 57% aller in MOE verfügbaren 2D-Deskriptoren eingesetzt werden. Zusätzlich sind die E-State-Deskriptoren von Hall und Kier im Skript implementiert.

Um die graphische Darstellung weiter zu verbessern, existiert ein MLCS-Modul (Multiple Largest Common Substructure), das Atome und Bindungen, die alle Moleküle im Datensatz besitzen, hervorheben kann. Weiters sind verschiedene Visualisierungsoptionen und eine spezielle Ansicht verfügbar, die bis zu vier Moleküle gleichzeitig darstellt.

In einer Proof-Of-Concept-Studie mit einem Datensatz aus 79 Propafenonen, die ABCB1 inhibieren, werden Anwendungsmöglichkeiten demonstriert. Einige in Vorarbeiten aufgezeigten Eigenschaften der Propafenone, die für die Aktivität wichtig sind, finden in die Gleichung Eingang, etwa Lipophilie oder π-π-Interaktionen. Andere Muster, etwa die Bedeutung von Wasserstoffakzeptoren, werden hingegen nicht repräsentiert.

E. Curriculum Vitae

Christoph Waglechner

Personal Details

Address	Eduard-Huebmerstraße 164 2823 Pitten Austria cwaglechner@gmx.at
Gender	Male
Date of birth	1984-06-03, Vienna
Citizenship	Austrian

Education

09/1990–06/1994	Elementary School in Pitten, VS Pitten
09/1994–06/2002	Secondary School in Wiener Neustadt, BG Babenbergerring
10/2002–12/2009	"Diplomstudium" (roughly comparable with Master) in Pharmacy at the University of Vienna, special focus on Pharmacoinformatics
03/2003–	Undergraduate Studies in Information Systems ("Wirtschaftsinformatik") at the Vienna University of Economics and Business, special focus on "New Media" and "E-Commerce" I intend to complete the BSc in 02/2010.

Scientific Experience

Poster presentation at the ISMC 2008 in Vienna

Poster presentation at the MOE European User Group Meeting in Hinxton Hall 2008

Military Service

07/2002–02/2003 SanA MilKdo B in Baden.

Language Knowledge

German: native

English: fluent

Working Experience

03/2003–09/2008 Unskilled worker at the “Apotheke zum Hl. Georg, Pitten”

03/2007–07/2009 Student’s support worker (“Tutor”) in the “PR Arzneimittelanalytik und Wirkstoffentwicklung”, University of Vienna

04/2008–09/2009 Non-scientific project staff of the EUROPIN-project (webmaster)

10/2008– Unskilled worker at the “Apotheke Bad Erlach”

Miscellaneous

12/2002– Voluntary service at the Austrian Red Cross in Neunkirchen, currently as “Notfallsanitäter NKV” (Austrian type of paramedic).

November 10, 2009