Simulation of prebiotic chemistries

Daniel Högerl

June 2010

'More than 30 years of experimentation on the origin of life in the fields of chemical and molecular evolution have led to a better perception of the immensity of the problem of the origin of life on Earth rather than to its solution. At present all discussions on principal theories and experiments in the field end either in stalemate or in a confession of ignorance'

Klaus Dose

In the presented work I would like to thank everyone contributing to my successfully finishing this diploma thesis:

My supervisory tutor Christoph Flamm (University of Vienna), Martin Mann (Albert-Ludwigs University of Freiburg) for providing the GGL [10], a C++ graph grammar library, Linda Petzold and group (University of California, Santa Barbara) for providing StochKit [24], a stochastic toolkit written in C++, Sebastian Sauer (University of Vienna) for supporting me in terms of quantum mechanical calculation tools, and my colleague Heinz Ekker for Perl libraries to automatically identify the atom-atom mapping in chemical reactions.

Abstract

Chemical reaction networks, i.e. a set of chemical compounds and a system of reactions that inter-convert them, are a key concept in many research areas of Chemistry or Biology. They have successfully been applied to study such diverse systems as combustion and petro-chemical processes, atmosphere chemistry, metabolic networks or the possible chemical routes to the first life forms.

The objective of this diploma thesis was to set up a rule-based simulation platform to explore the topological features and the dynamic behavior of chemical reaction networks that unfold in time. To achieve this goal an algebraic chemistry framework, which models molecules as labeled graphs and reactions as graph rewrite rules, was combined with a method for the on-the fly calculation of physicochemical properties of the molecules. This strategy allows on demand computation of e.g. reaction rates for novel chemical compounds that emerge during the simulation.

The chemical reaction networks are assembled in an incremental way starting from a set of molecules and a set of reactions. To avoid combinatorial explosion in terms of network size each expansion step is followed by a shrinking step. For the pruning step a kinetic filtering strategy based on Gillespie type stochastic simulation of the network was developed.

The computational framework was applied to study (i) a network with only a single reaction rule, i.e. the Diels-Alder reaction, that shows rapid combinatorial explosion and (ii) the Formose reaction network, an auto-catalytic reaction network producing sugars from formaldehyde.

Keywords: Algebraic Chemistry, Graph grammar, Rule-based Simulation, Gillespie Algorithm, SSA, Reaction Networks, Prebiotic Chemistry, Concentration Sampling Network Generator, Reaction Rate Calculation

Contents

1. Introduction

1.1. Status quo

Computer simulations have undeniably become important instruments to help scientists and engineers implement their ideas and proof theories. Though the computer is a mere appendage in cases of practical work it is getting much higher attention in science especially theoretical fields. An obvious reason is for instance the time and cost saving run of a simulation compared to providing a laboratory with test equipment and samples and the time and manpower to bring it to expected results.

Working in the field of theoretical chemistry we have a special interest in computationally predicting chemical behavior as found in single molecules or more often in chemical reactions being nothing else than the interaction between molecules. In fact there are simulation programs available for certain applications [14, 29, 30]. However, they all struggle with the complexity of the systems and processes which are intrinsic to even primitive biological conditions. Although individual aspects like thermodynamics or spatial organization and partial components are already taken into account by own specific models the coherent view onto the whole is still not the scope of any approach. Nevertheless these attempts have successfully shown their qualification when asking for modeled properties that can be decoupled from physical detail. This means that such approaches are absolutely sufficient as long as the embedded rules of the models do not require detailed knowledge of the underlying chemistry.

Though if we now want to take a look at evolutionary aspects of molecule networks or metabolic pathways we assume that the chemistry becomes an inevitably important part of the describing computer model. Thus it needs to incorporate chemical ground rules and constraints like conservation of mass and conservation or dissipation of energy respectively. Of course this does not mean we want all facets of chemistry to be included in a computer model as it is not yet feasible to simulate nature in all its details. Nevertheless we doubtlessly need to embark on a new strategy for modeling artificial chemistries.

1.2. Motivation

So why are we interested in evolutionary aspects of chemistry? – We do not want to argue the inherent curiosity of humans. Rather, there were many attempts in the past searching for the origin of organic chemistry and furthermore searching for the origin of life on earth. All these experiments have in common that conditions, or at least parts of it, are reproduced which might have prevailed on the early earth and in its atmosphere respectively. In this simulated world small molecules are provided in order to react with each other and with the environment. The output of such an experiment is then filtered for molecules which are known of playing important roles in organic life on earth, like for instance amino acids, carboxylic acids, purines and sugars. Also of great importance are groups of molecules forming cycles with respect to their chemical relationship, i.e. have the capability to establish self-replicating cycles.

Nonetheless, there are of course crucial differences between all the experiments. The differences are the start set of molecules and the theories to be proven. The molecules at the beginning of the scenarios are all small and not complex so it is very likely that they were present at that time in the given environment. Coming to the various theories we also have to mention the milieu every theory proposes. Though Miller and Urey [1] experimented with inorganic molecules in a reduced hot atmosphere (figure 1), Wächtershäuser suggested in his iron-sulphur world hypotheses [2] that prebiotic synthesis of organic molecules could have taken place on the surface of iron-sulphur minerals. His theory also fits to geological observations made in the near of deep-sea volcanoes and black smokers, the habitats for some sorts of extremophiles. Furthermore Wächtershäuser's theory covers proposals for the polymerization processes of prebiotic molecules.

Polymerization of molecules is considered to be important in prebiotic chemistry as many stakeholders of present biological systems are macromolecules. Proteins and nucleic acids for instance are formed by polymerization and condensation of smaller molecules like amino acids and nucleotides. The polymerization process consumes energy and as ATP (adenosine triphosphate, the common energy transfer unit in biochemistry) is produced only by enzymes it could not be in charge of the prebiotic energy transfer. In the iron-sulphur world hypotheses anaerobic redox reactions reliably provide energy.

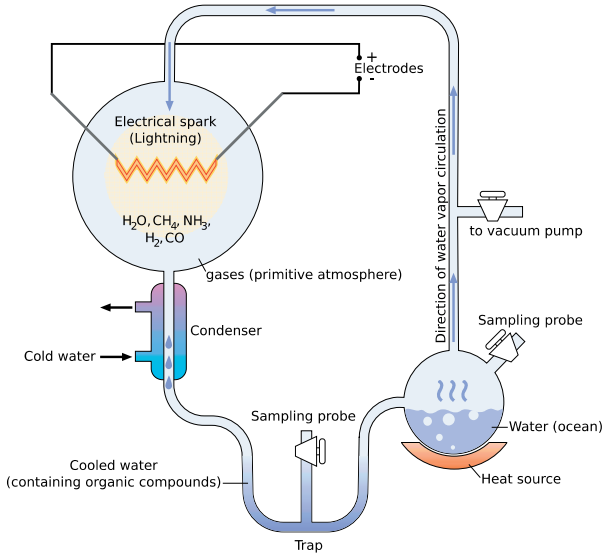


Figure 1: Configuration of the famous Miller-Urey experiment. Conditions of the early earth are simulated by a source of hot water, a primitive atmosphere containing gases supposed to having been present and indispensable for an emerging prebiotic chemistry. Energy is brought into the system by artificially generated electrical sparks.

Another discussion is present these days whether the "origin of life" follows suggestions of the RNA world (also known as "Replication first") hypotheses or the "Metabolism

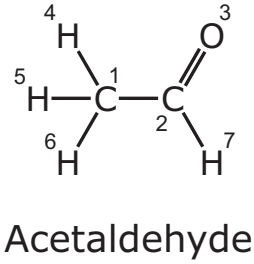
first" hypotheses[1]. The latter proposes emerging metabolic cycles like for example the (reductive) citric acid cycle to mark the early beginnings of life and says that metabolism is more likely to establish than forming more complex RNA molecules as suggested in the former hypotheses. In contrast the RNA world hypotheses claims replication and storage of information to be life's origin whereas experiments showed that already small RNA molecules and ribozymes (the RNA pendant of enzymes) can store and replicate (genetic) information.

Our objective now is to computationally view chemical aspects of the mentioned types of theories and hypotheses to be able to either resume some discussions and stop others or to give new and probably unconsidered input to established discussions.

1.3. Solution statement

Below we describe a computational approach that covers the above mentioned scenarios of chemical reaction networks and related proposals.

The core idea of our approach is the transformation of the chemical problem into a graph-based artificial chemistry. In this toy universe (this term was introduced by [6]) molecules are modeled as mathematical undirected graphs, comparable with adjacency matrices in which atoms are labeled vertices and bonds between atoms are labeled edges connecting two vertices.



Acetaldehyde

	1	2	3	4	5	6	7
1	0	1	0	1	1	1	0
2	1	0	2	0	0	0	1
3	0	2	0	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	0	0	0	0	0
6	1	0	0	0	0	0	0
7	0	1	0	0	0	0	0

Figure 2: Numbered molecule graph and an adjacency matrix like connection table of acetaldehyde (systematic ethanal). The matrix is redundant, which means mirrored on the main diagonal, and not zero compressed.

Furthermore chemical reactions are modeled as graph rewrite rules [7] (an example is given in figure 3). A graph rewrite rule plays the role of the reaction in the sense of taking one or more molecules as input and transforming this input into one or more output molecules. The inputs for such a rewrite rule are the educts of the chemical reaction as well as environmental conditions like catalysts. The output, however, represents the

[1] A short summary can be found in [3].

products of the chemical reaction. The process of the chemical reaction itself is now modeled by applying the graph rewrite rule to the given educt molecules.

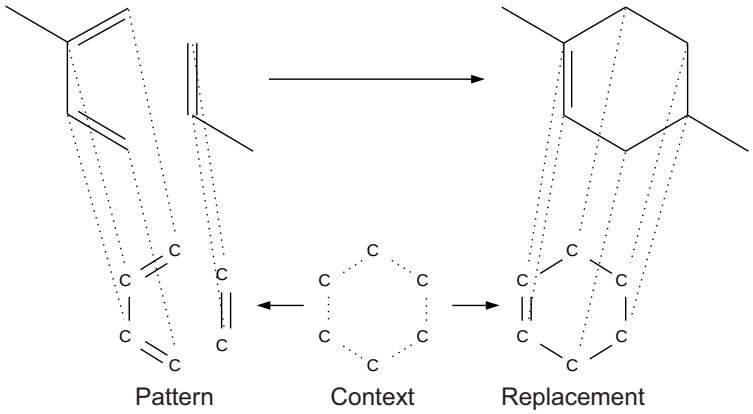


Figure 3: Example of a Diels-Alder reaction at the top of the figure and the corresponding graph rewrite rule in the bottom.

Since single chemical reactions and hence single graph rewrite applications are only of limited interest with respect to our intention we expand this system to an extent so that no longer single reactions but whole reaction networks are modeled. This is done by unspecific application of all available graph rewrite rules to all available molecule graphs. We call this step "network expansion" as the network, consisting of molecule species and reactions between all known molecules, is expanded by such a step – new reactions and also new molecule species are formed and added to the network.

The described network expansion alone would induce a combinatorial explosion in terms of network size in more or less short time, depending on the initial set of molecules and on the diversity and number of graph rewrite rules, as observed by [4, 8] and [9]. Thus to avoid combinatorial explosion in our network generator we apply a network shrinking step after each expansion step according to [4]. As pruning strategy we suggest the simulation of the reaction kinetics in the chemical reaction network because this best reflects the evolutionary aspects of the distribution of species concentrations in molecule networks. Subsequently molecule species with the lowest concentration can be removed from the network.

For simulating the chemical kinetics of the reaction network a stochastic simulation method, the Monte Carlo type Stochastic Simulation Algorithm SSA, introduced by Daniel T. Gillespie [5] is used. Therefore our reaction system is assumed to be well-stirred and the dynamical behavior of its molecules shall exhibit some degree of randomness.

As a prerequisite for simulating the concentration distribution of the network molecules the reaction rates of the chemical reactions in the network need to be known. Since real reaction rates can not yet be calculated *ab initio*, but only measured in physical experiments, we use a way of consistently estimating the rates by using quantum mechanical calculating programs[1].

[1]See appendix D for details on the used quantum mechanical software package.

1.4. Applications

Like mentioned above chemical reaction networks can be found in various fields of nature. Similarly applications in those fields are present and could need computational assistance. This assistance accounts for simulating evolving networks whereas the initial set of molecule species and reactions and constraints have been transformed into a structure our approach or rather formalism can handle them.

Below examples for various related fields are listed:

- Biochemistry
  - Metabolic pathways
  - Enzyme-catalyzed reactions
  - Gene transcription
- Petro Chemistry
  - Petro chemical refining
  - Combustion reactions
  - Polymerization reactions
- Atmospheric Chemistry
  - Gas phase reactions
  - Photochemistry
- Cell Biology
  - Signal transduction
  - Auto-regulation
  - Metabolic networks

2. Theory

In the following sections we outline the theories our presented thesis relies on. Starting with basics of Cheminformatics and a general theory of chemical reaction networks we go further into detail of graph theory, stochastic simulation and chemical kinetics.

2.1. Cheminformatics

2.1.1. Introduction

Cheminformatics or Chemoinformatics is defined [34] as the use of computer and informational techniques, applied to a range of problems in the field of chemistry. These techniques are furthermore used for calculating, predicting and changing molecular properties computationally and are even used in the process of drug discovery in pharmaceutical chemistry for instance.

As the word Cheminformatics implies it combines the scientific working fields of chemistry and computer science and is nowadays often understood as a branch of theoretical chemistry. Although Cheminformatics is a relative young term its basic ideas and concepts are nonetheless known for many decades now, however under other titles such as computational chemistry or chemical graph theory. Therefore the latter is still one of the main topics Cheminformatics deals with. Furthermore it helps establishing and extending the chemical space. The Chemical space can be thought of all known and possible molecules spanning a space in which chemical reactions allow us to move among these molecules. By extending the scope of Cheminformatics so that the interest in even bigger molecules like nucleic acids and proteins becomes feasible another scientific subject was established known as Bioinformatics.

2.1.2. Applications

In the following sections we describe prominent applications of Cheminformatics and Bioinformatics.

Storage and retrieval

One primary application of Cheminformatics is the storage and processing of information relating to chemical compounds. In this sense chemical databases have been established, specifically designed to store chemical information. This information covers chemical and crystal structures, spectra of molecules or atoms, reactions and syntheses and of course thermo physical data. For the preceding retrieval of all kinds of storable information more or less complex experiments including various measuring instruments are set up.

The retrieval from the Cheminformatics point of view starts with reading the raw data from the experiments instruments. The used instruments and the retrieving software either need to comply with the same defined interfaces. This is required in order to enable data exchange in possibly both directions and interpreting the data in a defined way. The retrieved data is then processed to extract the needed information. Some

properties, such as the inter atomic distance, can be directly accessed from the raw data and potentially need to be converted into another value range. Other properties, however, have to be calculated from measurements using specific models. When the required chemical attributes are available the data can be stored finally.

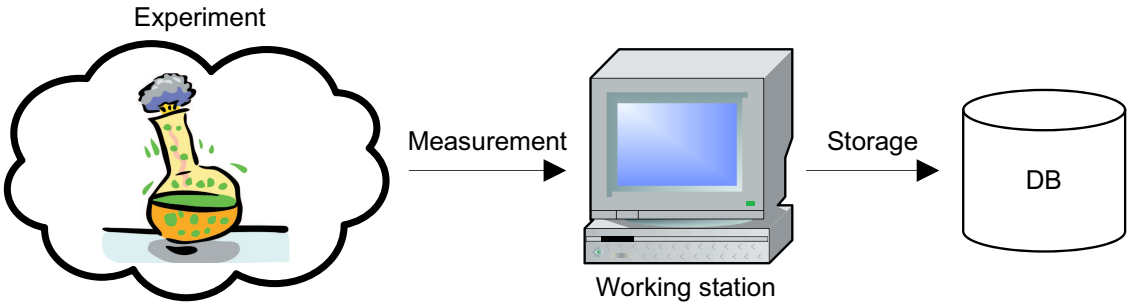


Figure 4: Schema of a chemical experiment depicting the way from the actual experiment towards the measurements and finally to the data storage.

When storing the chemical data certain considerations have to be made. The first question about what parts of the chemical information should be stored was handled by the previous steps of selection, conversion and extraction. We only process the attributes we really are interested in, which is why our first question has already been answered. Another question is what format the data should be saved as. In practice two main concepts have proven true. On the one hand relational databases, mainly MySQL and Oracle, are extensively used in organizations and companies to store such kind and amount of data. On the other hand there are still applications and databases in the fields of Cheminformatics and Bioinformatics that use a flat file format, for instance the FASTA database in Bioinformatics. Nevertheless applications encapsulate those storage formats so that the relational access semantics are the same for both underlying worlds. More technical detail is demanded to clarify the required value ranges or reserved sizes for the attributes to store. In the following enumeration existing data formats are listed.

- **Crystallographic databases** store x-ray crystal structure data. Examples are Protein Data Bank PDB and Cambridge Structural Database.
- **NMR spectra databases** correlate chemical structure with NMR (Nuclear Magnetic Resonance) data. Often other characterization data such as infrared spectroscopy or mass spectrometry is also included.
- **Reactions databases** contain intermediates and temporarily created unstable molecules in addition to the reactions participants in contrast to ordinary chemical databases. This information allows us to make statements about the reaction mechanisms.

- **Thermo physical databases** contain all kinds of thermo physical information such as equilibria, solubility, heats of vaporization or formation, viscosity or thermal conductivity.

Intersections with the above mentioned topics can already be found in general computer science and are reported as data mining and machine learning[1].

Virtual libraries

In the Cheminformatics viewpoint libraries are libraries of chemical compounds and classes of compounds respectively existing in reality. Virtual libraries, however, extend such libraries by computationally generated compounds that are similar to already observed ones with respect to particular properties. Chemical libraries are used to represent and explore the chemical space as mentioned earlier.

One application concerning virtual libraries is the search for novel compounds with desired properties. This intention is also called the design of new chemical compounds. Known molecules with at least partly favored properties are considered as input for the design of a new molecule combining their properties. The undeniable objective is of course the enhancement of naturally existing compounds and their combination in order to fulfill at least the same demands as met before with less (chemical) material though.

For the design of novel compounds the following is needed: information about existing molecules retrieved from physical/chemical experiments, information about correlating properties like conformation and active centers or spatial placement of atoms and the molecules hydrophobicity and finally algorithms qualified to generate chemically reasonable molecules. The second requirement, the property correlations, is in large part handled by looking for statistical correlations between properties in databases of chemical compounds. Identified correlations are then tested and confirmed statistically and furthermore for instance by verifying predictions with experiments or simulations. The third demand, the generation of novel compounds, these days is meet by several heuristics and approximations to implement some kind of chemical *in silico* process of forming molecules. These methods reach from artificial intelligence in form of Markov chains applied to compounds training databases to derive new molecules to so called *ab initio* procedures in which quantum mechanics manage such calculations without empirical parameters.

Virtual libraries in Cheminformatics are not only used for designing novel molecules. In some of the major applications virtual libraries act as sub unit within larger use cases. Besides the search for chemical compounds with desired properties in these libraries also the molecular docking (e.g. in protein-ligand interaction) is based on their information and furthermore used to identify relationships between molecules computationally and therefore possibly with high throughput. Another application of virtual libraries is given in the form of virtual screening which is a technique comparable to molecule docking but more specialized and therefore treated in more detail in the following section.

[1]Data mining is the extraction of patterns from data and machine learning derives algorithmic behaviors based on empirical data.

Virtual screening

Where the previous sections described the libraries or data storage of chemical compounds and their features in this section the actual methods for investigating those libraries and their chemical space is outlined. Virtual screening is a computational technique used in drug discovery research[43]. In general virtual screening is used to identify molecules likely to possess desired properties such as biological activity against a given target molecule. In this context virtual screening is one main part in the drug discovery process and has therefore become an integral part of it.

To find now molecules with desired properties or compare molecules with respect to these properties respectively two main screening methods have been established: the ligand-based and the structure-based screening.

Ligand-based

Ligand-based screening requires a set of structurally different ligand molecules binding to one receptor. Based on the information about the ligands a model of the receptor can then be built. Such models are called pharmacophore models derived from the term pharmacophore introduced by Paul Ehrlich in 1909. A pharmacophore consists only of the features and properties of a molecule that have a pharmaceutical effect. A candidate ligand can then be compared to the pharmacophore model to determine its compatibility with the model and furthermore how likely it is to bind to the receptor. Besides the pharmacophore model approach there is also a way to scan a database of molecules against one active ligand by applying chemical similarity analysis methods.

Structure-based

Structure-based virtual screening utilizes molecular docking of candidate ligands into a target molecule. Afterwards a scoring function is applied to estimate the likelihood that the ligand will bind to the target.

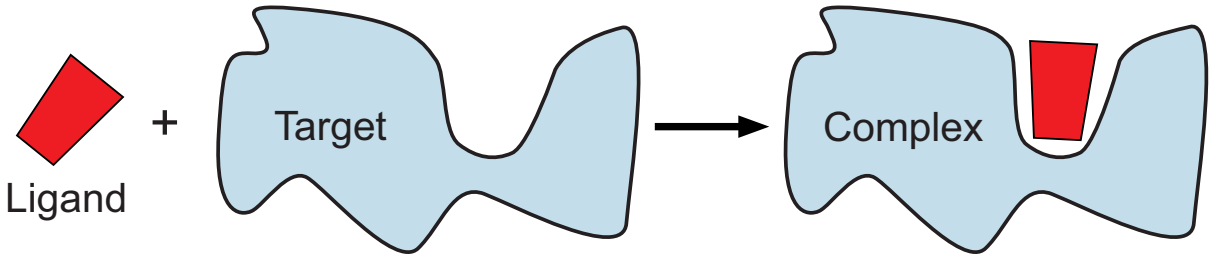


Figure 5: Target and ligand molecules forming a complex. The depicted molecular docking also considers the conformational changes of the involved molecules, also known as *induced fit*.

The prerequisite for the operation of many virtual screening programs is the computation of pair-wise interactions between atoms. As the latter is of $\mathcal{O}(N^2)$ computational complexity where N is the number of atoms in the system the necessary computing in-

frastructure varies depending on the method and the problem size. Ligand-based methods typically can be performed on single workstation CPUs even for large screenings. A structure-based screening, however, requires a parallel computing infrastructure such as a computer cluster running a batch queue processor to handle the work. Furthermore also the processed libraries or compound databases need to support being queried by the parallel cluster.

Protein-ligand docking

Protein-ligand docking is a molecular modeling technique with the goals: prediction if a ligand binds to a protein receptor or enzyme, and if so what is the spatial orientation of the components and the three-dimensional conformation of the resulting complex. Furthermore the strength of the eventual binding, i.e. the binding affinity, should be determined to allow the comparison of diverse ligand molecules. To summarize the single central question is: *How do two molecules interact?*

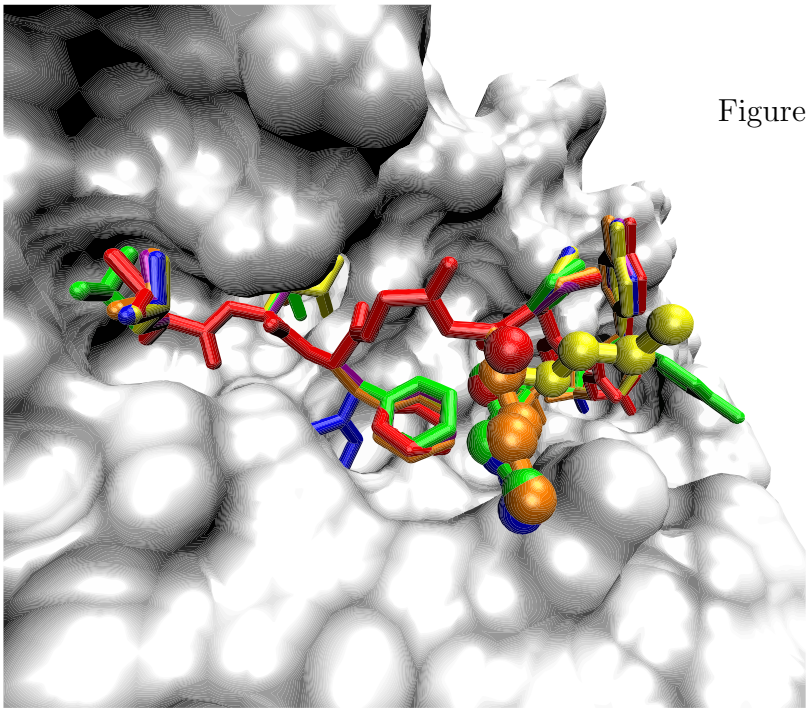


Figure 6: Molecular docking in the application of protein-ligand interactions, exemplified by a colored ligand molecule docking to a grey surface model of a receptor protein.

Knowing the appropriate answer (i.e. method) there are still enough problems to handle:

- *3D Puzzle* – Six degrees of freedom
- Flexible or rigid conformers
- Appropriate molecule representation
- Steric and electrostatic adjustment of conformations - Induced fit

- Treatment of biomolecules in solvation
- Energetic examination − Physicochemical descriptors
- Search algorithms - Scoring methods

Listing some direct applications of the protein-ligand docking we understand why it is worth to explore the molecular docking computationally:

- Function prediction of novel formed protein-ligand complexes
- Molecular physiological processes like signal transduction, gene expression, enzymatic reactions
- Drug discovery and design to create and test pharmaceutical compounds
- Virtual screening

Quantitative structure-activity relationship

Quantitative structure-activity relationship QSAR is the process or mapping by which chemical structure is correlated with a well defined property such as the pharmacological, chemical, biological or physical activity[44]. Especially in the fields of drug discovery and general Cheminformatics the QSAR is frequently used to relate chemical structure to observed properties. First achievements are assumed to date back over 150 years when a correlation between the boiling temperature and the molecule size, or better molecule weight, of alkanes was found.

A well defined property of course also needs measurable values that can be stored and that can be furthermore compared to each other. As an example the biological activity can be expressed quantitatively as in the concentration of a substance required to give a certain biological response, i.e. visually measurable growth or similar outcome. If now the variables being related to each other are expressed by numbers it is obvious to form a mathematical relationship in terms of a mathematical expression. The most general mathematical form of a QSAR would be

$$Activity\ A = f(physicochemical\ and/or\ structural\ properties)$$

and could be used to either calibrate and enhance established relationships or predict responses of chemical structures that have not been present in the training data set used for the mapping function.

The basic assumption of all chemically grounded hypotheses is that similar molecules have similar activities. This principle is amongst others called structure-activity relationship SAR. This inherently leads to the nontrivial problem how to define either the similarity of molecules or the difference or dissimilarity respectively between molecules. The stiffness of the mentioned problem to a great part derives from the fact that there are quite a few molecular properties or dimensions in a mathematical sense to observe a

quantitative difference. Furthermore it remains still unclear which molecular properties have an influence on which kind of activity, such as reactivity or solubility. Finally it must be said that the above-mentioned assumption has proven not to hold under all circumstances. The fact that not all similar molecules have similar activities is called the SAR paradox.

In the following a few concrete applications of the QSAR approach are listed to show its certain applicability:

- Prediction of boiling points of chemical compounds within a particular compound family
- Interactions between structural domains of proteins besides the interactions of a family of molecules with an enzyme or receptor binding site
- Chemical risk management using QSAR models

2.1.3. Representations

As soon as we think of chemical molecules, write them down on a paper or download the three-dimensional atom coordinates of a molecule we inevitably deal with representations. They are depictions and images of what we cannot see in reality but what we still work with. Currently dozens of different representations of chemical information are available and even more could be listed if we consider all subsidiary developments. Depending on the considered application and the use case the adequate form of representation has to be chosen. For the right or at least appropriate choice the requirements of the particular application must be taken into account. This means that the required information is available in the representation and furthermore that the information complies with the necessary detail.

For example, if we want to quickly depict the mechanism of a chemical reaction, maybe on a sheet of paper, it is certainly sufficient to write down the structural formula of the involved molecules instead of giving the coordinates and polarization of all atoms. The other way around it is rather useless to provide a molecule in form of a mere elemental formula if you want to calculate optimized geometries. Because of the different intentions and applications in computational chemistry there are naturally various representations and formats. Which format should be used still depends on the particular application and usage and therefore has to be chosen with respect to exactly those points. Below we describe several representations and formats used for the various Cheminformatics applications.

Chemical formula

The chemical formula or molecular formula such as H_2SO_4 for sulfuric acid is probably the easiest way of expressing information about the atoms of a single molecule of a chemical compound. The chemical formula identifies each individual element by its

chemical symbol and furthermore indicates the stoichiometric occurrence of elements in one single molecule of a chemical compound using a subscript number following the particular chemical symbol. Chemical formulae were introduced in the 19^{th} century by the well known chemist Berzelius and are still used in chemical equations to describe chemical reactions.

Nevertheless the chemical formula only gives the atomic stoichiometry of a chemical compound. No further information about the connectivity among the atoms or even the spatial positioning within the molecule can be read from it. For instance both chemical formulae CH_4 respectively H_4C stand for the molecule methane irrespective of its molecular geometry and conformation. To avoid this lack of information sometimes another representation has to be used such as the structural formula illustrating the connectivity among the atoms and giving an idea of the molecular geometry of the compound.

Structural formula

The structural formula of a chemical compound is a graphical representation of the molecular structure and therefore shows the arrangement of the atoms and the connectivity among them. The connectivity is shown either explicitly or implicitly via chemical bonds.

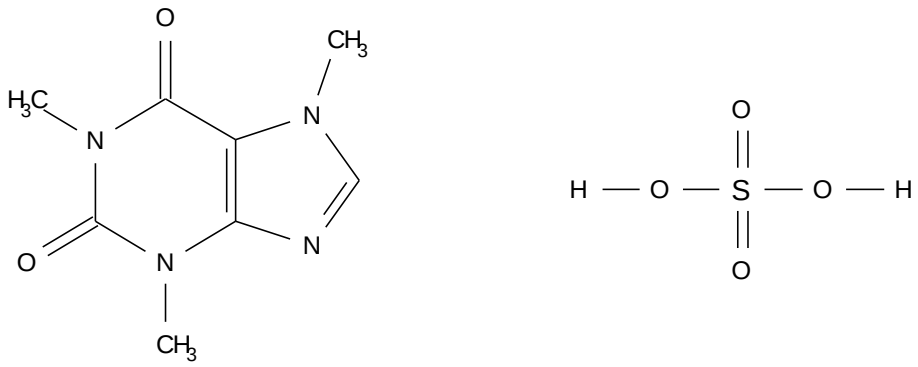


Figure 7: Examples of structural formulae showing caffeine on the left and sulfuric acid on the right.

In contrast to chemical names or chemical formulae structural formulae provide a representation of the molecular structure. The obvious advantage arises when chemists describe chemical reactions or processes because the structural formulae allow us to visualize molecular mechanisms and changes that occur during a reaction as depicted in figure 8. In order to provide a representation that covers the required information to uniquely describe molecules and outline reaction mechanisms such as changing electronic bonds and atom movements, the following demands need to be met. Molecules have to be labeled explicitly except for organic molecules where the mainly used atoms carbon and hydrogen do not need to be labeled as their position is defined by the molecular

context. Valence electrons and bonds connecting the atoms also need to be labeled only if it is unambiguous to leave the bond label. If a bond is present it also shows the bond type, i.e. single, double or triple bond.

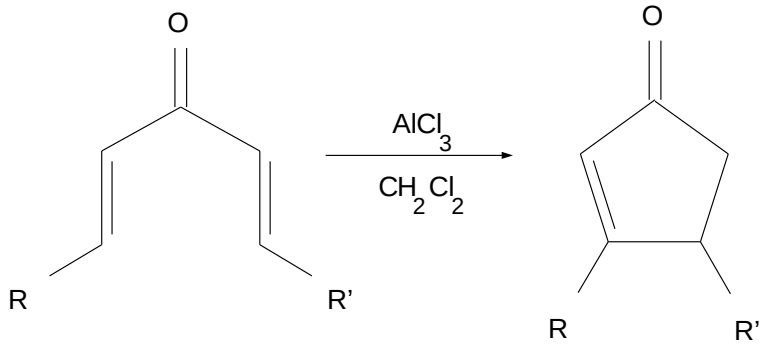


Figure 8: The Nazarov Cyclization depicted using structural formulae. The Nazarov Cyclization allows the synthesis of cyclopentenones from divinyl ketones. The reaction is catalyzed by strong Lewis acids.

Analog to the different isomeric forms of many chemical compounds there also exist many different developments of structural representations of molecules. In the following some molecule structure representations are listed and exemplified in figure 9:

- **Fischer projection**
  This representation is a projection of a three-dimensional organic molecule onto two dimensions.

- **Haworth projection**
  The Haworth projection is a common way of representing the cyclic structure of monosaccharides. The projection shows a simple three-dimensional perspective and indicates atomic bonds being closer to or farther from the observer by different line thickness.

- **Skeletal formula**
  Skeletal formulae are the standard notation for more complex organic molecules because they are quick and easy to draw. The reason for this is that on the one hand carbon atoms are only represented by vertices (corners) and line endings and not written explicitly and on the other hand hydrogen atoms are only marked explicitly when bound to elements other than carbon.

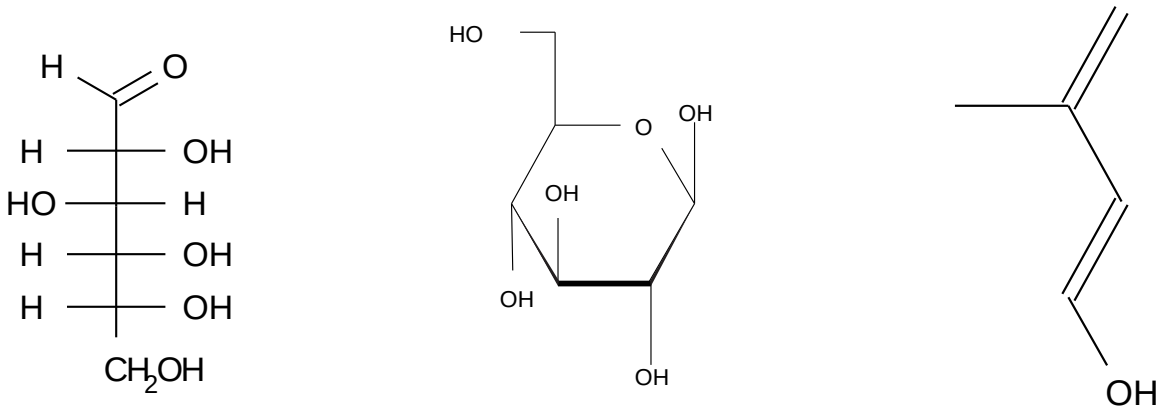


Figure 9: Examples of three popular projections. The left figure shows the Fischer projections of a D-glucose. The center figure shows the Haworth projection of beta D-glucose whereas also a sense of perspective is given by varying the binding line weight to outline the visual depth of the molecule. In the right figure the skeletal formula of 2-Methyl-1,3-butadienol is shown.

Molecule graph & Connection table

In Cheminformatics a molecular graph or chemical graph is a representation of the structural formula of a molecule in terms of graph theory, as mentioned in chapter 2.3. A molecular graph is an undirected labeled graph whose vertices correspond to the atoms of the chemical compound and whose edges stand for the chemical bonds connecting particular atoms. The vertex labels typically show the chemical elements symbol and the edges are labeled with the type of bonds. One possibility to store such a graph computationally is a connection table that map the information about adjacent vertices and by what edges they are connected onto a two-dimensional table with N columns and rows respectively, whereas N is the number of atoms in the considered molecule. A cell with table coordinates C and R indicates the connectivity of the atoms defined by column C and in row R. The meaning of a certain cell value varies between different formats. Nevertheless all sorts of connection tables mentioned here are redundant in the sense that the matrix they represent can be mirrored on the diagonal. For instance the following connection table formats based on the graph representation have been established, exemplified using figure 10:

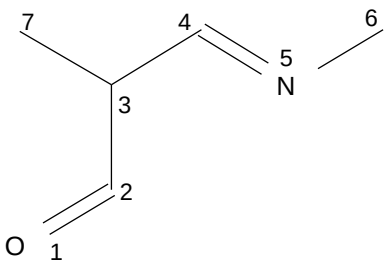


Figure 10: A numbered molecule structure used to explain the different formats of connection tables and matrices.

- **Adjacency matrix**

  In the adjacency matrix the cell values can only be 0 or 1 indicating whether two atoms (vertices) are connected by a bond (edge) or not.

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	0	1	0	0	1
4	0	0	1	0	1	0	0
5	0	0	0	1	0	1	0
6	0	0	0	0	1	0	0
7	0	0	1	0	0	0	0

- **Connection matrix**

  The connection matrix is set up similarly to the adjacency matrix with one crucial difference. The cell values do also indicate the particular chemical bond type and therefore go from 0 to 3, where 0 still means that the considered atoms are not connected at all.

	1	2	3	4	5	6	7
1	0	2	0	0	0	0	0
2	2	0	1	0	0	0	0
3	0	1	0	1	0	0	1
4	0	0	1	0	2	0	0
5	0	0	0	2	0	1	0
6	0	0	0	0	1	0	0
7	0	0	1	0	0	0	0

- **Distance matrix**

  The cell values of the distance matrix may show two characteristics. The value is either the geometric distance of the two particular atoms in the spatially spread molecule or the topological distance (figure on the right), for example the number of bonds (edges) forming the shortest path between each two atoms.

	1	2	3	4	5	6	7
1	0	1	2	3	4	5	3
2	1	0	1	2	3	4	2
3	2	1	0	1	2	3	1
4	3	2	1	0	1	2	2
5	4	3	2	1	0	1	3
6	5	4	3	2	1	0	4
7	3	2	1	2	3	4	0

Taking a look at the outlined table representations it can easily be recognized that all of them are redundant. The same informational content can be achieved when leaving one half of a particular matrix as it is mirrored on the main diagonal. More storage space can be saved if only cell values unequal to zero are stored. Especially the adjacency matrix and the connection matrix are rather sparse due to zero values.

Applications of the mentioned graph representation are first of all the calculation of topological indices or connectivity indices, like the Wiener Index, as part of graph isomorphism and of course the graph grammar approach applied to chemical reactions that our main application is based on. Other applications such as the computational storage of chemical data have been mentioned earlier in this thesis.

SMILES

SMILES is the abbreviation for *simplified molecular input line entry specification* and is a specification for unambiguously describing the structure of chemical molecules using short ASCII strings. It is therefore a linear notation for molecule structures. In history a substantial amount of effort has been spent on designing textual descriptions of molecular structures and graphs which could be used for computer aided storage, query and analysis [39]. Other textual description forms are known as Wiswesser Line Notation WLN and SYBYL Line Notation SLN. Nevertheless SMILES is still widely-used among chemical and biochemical applications, databases and research.

The specification SMILES was originally developed by David Weininger [36] in the late 1980s. Weininger later on founded the company Daylight Chemical Information Systems Inc. which since than advances the further development and extension of the SMILES [38]. Since the term SMILES denotes the mentioned specification for a molecular representation a specific instance should strictly be called SMILES string to avoid misunderstandings. The grammar for SMILES strings defines the representation of atoms, bonds, branches, aromaticity and even specialties like stereochemistry and isotopes.

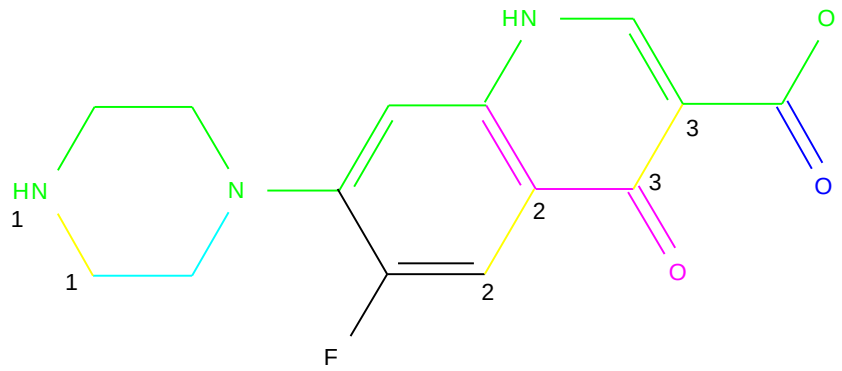


Figure 11: A visualized SMILES string generation. Given a molecule at first the cycles are broken (yellow bonds) and a main backbone (typically the longest path) is selected. Then the branches and broken cycles are written in parentheses according to their appearance on the backbone. The resulting SMILES string is: N1CCN(CC1)C(C(F)=C2)=CC(=C2C3=O)NC=C3C(=O)O

Atoms are represented by their chemical element symbol from the periodic table and have to be written in square brackets: [Fe], [Au]. The brackets can be omitted for the most common elements in organic components such as B, C, N, O, P, S, F, Cl, Br and I. Omitted brackets also imply that the proper number of implicit hydrogen atoms should be assumed by the particular unit interpreting the certain SMILES string. Eventual electrical charges of atoms are coded within the brackets by a + or − sign after the element symbol followed by the number of charges if greater than one: [Ti+4], [OH-].

Bonds are coded by the characters −, = and # according to the bond types single, double or triple bond. Again the most common bond type, the single bond, can be omitted. In such a case the non-existing bond between two sequenced atoms in the SMILES string is assumed to be single. Because of this circumstance butane simply can be written as SMILES string CCCC. Special ring closure labels are used to indicate connectivity between non-adjacent atoms in the SMILES string. The rings are numbered consecutively beginning with one, which for cyclohexane results in the string C1CCCCC1.

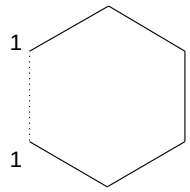


Figure 12: Example for ring closure labels, showing cyclohexane C1CCCCC1.

Since a SMILES string is defined to be a sequence of adjacent atoms, branches in a molecule structure have to be distinguished. Therefore branches are written with parentheses and may even occur hierarchically in the string, which means that branches can also start within another branch: CCC(=O)O. In computer science one would define this by the occurrence of at least two opening parentheses following in a sequence with possible atoms in between but without a closing parenthesis between them.

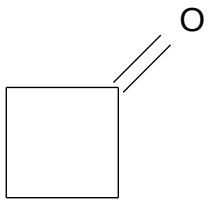


Figure 13: A branching example in cyclobutanone, specified by the SMILES string C1(=O)CCC1.

Aromatic atoms, which can be C, O, S and N, are outlined with their lower case c, o, s and n. Bonds between aromatic atoms are by default aromatic but can be specified explicitly using the ":" symbol as a forth bond type. As a result pyridine can be represented by the SMILES string n1ccccc1.

The ability of SMILES strings to represent not only the two-dimensional structure of a chemical compound but also certain information about its stereochemical properties, is very useful for applications and users working with them. The configuration around double bonds is declared using the symbols / and \. A trans-difluoroethene, in which the fluorine atoms F are on opposite sides of the double bond, is written as F/C=C/F. The

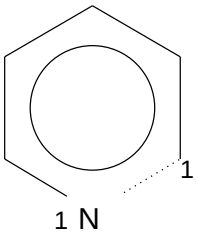


Figure 14: An aromatic example showing pyridine, specified by the SMILES string n1ccccc1.

cis-difluoroethene is therefore written as F/C=C\ F, depicting a symmetry of slash and backslash on behalf of the vertical axis of the double bond. To distinguish enantiomers of a molecule the view onto its chiral center must be specified. This configuration at tetrahedral carbon is specified by @ or @@, standing for the counter-clockwise or clockwise appearance of the substituents in the SMILES sequence.

Figure 15: Structure of trans- and cis-difluoroethene, specified by the SMILES strings F/C=C/F (trans) and F/C=C\ F (cis).

Isotopes are simply defined by their integer isotopic mass preceding the atomic symbol. Thus butane with one carbon-14 atom is written as CC[14C]C.

In graph-based nomenclature the computational procedure for deriving a SMILES string can be defined as follows: A SMILES string is obtained by printing the symbol nodes (atoms) encountered in a depth-first tree traversal of the chemical graph. The molecular graph is optionally trimmed to remove hydrogen atoms and cycles are broken as mentioned above to turn the graph into a spanning, cycle-free tree. At the positions where cycles have been broken enumerated numeric labels are included to indicate the connectivity of the particular nodes. Parentheses are used to represent branching points on the tree and a branching-depth respectively.

Now that we know how to extract a SMILES notation from a chemical compound we might recognize that there is more than one way to build a correct SMILES string for one single molecule. For this reason it is of high importance for databases and many other applications to work with and provide unique SMILES strings that are always the same string for the chemically equal compound. Such SMILES strings are called

canonical. All of the following SMILES strings are valid for describing cyclobutanone (figure 13), the last one being the canonical:

C1(=O)CCC1

C1C(=O)CC1

C1CC(=O)C1

C1CCC1=O

O=C1CCC1

In August 2006 the International Union of Pure and Applied Chemistry IUPAC introduced the International Chemical Identifier InChI as standard for molecular formula representation. Although SMILES is considered of being only slightly more readable for humans than the InChI, it has actually a wide base of software applications and support in Cheminformatics.

Besides the linear representation of molecule structures the SMILES framework also provides a convenient query-specification language (SMARTS) and representations for chemical reactions (SMIRKS) [38]. Both the *SMILES arbitrary target specification* SMARTS and the SMIRKS are extensions of the SMILES and therefore share a common string grammar with certain extensions. SMARTS basically allows one to query substructures in SMILES strings and therefore defines patterns to match target SMILES string. The extensions of SMARTS compared to SMILES contain wild card symbols and logical operations like AND and OR. For instance the SMARTS pattern [O,S]CCC matches a set of four atoms whereas an oxygen or a sulphur is bonded to three carbons.

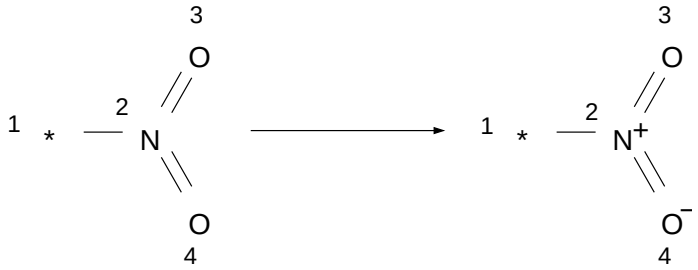


Figure 16: The chemical reaction nitro group conversion written as SMIRKS string showing wild cards and the atom mapping.

SMIRKS, however, is a combination of SMILES and SMARTS used to depict chemical reactions in terms of graph transformation. A SMIRKS reaction string specifies the involved reactants, products and optionally the agents in the format of "reactants>agents>products". Molecule SMILES strings within these three groups are concatenated via a dot "." and the groups are separated using the > symbol. Furthermore an atom mapping can be coded using the SMIRKS outlining corresponding atoms on

the reactants side and on the products side of the reaction. The atom map is written as integer values within the atom label delimited by a colon. The SMIRKS string for the reaction in figure 16 is: [*:1][N:2](=[O:3])=[O:4]>>[*:1][N+:2](=[O:3])[O-:4].

Networks

In the previous sections various representations of chemical compounds and molecules in their single occurrence have been discussed. Now we are devoted to the depiction of a certain amount of molecules relating to each other, i.e. networks in terms of graph theory. A network can be defined as a directed graph with weighted edges, whereas these properties might differ between the various applications. The theory behind the special application of graphs in terms of networks is called network theory and apparently part of graph theory. In Cheminformatics and Bioinformatics networks are used when representing all kind of reaction networks. A reaction network outlines the relationships between the single contained molecules and therefore also allows inferring interactions among them. Analog to the necessity of representing chemical compounds, networks also have to be represented if we want to store data and information about them or process them computationally.

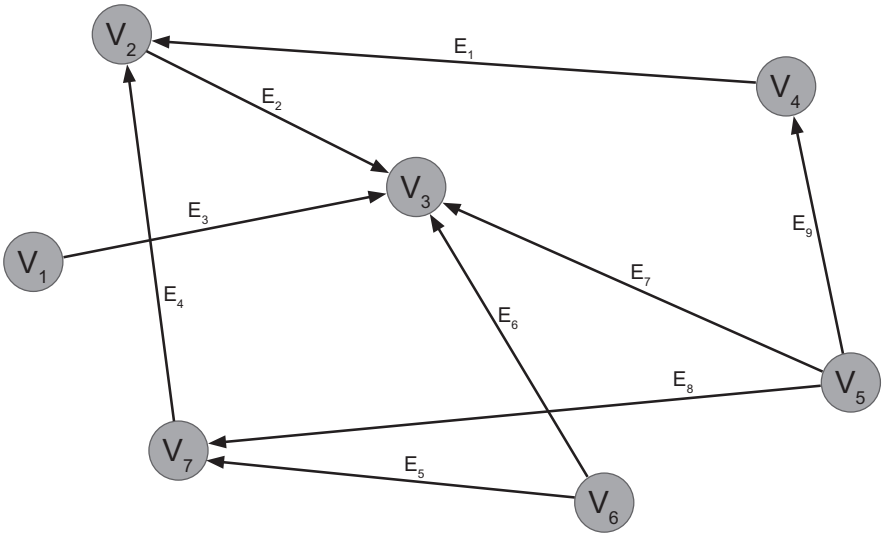


Figure 17: A reaction network only consisting of vertices V (grey circles) meaning molecules and directed edges E representing the reactions among the chemical compounds. Such a network is called substrate graph.

As the graph theoretical part in this thesis follows in section 2.3 we will confine ourselves to descriptive visualizations of the possible and useful network representations in this working field. In figure 17 we outline the probably most natural depiction one might think of. It consists of the key elements of a chemical reaction, the involved molecules (vertices V_1 to V_7) and a direction pointing from the reactants to the products (edges E_1 to E_9). On the other hand this representation has obviously a pitfall that we will

describe in the following. If more than one reaction leads to the same molecule species there are multiple equal reaction-edges pointing to this particular molecule species. The problem now appears if we want to get back the information about what reactions lead to our regarded compound and further what reactants belong to those reactions. It arises that the imaginative conversion or projection to the depicted substrate graph is not bijective and information definitely gets lost. In the next representation a second vertex type is utilized to stand for the intrinsic reaction. Molecule vertices hereby can only be related using a reaction node. The directed edges are needed on the one hand to group reactants of a reaction and point to the particular reaction node, and on the other hand to point from the reaction node to all products.

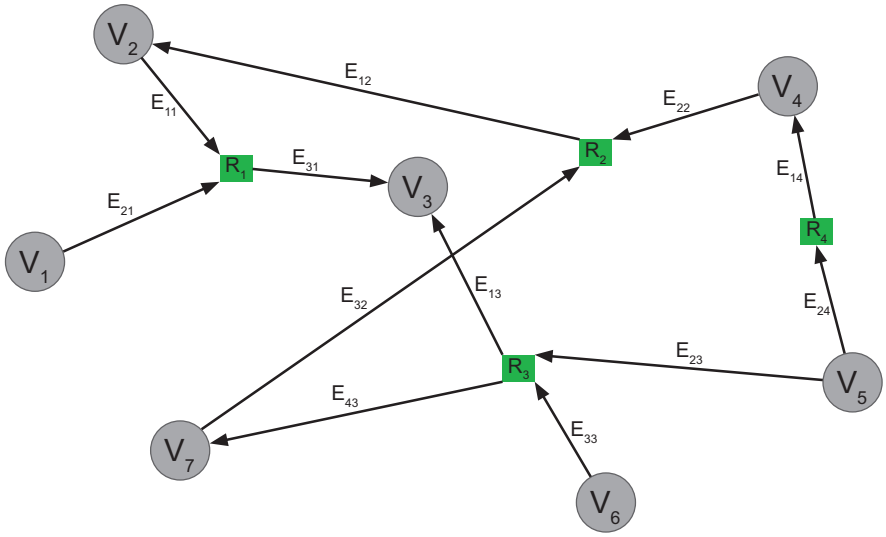


Figure 18: Reaction network with vertices V (grey circles) meaning molecules, directed edges E and separate reaction vertices (green rectangles) R_1 to R_4 marking reactions among chemical compounds.

The advantage of this treatment is that all information about the chemical reactions remains in the network and is therefore not lost. Nevertheless, there remains the disadvantage that the actual chemical reaction now requires not only the edges but also another vertex type. If we were to find a satisfying representation that not only covers all needed information but is also somewhat close to reality, we could call this minor drawback *unesthetic*.

Still there is another possibility to even get rid of the extra vertex type for reactions. The talk is of hypergraphs. A hypergraph is a generalization of a graph with only one crucial difference [6, 42]. An edge no longer has exactly two incident vertices but indeed connects potentially many vertices. A hyperedge[1] can therefore be thought of as a set of incident vertices and is mathematically a subset of the whole network. Moreover a hyperedge may also be imagined as an edge in a higher dimensional space in which more

[1] An edge in a hypergraph is adequately called hyperedge.

than two areas border on a single (hyper)edge. This single difference between a graph and a hypergraph in fact implies a lot of conclusions we cannot even touch on.

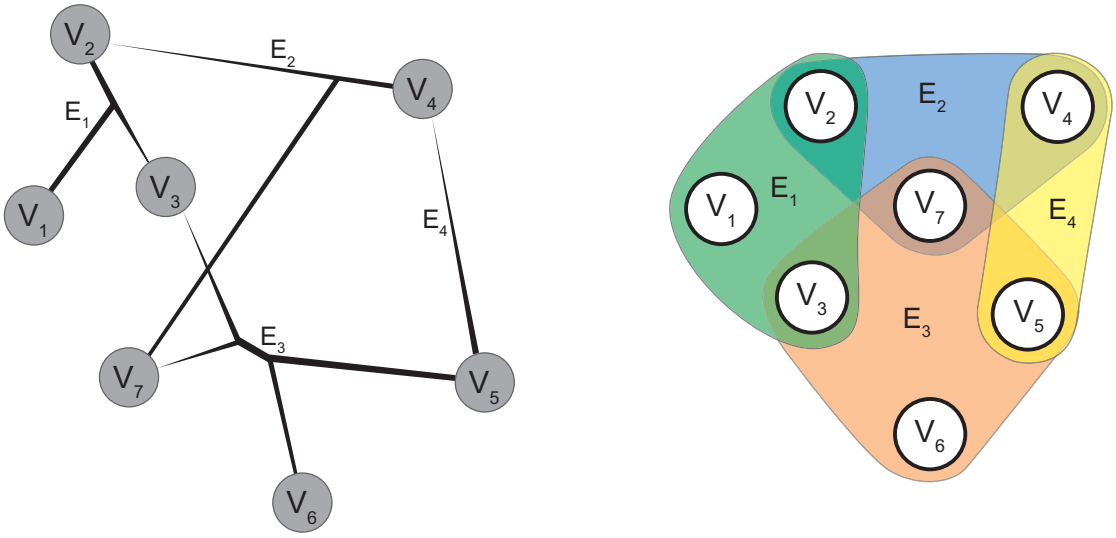


Figure 19: Reaction network represented as a directed hypergraph on the left. The vertices V (grey circles) are related to each other according to the chemical reactions R_1 to R_4 using hyperedges E_1 to E_4 (black lines). A hyperedge starts at each vertex describing a reactant (broad end) and ends in each of the vertices standing for product molecules (thin end). The right side shows the set-theoretical hypergraph.

Undeniably we finally described the ultimate representation for chemical reaction networks in which a network of seven molecule species and four reactions is depicted as a corresponding directed hypergraph of seven vertices and four hyperedges. Unfortunately not only the imagination of a hypergraph is of high complexity. The theoretical descriptions and furthermore the graph theoretical operations on a hypergraph, such as the graph rewriting, are of even higher complexity compared to an "ordinary" graph. As a consequence in computer science, especially in Cheminformatics and Bioinformatics the hypergraph is rarely used and still lacks a widespread applicability.

2.2. Chemical Reaction Network Theory

We studied the Chemical Reaction Network Theory CRNT mostly guided by the work of Gunawardena [12]. Models of chemical reaction networks or intra-cellular biological systems are in general based on systems of nonlinear ordinary differential equations (ODEs). Because of the complexity of these ODEs mathematical theorems would be appreciated that are able to describe fixed points[1] or other hints for the solutions of the ODEs. The CRNT, however, supplies a context for which relevant mathematical

[1]Fixed points are points x of a function $f(x)$ where $f(x) = x$

results for this problem have been established. More exactly it contains several theorems about systems of nonlinear ODEs which describe the behavior of reactor basins used in chemical engineering.

The Chemical Reaction Network Theory has been developed over the last 30 years. It bases on the work of Horn and Jackson [31] and was subsequently extended by Martin Feinberg and his students [32]. The theory introduces new concepts, as the *deficiency* of a reaction network[1], and even though it is still unclear to what extent it is applicable to biological problems and viewpoints there can already be made interesting associations between both fields. The conclusions of the CRNT sometimes hold irrespective of parameter values in the system which might for instance be associated with biological robustness[2].

The key insight of the CRNT is that the ostensible nonlinearity of the ODEs arising from mass-action kinetics accommodates considerable linearity. This results from the immanent network of chemical reactions which in fact define a directed graph upon the contained molecule species. In addition this intrinsic structure limits the possible nonlinearity and furthermore strengthens why the results of the CRNT are possible. Thus, to explain the main results we utilize an example reaction network taken from Feinberg's work [33]:

$$\begin{array}{rcl} A_1 + A_2 \rightleftharpoons A_3 & \rightarrow & A_4 + A_5 \rightleftharpoons A_6 \\ 2A_1 & \rightarrow & A_2 + A_7 \\ \nwarrow \quad \swarrow & & \\ A_8 & & \end{array} \tag{1}$$

In the CRNT the following properties of a reaction network are of interest and essentially for deriving and expressing further thoughts and explanations:

- *The complexes of a network* are the sets of molecule species before and after a particular reaction arrow. Therefore the set of complexes for network (1) is

  $$\{A_1 + A_2,\ A_3,\ A_4 + A_5,\ A_6,\ 2A_1,\ A_2 + A_7,\ A_8\}$$

  and the number n of complexes is 7 respectively. Complexes can also be written as so called *complex vectors*. A complex vector is of length N, where N is the number of species in the network, and represents the stoichiometric coefficients for the molecule species of the associated complex. The species number N being 8 and associating vector indices 0 to 7 to species A_1 to A_8 we can for instance write complex $A_4 + A_5$ as vector $[0, 0, 0, 1, 1, 0, 0, 0]$ and complex $2A_1$ as vector $[2, 0, 0, 0, 0, 0, 0, 0]$ respectively.

[1] The deficiency of a reaction network is the dimension of a particular vector space and can be calculated after determining the network variables number of complexes, number of linkage classes and its rank. Terminology and mathematical details can be found in reference [33].

[2] Robustness by means of a ubiquitous and fundamental feature of complex evolvable systems. It facilitates evolvability and vice versa robust traits are often selected by evolution.

- *The reaction vectors of a network* are defined analog to the complex vectors above and therefore represent the stoichiometric changes induced by the reactions. A reaction vector is obtained by subtracting the reactant side from the product side, in terms of the particular complex vectors. Thus, for reaction $2A_1 \rightarrow A_2 + A_7$ the reaction vector is the subtraction of complex vector $A_2 + A_7$ minus complex vector $2A_1$. This produces the final reaction vector $[-2, 1, 0, 0, 0, 0, 1, 0]$, which also means that within this reaction two molecules of species A_1 are destroyed and one molecule of the species A_2 and A_7 is created in each case.

- *The rank of a reaction network* is defined as the number of elements in the largest linearly independent set of reaction vectors. Whether a set of such vectors is linearly dependent or independent can be distinguished when going a path along all reaction in the particular set. If we cannot make any cycles or take a path on which reactions might collide with other reaction, in the sense of annihilating each others stoichiometric change, the set is called linearly independent. Otherwise it is linearly dependent. More mathematically the rank of a reaction network can be defined as the rank of a matrix consisting of all single reaction vectors, whereas the matrix has to be reduced to echelon form before determining the rank. For our example network (1) the rank is 5, as the largest set of linearly independent reaction vectors is the following:

  $$\{A_1 + A_2 \rightarrow A_3,\ A_3 \rightarrow A_4 + A_5,\ A_4 + A_5 \rightarrow A_6,\ 2A_1 \rightarrow A_2 + A_7,\ A_8 \rightarrow 2A_1\}$$

  The matrix derived from reaction network (1) is given as follows:

  $$R = \begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

  After reducing the matrix to row echelon form, for instance via Gaussian elimination, the number of non-empty rows is the rank of the matrix, which is 5 in our example.

- *The linkage classes of a reaction network* are sets of complexes that show linkage among each other only within one set but not to any complex of another set. Linkage classes can also be understood as the partitions in a *k-partite* reaction network graph. In reaction network (1) there are the following two linkage classes:

  $$\{A_1 + A_2,\ A_3,\ A_4 + A_5,\ A_6\} \text{ and } \{2A_1,\ A_2 + A_7,\ A_8\}.$$

- *The deficiency of a reaction network* is now defined as a non-negative integer δ that can be calculated when knowing the above mentioned properties of the particular network. The deficiency δ is defined by the formula

$$\delta = n - l - s$$

where n is the number of complexes in the network, l is the number of linkage classes and s is the rank of the network. For our network (1) $n = 7$, $l = 2$ and $s = 5$, which results in a deficiency $\delta = 7 - 2 - 5 = 0$.

On the basis of the just defined properties of a chemical reaction network, especially the deficiency and the rank, the CRNT presents the *deficiency zero theorem* and the *deficiency one theorem* among others. These two major theorems make statements about the kinetics and the existence and properties of steady states of a reaction network in a way that avoids complex and intricate calculations of differential equations.

2.3. Graph theory

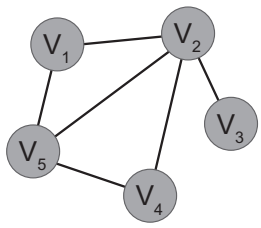
Graph theory is simply the study of graphs [45]. As mentioned earlier in Cheminformatics graphs are used in various applications and for diverse problems. The usage ranges from representing single molecules to reaction networks of chemical compounds and also more generally to the representation of any relations between compounds or bigger complexes such as in gene maps or proteomics. Leonhard Euler's work on the *Seven Bridges of Königsberg (1736)* is regarded as first paper in graph theory. In order to be able to appropriately describe usages and applications of graphs and furthermore to utilize their inherent properties and possibilities, we start with the very basics of graphs. Subsequently we go further to some mathematical problems, also occupying all sorts of computer scientists, and finally to graph grammars and graph transformation.

2.3.1. Graphs

Graphs undeniably provide the most basic mathematical model for entities and relations [7]. A graph G can be declared to consist of a set of vertices V and a set of edges E such that every edge $e \in E$ links two adjacent vertices (source, target) $s, t \in V$. Therefore a graph is entirely written as $G = (E, V)$. Furthermore a graph shows some properties that are described by depicting different types of graphs in the following list:

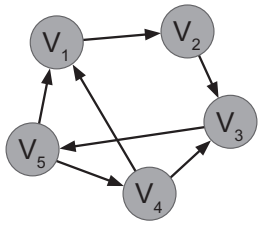
- **Undirected graph**

  An undirected graph $G = (V, E)$ is characterized by edges E that do not show any orientation. Each edge is therefore not an ordered pair of vertices in V but a set or multiset of vertices. The edges are depicted by lines without any arrows and outline the symmetric relations present in the undirected graph. An example is shown in the right figure.



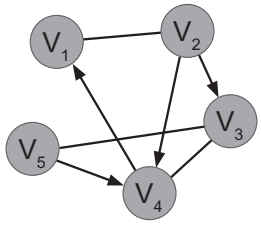
- **Directed graph**

  A directed graph or digraph is an ordered pair $D = (V, A)$ with a set of vertices V and a set of directed edges A, also called arcs or arrows. Each arc represents an ordered pair of vertices $(s, t) \in V$ and therefore directs from s to t. Furthermore a directed edge (t, s) leading from t to s is called (s, t) *inverted*.



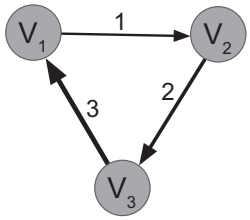
- **Mixed graph**

  A mixed graph is an ordered triple $G = (V, E, A)$ and represents the combination of undirected and directed graphs. Obviously both classes of edges are allowed in one graph and therefore directed and undirected graphs are only special cases of the mixed graph.



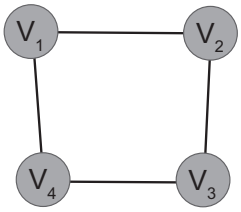
- **Weighted graph**

  In a weighted graph the edges are kind of labeled to assign a weight to each edge. Those weights, typically numbers, might describe various properties depending on the scope of the particular application of the graphs. In Cheminformatics the weight might denote the bond type in a molecular graph, the spatial distances between the atoms in a molecule or even the reaction rate in a chemical reaction network. In our example the edge weight is furthermore stressed by according line width.



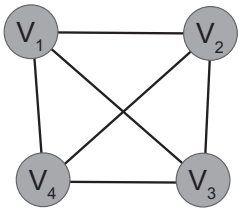
- **Regular graph**

  In a regular graph every vertex has the same number of neighbors, i.e. the same number of adjacent vertices. Thus, every vertex has the same degree k and in this case the graph is then called a k-regular graph or regular graph of degree k. Our example is a 2-regular graph as each vertex has degree two.



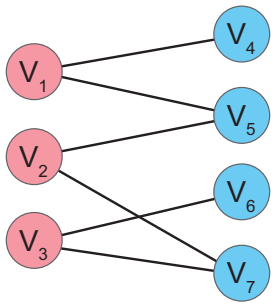
- **Complete graph**

  A graph is said to be complete if each pair of vertices is connected by an edge. As this means every vertex has the same degree, it follows that a complete graph is also regular with the degree $n-1$ where n is the number of vertices. The described completeness is also called maximum connectivity.



- **Bipartite graph**

  A bipartite graph is the special case of a k-partite graph for $k = 2$, whereas a k-partite graph can be divided into k disjoint sets of vertices such that no edge in the graph connects vertices contained in the same subset. This means edges may only connect vertices of disjoint sets. If this requirement is fulfilled the graph is divided into k partitions. The example figure on the right shows a bipartite graph where the partitions are formed by red and blue colored vertices. Consider that there is only connectivity among the partitions but not within one.



Having depicted some properties of graphs we now know the different viewpoints a graph can be specified and examined. In the following section we dedicate ourselves to some famous problems and implications regarding graphs and operations on them.

2.3.2. Famous problems

As a consequence of the wide range of applications of graphs and their variety with respect to the observable properties, there have been defined and proposed many problems of either theoretical kind or nonetheless practical kind. In this chapter we describe the most important graph theoretical problems regarding our overall application.

Route problems

Route problems are among the oldest applications in graph theory. As mentioned earlier the well-known mathematician Leonhard Euler laid the foundations of graph theory and anticipated the idea of topology with his work on the *Seven Bridges of Königsberg*. In Euler's time there were seven bridges over the river Pregel in Königsberg and the particular question was to find a way through the city that would completely cross every bridge exactly once. Euler utilized graph theoretical considerations to prove that it was actually *not* possible to find such a path [47]. However, Euler did not only falsify this special problem mainly concerning tourists of the Russian city Königsberg. Euler indeed deduced general statements about finding such so called *Eulerian* paths by reducing the city Königsberg and its bridges to an abstract graph of vertices (land masses) and edges (bridges) connecting them. He then showed, that the existence of such a path depends on the degrees of the vertices and in the special case of Königsberg the four vertices connected by seven edges lead to an impossibility of finding an Eulerian path. Nowadays finding Eulerian paths is mainly used in Bioinformatics when a complete DNA sequence is to be reconstructed from the assembly of its single fragments – the biochemical output of most DNA sequencing techniques.

Another route problem was introduced by William Rowan Hamilton. Hamilton invented the *Icosian game* in which a Hamiltonian cycle or path has to be found in the

edge graph of a dodecahedron[1], which is more or less its two-dimensional projection. In contrast to the Eulerian path, which is defined to visit all edges of a graph, the Hamiltonian path visits every vertex exactly once [48]. Additionally an edge may not be visited more than once, but may of course be missing at all in the path. Figure 21 shows a Hamiltonian cycle which is nothing more than a Hamiltonian path requiring the start vertex and the end vertex to be adjacent. A graph is called Hamiltonian if it contains a Hamiltonian cycle. Determining whether Hamiltonian paths or cycles exist in a graph is the *Hamiltonian path problem* which was proven to be NP-complete[2].

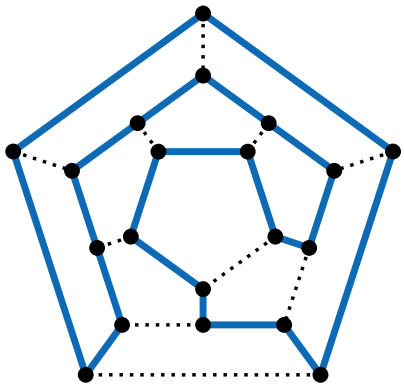


Figure 21: Hamiltonian cycle in the edge graph of a dodecahedron. The dodecahedron is Hamiltonian like all Platonic solids.

Our next graph theoretical problem is called *Shortest path problem* and already implies what it is about. The talk is of finding the shortest path between two vertices in a weighted graph, whereas each path is rated with the sum of all involved edges or better their weight. Regardless of all known generalizations we describe only the *single-pair shortest path problem* in which only one path between two particular vertices is questioned. Applications of handling and solving this problem show the immense importance of theoretical work in the working field of graph theory and are therefore listed below:

- *Routing services* between physical locations (Navigation systems).
- *Deriving phylogenetic trees* from DNA sequences or fragments by assessing the genetic distances among them.
- *Robotic systems* optimizing sequences of operations.
- *Routing algorithms* for generating electronic printed circuit boards (placement of components and circuit paths).

[1] A dodecahedron is a polyhedron with twelve flat faces. A regular dodecahedron is a Platonic solid and is composed of twelve regular pentagonal faces.

[2] Complexity class NP-complete means, that solutions to a problem can be verified quickly (in polynomial time), but there is still no efficient way of computing a solution of a particular problem. Therefore approximation algorithms and heuristics are used.

Related to the *shortest path problem* and even more interesting for transport companies and also tourists is the famous *Traveling salesman problem TSP* [49]. Obviously this problem comes from the practical considerations one has to make when planning a route under certain circumstances. In the present form this circumstances define the cities one has to visit and the aim is to minimize the total way for the trip. As the main applications for the mentioned problem are still found in the transport and delivery field an additional requirement is that the traveler returns back *home* again. More general and abstract we might again switch to the world of graphs and declare the problem as follows: Given a weighted graph $G = (V, E)$ and a set of selected vertices that need to be visited, the shortest cycle $p = (V_p, E_p)$ containing all of those vertices is demanded, whereas the cost-function $f(p)$ for comparing the cyclic paths is defined as the sum of the weights of the n participating edges $e \in E_p$, i.e.

$$f(p) = \sum_{n} weight(e)$$

Already the first mathematicians studying the problem in the 1930s observed the important differences to the *shortest path problem*. Following always the shortest path between two successive vertices equals the nearest neighbor heuristic which was shown to be non-optimal. Cases in which it is better (entirely shorter) to first travel to a more distant target but reach the following target in a shorter way then from a point that was nearer in the first step can easily be constructed and are exemplified in the following figure:

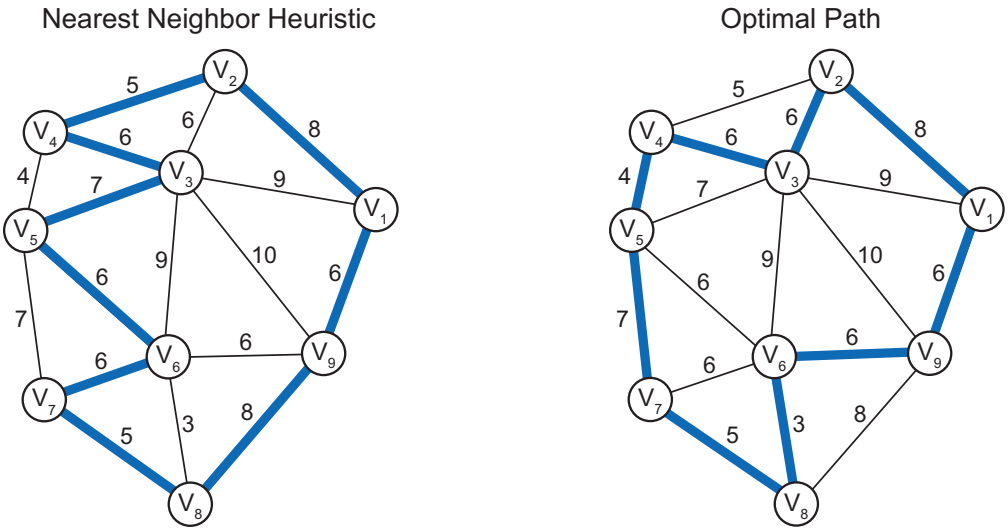


Figure 22: Example of a weighted undirected graph depicting the *Traveling salesman problem* for visiting all nine vertices V_1 to V_9 in the shortest path through the edges without using an edge more than once. In contrast to the path length of 57 units found by the nearest neighbor heuristic, the optimal, because shortest, path for a *traveling salesman* would be only 51 units.

As the TSP has been shown to be NP-hard the direct calculation of an exact solution is only feasible and practicable for small problems such as in figure 22 above. The reason for it is that an exact solution for the general problem can only be found by a *brute force* search incorporating all possible permutations, which implies a factorial runtime complexity $\mathcal{O}(n!)$. Although nowadays there exist faster algorithms it is still not practical for larger problems like the ones emerging in industry, which is why heuristics have been developed. Besides iterative improvements and randomized improvements like Monte Carlo type algorithms or Markov chain algorithms, for special cases heuristics could be optimized to find exact solutions.

Subgraphs

A subgraph H of a graph G consists of a subset of the vertices of G and furthermore of a subset of the edges of G. The relation between a graph G and a subgraph H may also be summarized by saying: Graph G *contains* another graph H. Other subgraph definitions reflect different assertions about the relation between G and H. A spanning subgraph H, for instance, has the same vertex set as its supergraph G. A subgraph H is furthermore called induced, if it contains all edges from G that connect the same vertices of H in G. The subgraph H is then said to be induced by its vertex set $V(H)$. Otherwise G can be declared to be H-free, as H is no induced subgraph of G.

The really *stiff* problems now occur when trying to verify a graph-subgraph relation. It is called the *Subgraph Isomorphism Problem* and already encapsulates the hardest parts: finding a subgraph within another graph and doing so for any isomorphic subgraph as input, which presumes determining whether two graphs are isomorphic. Therefore we first describe the graph isomorphism.

In graph theory two graphs G and H are said to be isomorphic if they have the same number of vertices which are furthermore connected in the same way [50]. In other words there exists a bijective function that maps the vertex set $V(G)$ of graph G to the vertex set $V(H)$ of graph H such that the adjacency information of $V(G)$ is the same as of $V(H)$ and vice versa, i.e. the set of edges remains the same. We can then write $f\colon V(G) \to V(H)$. Such graphs are called isomorphic and their relation can be summarized by writing $G \cong H$.

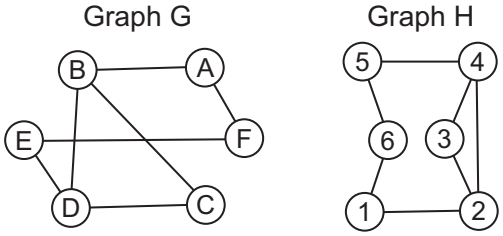


Figure 23: Isomorphic graphs G and H with a bijective function mapping vertex set $V(G)$ = [A, B, C, D, E, F] to $V(H)$ = [1, 2, 3, 4, 5, 6].

For the above outlined graphs the isomorphism can be easily verified. After transforming the vertex set of one graph into the vertex set of the other graph the adjacency

relations remain the same. Nevertheless the general task of determining graph isomorphisms is of high complexity and could still not be ranked satisfactorily according to complexity classes. To build now bridges to Cheminformatics we want to mention the related issues of isomorphic chemical graphs and canonization. The canonization, or at least the initial idea of it, has been described earlier in this thesis in terms of canonical SMILES strings. The incentive behind this approach is to assign one unique SMILES string to all possible inputs for one particular molecule species. Ostensible applications are database related storage of unique chemical compounds, which is also a reason for the many different canonization algorithms having been developed so far. Analog to the canonical SMILES, and as it is nothing else than the following, also the canonization of a graph is aimed at providing unique graphs for all possible isomorphic graphs. Most algorithms number the vertices in a defined way that results in the very same order for all isomorphic graphs.

Isomorphism regarding chemical graphs is a special case in graph isomorphism. Because the vertex sets of two isomorphic chemical graphs contain the same vertex labels, only their order might differ. Therefore the task of identifying isomorphic molecule graphs can be imagined by comparing graphs with their for example rotated or mirrored copy. As chemical graphs are typically edge-labeled and the edge-labels as part of the adjacency relation need to be matched, the problem remains hard *enough* compared to the general graph isomorphism.

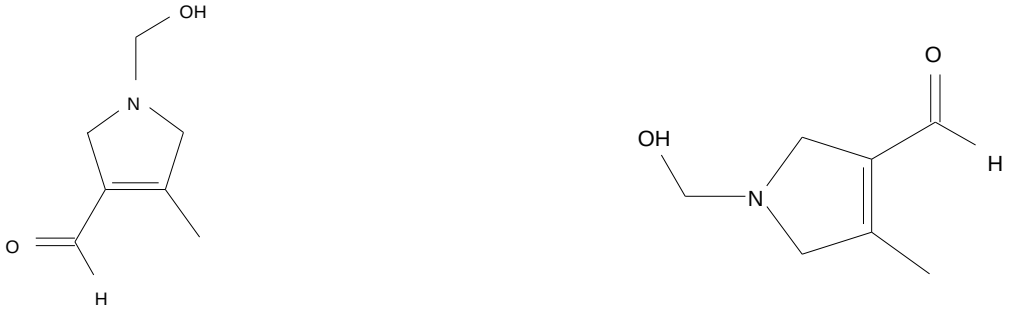


Figure 24: Both molecules are isomorphic and therefore show the *same* graph and the same unique SMILES string CC1=C(C=O)CN(CO)C1.

The second part of the previously mentioned *Subgraph isomorphism problem* regards the matching of subgraphs against possible supergraphs containing them. As described earlier a subgraph quasi represents a subset (in terms of vertices and edges of course) of another graph. Looking for a given subgraph H in another target graph G presumes that every imaginable graph isomorphic to H is equally found, if H is a subgraph of G. Applications of subgraph matching can be found in various working fields, ranging from Cheminformatics, where similarities between chemical compounds are determined by matching parts of their molecular graph against each other and chemical databases are queried for substructures, and Bioinformatics, for the investigation of gene and protein interaction networks, to even modeling social networks mathematically. In addition subgraph matching is certainly also the main step in graph rewriting, as the input graphs

(the left hand side) first need to be matched with the rewrite rule before the transformation can actually be applied.

What is now the difficulty with subgraph isomorphism that we call it a problem? The requirement of searching for one graph in another one is known for a few decades now and nevertheless there has been established a substantial amount of work in this direction. In 1976 Ullmann [17] published his exact matching algorithm that bases on a backtracking procedure with an effective look-ahead function to reduce the search space [16]. Due to its generality and effectiveness the Ullmann algorithm can be applied to both graph isomorphism and subgraph-graph matching. Although the algorithm is rather old and there have been developed more specialized ones for large graphs [16] or reducing the matching problem to the clique detection problem[1], it is still one of the most commonly used algorithms for exact graph matching. The following figure outlines the subgraph matching procedure:

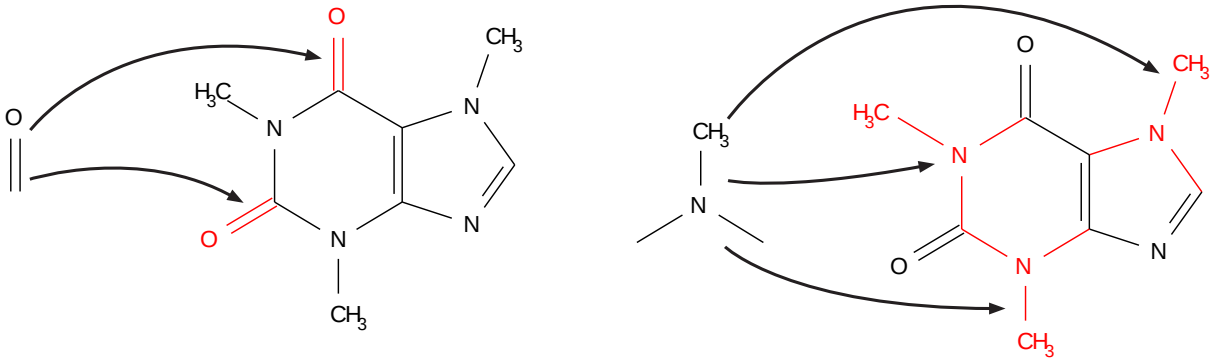


Figure 25: Matching two subgraphs with SMILES strings C=O and CN(C)C in the target molecule graph of caffeine. All matched subgraphs are colored red in the supergraph.

One part of the matching algorithm is enumerating the vertices in both graphs in a unique way, which is meant by *isomorphism*. The next is to effectively go through the target graph and look for occurrences of the query subgraph. Due to parallel iterating over both graphs and other improvements Ullmann's algorithm was shown to scale with polynomial time and memory requirement of $\mathcal{O}(N^3)$ [17]. Another advantage of the Ullmann algorithm is that its applicability is not limited to merely finding subgraphs within other graphs. In his work Ullmann describes and derives mathematically that the subgraph matching is a generalization of the graph isomorphism because it is contained as a sub problem.

[1] A clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge. The clique problem, however, is to find cliques of a given size in a graph.

2.3.3. Graph transformation

On closer look graphs, regardless of the actual application, always represent a static image of something, be it a molecule or a chemical reaction network. Even if we meant something in motion, like the expanding behavior of a reaction network or a single chemical reaction, a graph only depicts a snapshot of a possible time evolution. Thus graph transformations, also graph rewriting, allow us to describe and model changes of graphs, either concerning labeling or the adjacency information. As graph transformation is one of the core concepts of our work we will start with some general basics and then proceed with chemical graph transformation.

Common basics

Originally graph transformation has evolved from classical approaches to rewriting, like Chomsky grammars and term rewriting, due to their shortcomings in expressiveness [7]. The first proposals in the direction of graph rewriting appeared in the late 1960s and early 1970s and since then a broad theoretical basis has been developed knowing many concepts and having found and solved many problems. The first basic concept is the modeling of given scenarios by applying abstraction processes and further generalizations as shown in figure 26.

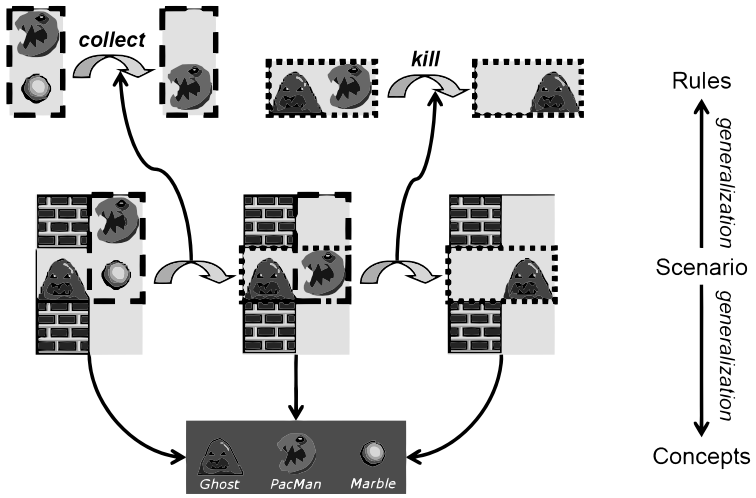


Figure 26: The general concepts of graph transformation systems depicted on the example of the famous game PacMan, viewed in terms of graphs. Graph rewrite rules (*collect, kill*) are derived from known scenarios by ignoring uninformative context and considering nodes labeled with the concepts *Ghost, PacMan* and *Marble*.

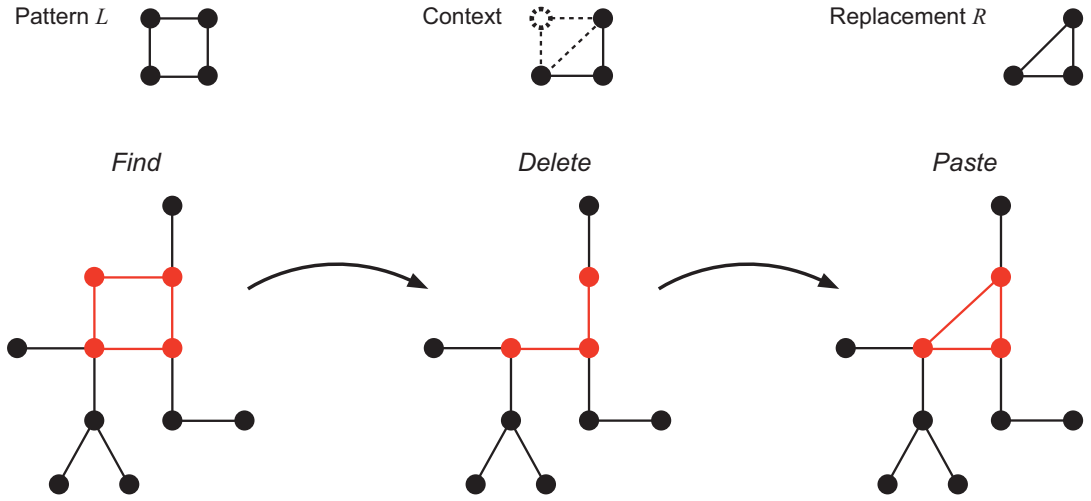
The above figure also outlines the basic idea of a type-instance relation. The concepts or entities are listed on the bottom of the figure and represent the types of available nodes or vertices in the graphs (scenarios). Every occurrence of these entities in the

scenarios is of course an instance of one of the types. Considering this relation is crucial for deriving rewrite rules from given scenarios as the context of a rule in fact represents some kind of a type whereas its instances are the graphs and subgraphs the rule can be applied to.

Coming back to the genuine core of the graph approach we now describe the specification of graph transformations by means of rules. Graph rewriting is the technique of rule-based creation of a new graph out of an original graph. Such graph rewrite rules are formally written as $p\colon L \to R$, which means: a rewrite rule p transforms a pattern graph L into a replacement graph R. The pattern and replacement graphs are also called left hand side or right hand side of the rewrite rule, according to their logical position relative to the actual transformation operation, i.e. the arrow pointing from left to right. Possible is also the interpretation of the left hand side L being the pre-condition of the rule and the right hand side R describing the post-condition respectively [7]. The constructive matter of a rewrite rule, however, is to generate transformations by replacing occurrences of L in a given graph each by a copy of the right hand side R. A graph transformation leading from a pre-state graph G to post-state H according to a rewrite rule p is denoted by $G \overset{p(o)}{\Longrightarrow} H$. One complete graph transformation is therefore performed in the following three steps:

- *Find* an occurrence o_L of the left hand side L in the given graph G, i.e. our previously described subgraph isomorphism problem.
- *Delete* all vertices and edges from G matched by $L \setminus R$[1].
- *Paste* to the result a copy of $R \setminus L$, resulting in the derived graph H.

The following figure illustrates the individual steps of a complete graph transformation:



[1]Symbol $\setminus$ denotes the set-difference of L *minus* R, which results in all vertices and edges of L that do not also exist in R.

In general graph transformations problems may arise that nodes are removed but adjacent edges remain or even that all edges connecting a vertex are removed, leading to a *dangling node*, i.e. a vertex without connection to the rest of the graph. As such behavior may lead to surprising or even undesired effects in the application the following solution is known in graph theory and summarized in [7]: Formulating application conditions which exclude such situations as valid transformations and therefore avoid them in the first place. This is achieved by the *gluing conditions* of the so called double-pushout approach[1] to graph transformation. For further applicability and generality many advanced concepts extending the basic approach have been developed and shall be listed exemplarily:

- *Constraints* can be used to limit the number of connections of a vertex. More complex constraints could even deal with the (non-)existence of certain patterns, paths or cycles. With respect to our main focus on chemical graph transformation this also allows us to avoid the removal or creation of atoms in a molecule.
- *Application conditions* are used to restrict connections in the vicinity of the rule's left hand side.
- *Control conditions* may be used to represent *programmed graph transformations*. This means, for example, to insert conditions in the control flow of rewrite applications, i.e. change the order of rewrite rules to apply according to some condition.

After this introduction to the theory of graph transformation in the next section we describe the more specific usage of graph rewriting in Cheminformatics.

Chemical graph transformation

We already stated earlier in this work that chemical molecules are usually represented as chemical graphs in Cheminformatics. Considering the finding that graph transformations allow us to outline and describe some transformation from one graph G to another graph H, we might well think of chemical reactions as such graph rewriting. About thirty years ago the first applications arose using graph rewriting to model chemical reactions expressed by a graph grammar. After those so called *graph L-systems* [40] several approaches were introduced with their application domain ranging from whole organisms and tissues to organic molecules and even biomolecules like RNA. Reasons for the success of graph grammars in the mentioned context might be the ability of generalizations in the application on the one hand and certainly their intrinsic power of handling patterns.

The main focus of this work lies on reaction networks of organic molecules and therefore chemical reactions involving the latter. Nevertheless any chemical reaction can generally be represented as a sequence of applications of bond breaking and bond creation – atoms are generally conserved in chemical transformations [39]. Dedicated to

[1] The double-pushout is one of several algebraic approaches to graph rewriting in which the whole rule application is divided into two separate (pushout) steps, connected via an *interface* or *gluing* graph.

the approaches described in the previous section we will now illustrate examples of graph rewrite rules that were derived from chemical reactions.

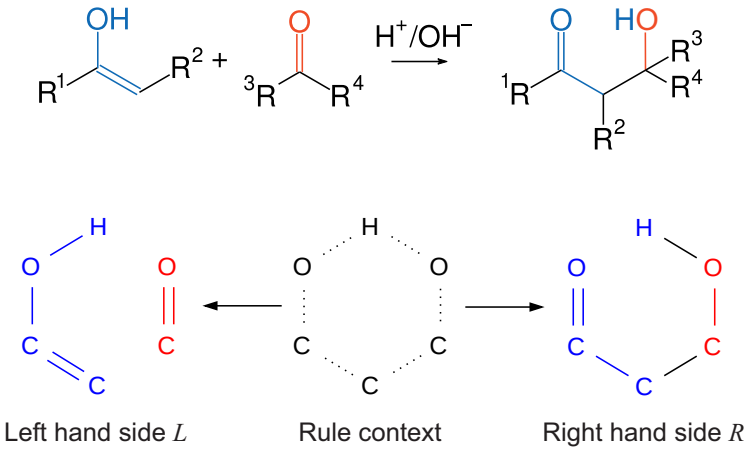


Figure 28: The chemical reaction *Aldol addition* depicted as graph rewrite rule. The first step of the whole reaction, the keto-enol isomerization, has been omitted such that the first aldehyde is already an alcohol. Residues R_1 and R_2 are unspecific and do therefore not occur in the rule. Residues R_3 and R_4 need to be restricted especially not to be oxygen O.

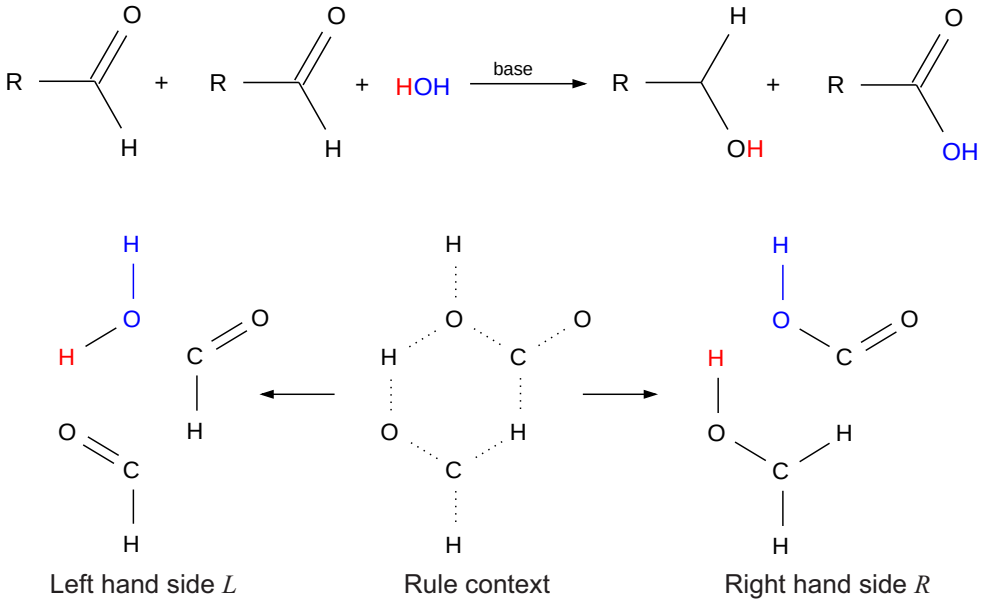


Figure 29: Rewrite rule of the *Cannizzaro disproportionation*. Basic environment transforms two aldehydes into a carboxylic acid and an alcohol. Of high importance is that the aldehydes may not posses an α-hydrogen on their residue R.

The two figures on the previous page delineated both, a *combination reaction* in terms of the *Aldol addition* and the *Cannizzaro disproportionation* as some kind of a *displacement reaction*. Also the two other main classes of chemical reactions (*decomposition reaction* and *exchange reaction* [41]) can certainly be written as graph transformations.

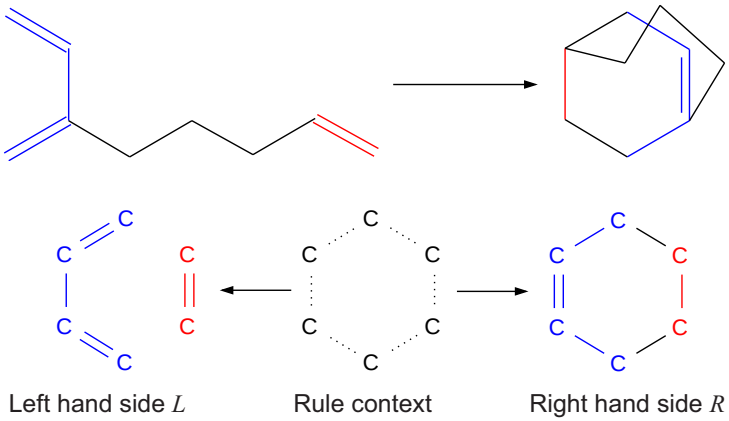


Figure 30: Rewrite rule of the famous Diels Alder reaction. Depicted is the intramolecular application of the reaction mechanism.

In contrast to ordinary (or better non-chemical) graph rewriting in Cheminformatics much more emphasis is placed on constraints and restrictions and of course valid rewrite rules to ensure the avoidance of dangling vertices or edges or even the creation or deletion of atoms. The obvious reason is that every graph transformation taking chemical compounds as input should return only valid and plausible chemical compounds in any case. However, the possibility of constraining graph transformations is also a reason why the graph grammar approach is so well suited for representing chemical reactions. Indeed graph transformation systems and applications can support the operator to a great extent in such a case by testing involved graphs and rewrite rules and verifying them respectively[1]. Nevertheless everyone creating new rewrite rules should be aware of the valuable responsibility.

2.4. Stochastic simulation

Simulation is a procedure to analyze systems that are too complex and stiff to describe them as deterministic mathematical formula. The simulated model in general represents some key characteristics of the considered system but does not cover all details, and is therefore at best an approximation. Typically simulations are used to describe dynamical system behavior as their mathematical formulation via ordinary differential equations ODEs is only solvable exactly for very small problems. In the use case of describing the dynamics of a possibly large chemical reaction network the calculation of an exact solution is not practical.

[1] Invalid or implausible molecules for instance contain carbon atoms establishing more than the normal four bonds to neighbor atoms.

Therefore stochastic simulation algorithms and methods were developed to handle and analyze those cases of chemical reaction networks. The first algorithm proposed in 1977 was Gillespie's Stochastic Simulation Algorithm SSA [5], also known as Gillespie Algorithm. The SSA bases on probability theory and is a variety of a dynamic Monte Carlo algorithm. The algorithm takes account of the discreetness and stochasticity of the simulated reaction network. For small particle numbers such as in cellular systems the SSA is even better suited than deterministic solutions because the probabilistic behavior of such systems is the core of the stochastic approach. It is furthermore an essentially exact numerical simulation method for well-stirred systems, which was also deduced by Gillespie. Due to this exactness the original SSA keeps track of every single reaction event in the simulated system and is therefore rather slow and impractical for many realistic problems [18]. Improvements have been achieved by reducing the number of tracked reaction events per simulation time step. Gillespie therefore proposed the tau leaping strategy in which Poisson random numbers are used to leap over many reaction events if the changes in molecule particle numbers are small enough. With an adequate selection strategy for the step size in the tau leaping simulation time can be reduced significantly although the results are still comparable to the original SSA.

We will now briefly describe the theoretical foundations of stochastic chemical kinetics and the SSA by summarizing [18].

We consider a system of N molecular species $\{S_1, ..., S_N\}$ interacting through M chemical reaction channels $\{R_1, ..., R_M\}$. The state of the system is described by the vector $X(t) \equiv (X_1(t), ..., X_N(t))$, where $X_i(t)$ is the number of particles of species S_i in the system at time t. We assume the system to be well stirred and in thermal (of course not chemical) equilibrium. The dynamics of reaction channel R_j is characterized by a *propensity function* a_j and a *state change vector* $\nu_j \equiv (\nu_{1j}, ..., \nu_{Nj})$, whereas $a_j(x)dt$ gives the probability, given $X(t) = x$, that one reaction R_j will occur in the next infinitesimal time interval $[t, t + dt]$, and ν_{ij} is the change in the molecular population of S_i induced by the particular reaction channel.

The dynamics of the system obeys the *chemical master equation* CME,

$$\frac{\partial P(x, t|x_0, t_0)}{\partial t} = \sum_{j=1}^{M} [a_j(x - \nu_j)P(x - \nu_j, t|x_0, t_0) - a_j(x)P(x, t|x_0, t_0)], \tag{2}$$

where $P(x, t|x_0, t_0)$ denotes the probability that $X(t)$ will be x given that $X(t_0) = x_0$. The CME is computationally intractable for all but the simplest models, so recourse is taken to the logically equivalent SSA. It is based on the fact that, which

$$a_0 \equiv \sum_{j=1}^{M} a_j(x), \tag{3}$$

then given $X(t) = x$, the time τ to the next occurring reaction is the exponentially distributed random variable with mean $1/a_0(x)$, and the index j of that reaction is the

integer random variable with point probability $a_j(x)/a_0(x)$. To advance the system from state x at time t, the SSA generates two random numbers $r_1$and r_2 uniformly in the unit interval and then takes the time of the next reaction to be $t + \tau$ where

$$\tau = \frac{1}{a_0(x)} \ln\left(\frac{1}{r_1}\right), \tag{4}$$

and the index for the next reaction to be the smallest integer j satisfying

$$\sum_{j'=1}^{j} a_{j'}(x) > r_2 a_0(x). \tag{5}$$

The system state is then updated according to $X(t + \tau) = x + \nu_j$ and this process gets repeated until some final time or condition is reached. The SSA is exact in the sense that the sample paths it generates are precisely distributed according to the solution of the CME given in equation (2).

2.5. Chemical kinetics

Chemical kinetics or reaction kinetics is the study of rates of chemical processes or rather reactions. Such a reaction rate of a particular reaction is intuitively defined as how fast this reaction takes place, which can also be said to be the *speed* of a reaction. This *speed* of a reaction, however, is the rate at which the concentrations of reactants (educts) and products change. Consider a typical chemical reaction

$$aA + bB \rightarrow pP + qQ$$

in which the capital letters A, B, P, Q represent educts and products and the lowercase letters a, b, p, q stand for the stoichiometric coefficient of the certain molecule species. The reaction rate v for this reaction occurring in a closed system under constant-volume condition and incorporating full mass balance is then defined as

$$v = -\frac{1}{a}\frac{d[A]}{dt} = -\frac{1}{b}\frac{d[B]}{dt} = \frac{1}{p}\frac{d[P]}{dt} = \frac{1}{q}\frac{d[Q]}{dt} \tag{6}$$

whereas $[X]$ denotes the concentration of species X and the $-$ sign is due to the decreasing concentrations of the educts A and B. Considering only species concentrations, without the stoichiometric factor, the reaction rate r can also be written as the so called *rate equation*

$$r = k(T)[A]^m[B]^n \tag{7}$$

in which $k(T)$ is the reaction rate constant that depends on the temperature and exponents m and n are called reaction orders and depend on the reaction mechanism.

Chemical kinetics also includes investigating conditions and factors that influence the reaction rate. Therefore we mention a few of those factors in the following list:

- *Nature of the reactants*: Depending on the bonding types of the involved molecules (covalent, ionic) and other physical issues, like rearrangement of bonds, the speed of reaction varies naturally.
- *Concentration*: As explained by collision theory the reaction rate increases with higher species concentrations. The more particles are present in the same system volume the more collisions among them occur.
- *Temperature & Pressure*: These physical properties tend to influence the rate of a reaction as they deliver more or less energy into the system possibly causing more or less collisions.

The next two sections are dedicated to the calculation of the reaction rate constant k of a particular reaction as k is required to calculate the actual reaction rate which is furthermore needed in our simulation of the network dynamics.

2.5.1. Arrhenius Equation

The Arrhenius equation is a formula for the temperature dependence of the rate constant of a chemical reaction and was first proposed in 1884. The formula is written as

$$k = A \cdot e^{-\frac{E_a}{R \cdot T}} \tag{8}$$

where T is the absolute temperature in Kelvin, R is the gas constant, E_a is the activation energy and A is called the pre-exponential factor. The equation is seen as an empirical relationship and also A and E_a have to be determined in macroscopic experiments although the activation energy is often approximated by simply subtracting the heats of energy of the products and the educts. As chemical reactions do not always occur on macroscopic scale and the mere energy difference of product and educt side of a reaction does not at all reflect the mechanism of a chemical reaction, the Arrhenius equation does not provide consistent reaction rate constants when using computational determined reactions.

2.5.2. Transition State Theory

The Transition State Theory TST is a theory of the rates of elementary reactions which assumes a special type of equilibrium to exist between reactants and activated complexes or transition state as depicted in the figure below. Although the TST has not been successful in its original goal of calculating absolute reaction rate constants, it provides a good description about how chemical reactions take place and therefore also provides more accurate calculation results than the Arrhenius equation.

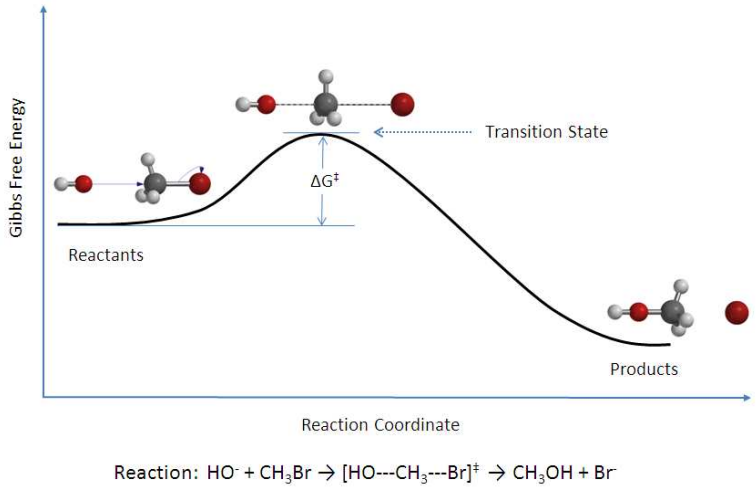


Figure 31: A reaction coordinate diagram showing the Gibbs free energy for molecules involved in a reaction. The transition state (or activated complex) lies at a saddle point of the potential energy surface.

In the Transition State Theory the reaction rate constant k can therefore be expressed by

$$k = \frac{k_B T}{h} e^{-\frac{\Delta G^{\ddagger}}{R \cdot T}} \tag{9}$$

in which k_B is the Boltzmann constant, h is the Planck constant, R is gas constant, T is the absolute temperature and $\Delta G^{\ddagger}$ is the Gibbs energy of activation describing the energy difference between the transition state of a reaction and the ground state of the educts [51].

Still the Transition State Theory has some limitations which are intended to be resolved by extensions to the theory. Improvements concern the traversal of the potential energy surface not to lead to unexpected products, and the quantum mechanical examination of the atomic nuclei.

3. Algorithm

3.1. Overall algorithm

For the overall algorithm we decided to implement the Concentration Sampling Network Generator CSNG proposed by Faulon and Sault [4]. Faulon and Sault have been working in petro chemistry for decades and therefore their studies are also settled there. Reaction network generation has been used by applications in combustion and refining for a long time up to now. However, formal techniques like our graph grammar based approach limit their applicability to real reaction systems because of their computational scaling. As a matter of fact it has been shown[1] that the number of generated reactions and intermediate molecule species generally scale exponentially with the number of reactants atoms.

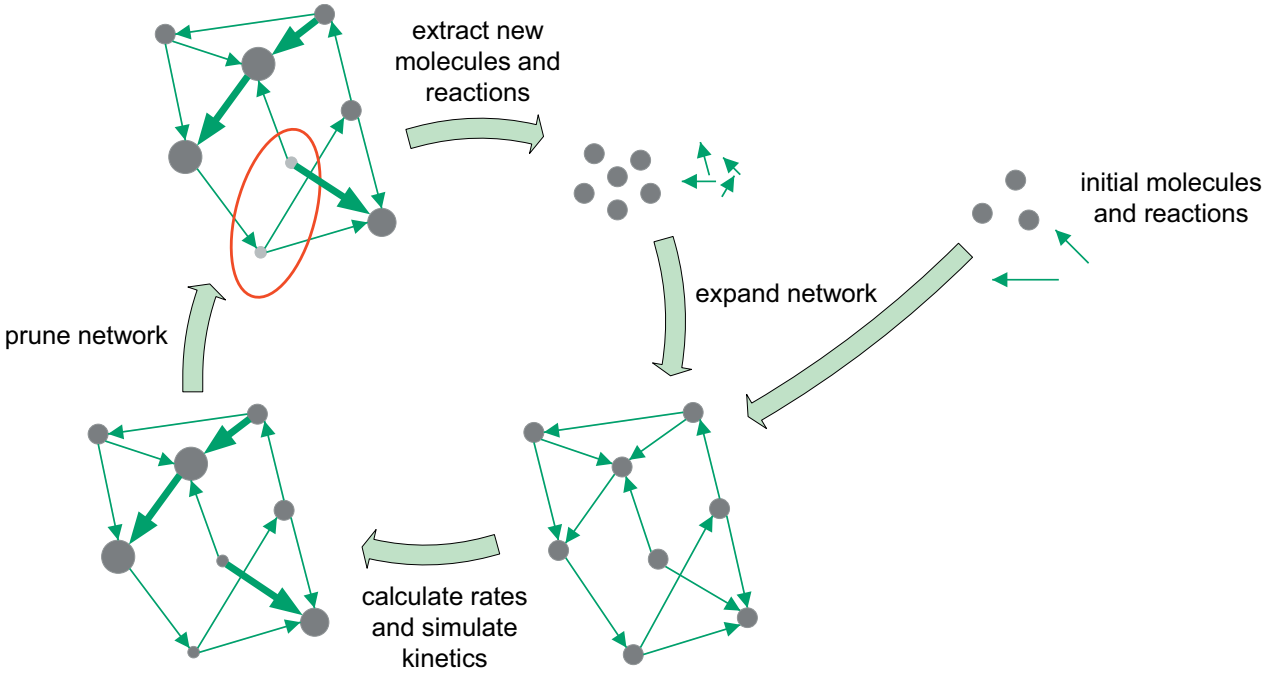


Figure 32: Starting from an initial set of molecules (grey circles) and reactions (green arrows) the reaction network is expanded. To avoid combinatorial explosion of the network a filtering step follows every expansion step. The pruning is performed based on reaction kinetics. Therefore first the reaction rate constants (proportional to the arrow thickness) are calculated using quantum mechanical models and secondly the dynamics of the reaction network is simulated using a stochastic method in order to remove molecules with the least concentration (proportional to circle radius) from the network.

To bypass the general limitation of exponential network growth several shrinking

[1]Details can be found in references 1−9 in [4].

strategies have been introduced to stabilize the network or at least prevent it from exploding by means of number of molecule species and reactions. According to Frenklach [13] there are five types of reduction strategies: 1) global reduction, 2) response modeling, 3) chemical lumping, 4) statistical lumping and 5) detailed reduction. Global modeling techniques are specific to a particular problem and therefore cannot be generalized as they transform a complete reaction scheme into a small number of global reaction steps. Response modeling techniques represent a mapping of model responses to model variables, whereas model responses are species concentrations and model variables are the initial environmental boundary conditions of the reacting system and parameters such as reaction rate coefficients. The solution method now returns model responses by expressing them by algebraic functions in terms of model variables. These algebraic functions (usually polynomials), however, are obtained by using experimental data, which implies that response modeling techniques again are specific to a certain problem. Chemical lumping and statistical lumping were developed for and are specific to polymerization-type reactions. The former is used to polymer growing reactions assuming that reaction parameters like the rate coefficient or thermo chemical parameters are more or less independent of polymer size. The latter model is used to describe polymer-polymer interactions like silica powder growth and metal oxide growth. The last reduction strategy is the detailed reduction. It comprises the identification and removal of non contributing reactions in the network. An effective implementation is to compare every reaction with a chosen reference reaction, for instance the rate limiting step or the fastest reaction, by means of the reaction rate. This approach is *a priori* applicable to any reaction system and therefore general enough to be picked up for a general reaction network generator.

Following the ideas and concepts of Ugi et al. and Frenklach, Faulon and Sault introduce a new method for reaction network generation in which the network generation and reduction is performed simultaneously. In the cited reference they outline four algorithms whereas the method performing the network reduction according to the species concentration is the most suitable. This because contrary to former network generation algorithms it was proven to scale in polynomial time (see equation (10)) and not in exponential time.

Time-Complexity of the full algorithm, whereas M_S is the maximum number of species in the network and M_C is the maximum number of network simulation steps:

$$M_S^2 \cdot \mathcal{O}(n^6) + (M_C + M_S + 1) \cdot \mathcal{O}(n^4) \tag{10}$$

Looking at figure 32 we can easily extract the encapsulated parts of our reaction network generator. In the following sections these parts are described in detail.

3.2. Reaction network generation

As mentioned in the general section above our algorithmic approach uses a formal technique to model its chemical universe or more concrete its reaction networks. The actual

formalism is a graph-based artificial chemistry and the underlying mechanism is one of the corner stones of our reaction network generator.

The talk is of graph transformations applied to chemistry. Through graph transformations we take a rather abstract look at a chemistry model. Molecular energies are not taken into account and therefore this part of the overall algorithm can be seen as mere graph grammar which is sufficiently for a computer scientific viewpoint.

A graph transformation per definition needs input or left-side graphs and graph rewrite rules to generate output or right-side graphs. By applying all known rewrite rules of a reaction network to all known molecule graphs contained in the reaction network the latter is expanded incrementally.

As we have learned earlier a graph rewrite rule consists of three parts: the left graph, the right graph, the context. Thus, in order to apply a graph rewrite rule within our reaction network, molecule graphs that fit into the left hand side of the rule need to be found. Therefore the subgraph isomorphism algorithms VF2 [16] and Ullmann [17] are used to identify molecule graphs or parts of them[1] respectively. Having identified the left hand side of the rewrite rule it can then be applied and leads to the right hand side molecule graphs by transforming vertices and edges in the defined way. These transformations represent establishing or breaking bonds between atoms in the chemical reality.

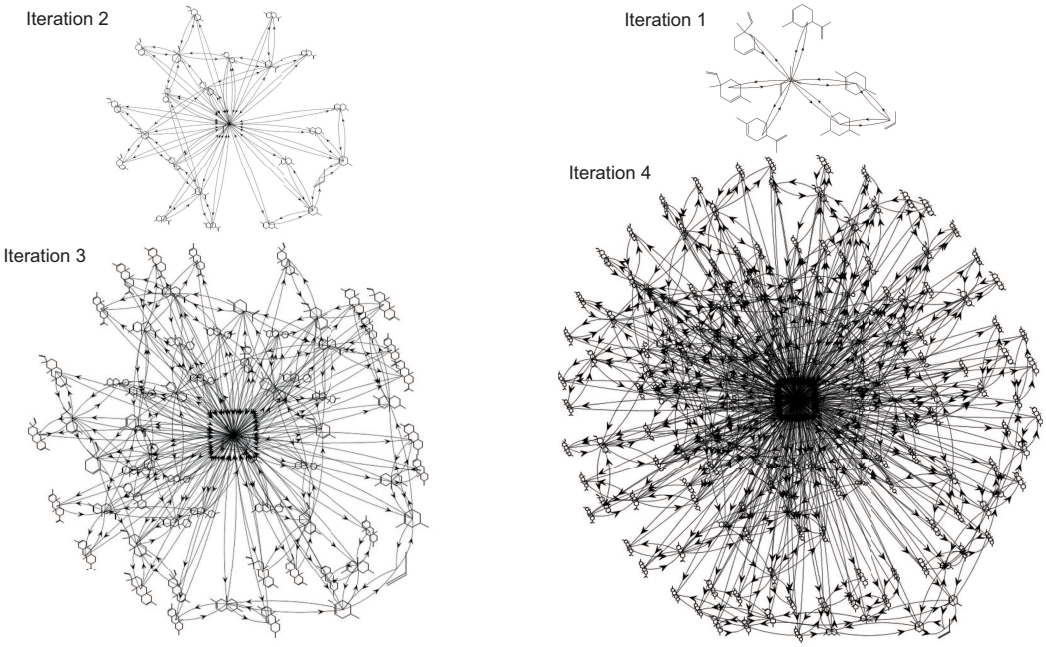


Figure 33: Combinatorial explosion of a Diels-Alder reaction network (4 iterations counter-clockwise). The bottom right graph shows iteration four with already 451 reactions and 160 molecule species.

[1]This is of even higher importance when allowing graph rewrite rules to be applied to one single left hand side graph which results in intramolecular reactions in a chemical sense.

Since the described procedure always leads to at least as many molecules as there were at the beginning and in the majority of all cases to much more molecules, this approach on its own obviously does not converge and therefore tends to cause combinatorial explosion with respect to the reaction network size.

3.3. Reaction cycle analysis

Besides the main focus of our application on the reaction network generation and simulation of network dynamics we also wanted to analyze the reaction networks from a different viewpoint than the pure chemical kinetics. Again a possibly evolutionary aspect in the reaction networks that we considered as use cases is crucial for the analyzed properties. We are talking about reactions of molecule species in the network forming cycles, because auto-catalytic or self-established cycles are supposed to play important roles in the chemical and prebiotic evolution.

In order to analyze the reaction network for such cycles our bipartite reaction network graph is transformed into a so called substrate graph. Whereas in the reaction network graph whole sides of reactions, i.e. the particular group of educts or the particular group of products, are connected to each other only via special reaction nodes , in the substrate graph there are only single molecule species connected to each other according to the reactions they are involved. This projection is called unidirectional or one-mode projection as there is no unambiguous conversion back to the reaction network graph.

For the following chemical reactions with molecule species S_1 to S_6 we exemplify the one-mode projection in the figure below.

$$\begin{aligned} S_1 &\rightarrow S_4 \\ S_2 + S_3 &\rightarrow S_5 + S_6 \end{aligned} \tag{11}$$

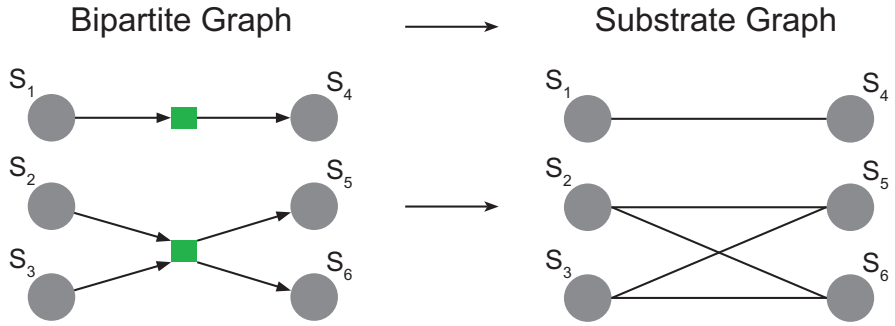


Figure 34: One-mode projection from bipartite network graph to substrate graph

The bipartite network consists of vertices for molecule species (grey circles), vertices for chemical reactions (green rectangles) and edges combining species to groups of educts and products and relating them to each other by connections to reaction vertices. The projection results in the substrate graph that only consists of species vertices and reaction

edges whereas the edges here connect all educts and products respectively involved in the particular reaction. Furthermore the substrate graph is undirected which is another reason for the irreversible process of the projection.

In such a substrate graph we now look for cycles in the following way representing some kind of a subgraph matching algorithm:

For each molecule species we start walking along all reactions developed by this species and proceed in the same way at every succeeding species and we avoid going back the path we came from. If we get back to the origin of our path we have finally found a cycle.

By counting the parts of our path, i.e. the vertices or molecule species occurring in the path, we are able to consider lower and higher limits for cycles to look for. However, one disadvantage of the former projection arises at this point. We lost the information about what reaction connects two molecules since two species might easily appear together in more than one chemical reaction, but are only connected by one single edge in the substrate graph. Nevertheless we can reduce this drawback by labeling the edges in the substrate graph and coding all reactions projecting onto an edge into its label. By post processing the outcome of this cycle analysis we can then reconstruct great parts of the involved reactions of a found cycle as mentioned in the results (chapter 6).

3.4. Stochastic simulation

To avoid now the inevitable combinatorial explosion of mere iterative network expansion a selection method has to be applied to the reaction network. Since we want to establish a chemically consistent model of a given reaction network the selection approach should preferably consider evolutionary aspects. With respect to the conceptual *survival of the fittest* this supposes that more accumulated molecule species are *fitter* than less accumulated ones. In mathematical terms this necessary information exactly corresponds to the molecule species concentration distribution whereas the concentration is nothing else than the accumulated species number divided by the total system volume.

In order to calculate the mentioned molecule concentration distribution the time evolution of the chemical reaction system has to be determined. We tackle this problem by simulating the chemical kinetics of a given reaction network. As simulation algorithm we use a stochastic method described by Gillespie [5]. In his work Gillespie introduced the Stochastic Simulation Algorithm SSA which is an essentially exact procedure for numerically simulating the time evolution of a will-stirred chemically reacting system. In theory this time behavior is described by the chemical master equation CME [18] which is a set of coupled ordinary differential equations ODEs that cannot yet be solved for every reaction system but only for a few very small and simple. In contrast the SSA is a Monte Carlo type procedure that leads in principle to the same results as the CME. However, as inherent for Monte Carlo type methods the precision of the particular output depends on the number of realizations (figure 35). As a matter of fact this can also lead to some sort of unsolvable problem when the calculation process takes too long to terminate.

Therefore we use an improved but approximate derivative method of the original SSA called the explicit Poisson tau leaping (or simply τ leaping). As the name suggests this algorithm does not simulate every single firing of reactions in the network but indeed leaps over several reaction firings by advancing the system by a time step τ that is small enough so that no significant changes in species numbers are missed. This so called leap condition is quite stiff but has been studied in detail by Gillespie et al. in for instance [21, 20, 18] and accelerates the simulation enormously.

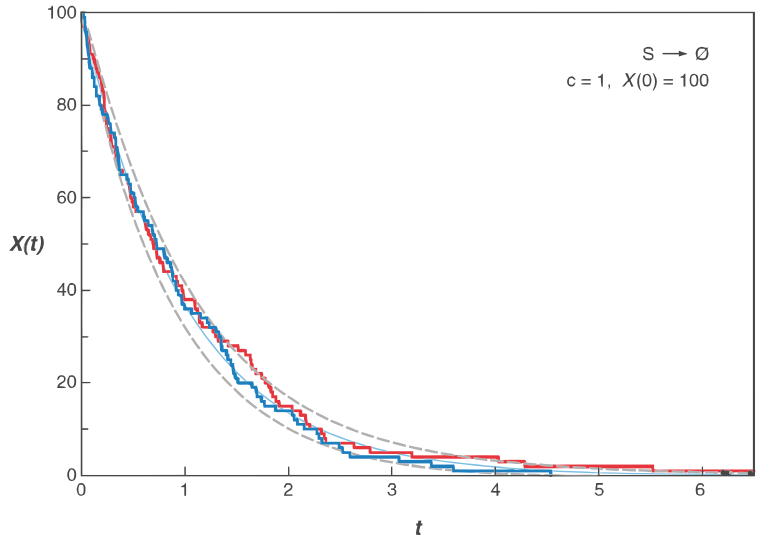


Figure 35: Simulating the simple isomerization reaction (Species S gets destroyed). The thin light blue line shows the correct theoretical solution. The two dashed gray lines show the one-standard-deviation envelope predicted by the CME. The red and blue jagged curves show two separate runs of the SSA.

For further acceleration in terms of simulation speed we additionally apply an adaptive selection algorithm for the step size τ. This means that we do not always advance the time by the same constant value τ but instead the step size depends on the slope of the calculated time trajectories. Therefore a fast reacting network with consequently high slopes in the trajectories is simulated in smaller time steps than a slow reacting system in which the curves of time evolution represent small slopes.

As the elements of our artificial reaction network are modeled based upon graph theory we first have to transform them into an appropriate form for the stochastic simulation environment. In the following sections this transformation in terms of the parts *initialization vector*, *stoichiometry matrix* and *propensity function* is described in detail. The last subsection is dedicated to the fact that we had to develop an own algorithm for terminating our simulation runs on quasi-equilibrium in terms of molecule concentration.

3.4.1. Initialization vector

The initialization vector has as many elements as there are molecule species in the reaction network. Every element gives the number of particles the simulation should start with.

In our application educts get assigned a high initial value upon which all other values are scaled and products get assigned the particle number *zero* as they first have to be created according to the particular reactions. Whether a molecule species belongs to the educts or is a product is also identified by the reactions in the network. Molecules that occur only on the right hand side in all involved reactions are therefore products and in contrast educts are molecules that occur on the left hand side of at least one reaction.

3.4.2. Stoichiometry matrix

The stoichiometry matrix N represents the compact form of stoichiometries of reactions. Supposed the reaction network consists of m reactions and n molecule species, the matrix has m rows and n columns. Every cell of the matrix represents the numerical stoichiometry of a particular molecule species in a particular reaction.

For the example reaction network below:

$$\begin{aligned} S_1 &\rightarrow S_2 \\ 5S_3 + S_2 &\rightarrow 4S_3 + 2S_2 \\ S_3 &\rightarrow 2S_4 \\ 3S_4 &\rightarrow S_5 \end{aligned} \tag{12}$$

the stoichiometry matrix can be written as:

$$N = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & -3 & 1 \end{bmatrix}$$

in which the columns represent the molecule species S_1 to S_5 and the rows represent the reactions in the same order as listed above. A cell value of -1 means that one molecule of a particular species is destroyed during the reaction and hence a value of 1 means that one molecule is produced.

Combined with the rate vector ν of a reaction network the stoichiometry matrix describes the rates of change of the molecule species S in the following compact equation:

$$\frac{dS}{dt} = N \cdot \nu \tag{13}$$

3.4.3. Propensity function

The product of the propensity function with dt gives the probability that a particular reaction will occur in the next infinitesimal time dt.

For almost every reaction − restrictions can be found in [22] − a propensity function can be derived from the deterministic reaction rate equation [18]. It takes into account the *law of mass action* and the probability that the molecules taking part in the reaction actually collide with each other.

Below we see example reactions where X denotes molecule species and c denotes reaction rate constants:

$$\begin{aligned} X_1 + 2X_2 &\underset{c_2}{\overset{c_1}{\rightleftharpoons}} 3X_2 \\ X_3 &\underset{c_4}{\overset{c_3}{\rightleftharpoons}} X_2 \end{aligned} \tag{14}$$

The particular propensity functions are listed as a_1 to a_4 whereas x denotes the number of particles of molecule species X:

$$\begin{aligned} a_1 &= \tfrac{c_1}{2} x_1 \cdot x_2(x_2 - 1) \\ a_2 &= \tfrac{c_2}{6} x_2(x_2 - 1)(x_2 - 2) \\ a_3 &= c_3 x_3 \\ a_4 &= c_4 x_2 \end{aligned} \tag{15}$$

Propensity functions only consider the reactants side of a reaction as it represents the crucial chemical input. The factor dividing the reaction rate constant c arises from the stoichiometric number for every involved species. For reaction a_1 in equation (15) the divisor 2 is the product of 1 for the factorial of only one molecule of species X_1 and 2 for the factorial of two molecules of species X_2. Furthermore to consider the collision probability of species X_1 and X_2 in the same equation a_1 the variables x denoting the particle numbers account for the function by answering the question: *How many ways are there to take one molecule out of all molecules for every contributing species (times their stoichiometry)?* As the stoichiometry for X_1 is one the particle number x_1 is taken without modification. For species X_2 we have to take two molecules to fulfill the stoichiometry. However, as we take out the first molecule of X_2 there are only $x_2 - 1$ molecules left for taking the second particle of this species, leading to the full propensity function $a_1 = \frac{c_1}{2} x_1 \cdot x_2(x_2 - 1)$.

This procedure for forming the propensity function can be continued and expanded respectively to represent a formal algorithm being applicable to every imaginable reaction.

3.4.4. Equilibrium termination

Our application requires deterministic behavior in the sense that every run finishes at least sometime but does definitely no run endlessly. On the whole this requirement mainly depends on the runtime of the stochastic simulation of the chemical kinetics. Therefore we strive for the earliest possible moment to stop the simulation but still get a valuable result out of it. In fact the most valuable result of a kinetics simulation exists at that time when there virtually are no more changes in molecule particle numbers.

More exactly we can define the last sentence by considering the following two cases:

- *Constant particle numbers of all molecule species:* This circumstance can easily be handled by comparing the current state of the simulated reaction network with the previous state in the last simulation step. If there are only insignificant differences between both states we can stop the simulation as we are in equilibrium.

- *Cyclic changes of particle numbers:* We hold a history of states of the simulated reaction network and reversely look for repetitions in the lists of particle numbers representing the states. As particle numbers are necessarily integral the equality of two states is well defined. The algorithm looks for two more repetitions of one state requiring the repeated states to be equal and equidistant by means of simulation time steps. Finding such a repeating pattern therefore also stops the simulation.

In case such equilibrium is not reached within the simulated reaction network, i.e. the molecule species concentrations do not converge or stabilize, the algorithm falls back to an arbitrarily high end time for the simulation within which all reaction networks should demonstrate a valuable result in the above declared sense. Therefore we can at least guarantee a runtime *much* shorter than infinity.

3.5. Reaction Rate Calculation

As discussed earlier the simulation of chemical kinetics is needed to be able to model reaction networks with evolutionary aspects. In our approach the evolutionary aspect of the network pruning algorithm is the reason for the simulation of the network dynamics. Exactly because of the necessity of a reasonable chemical simulation we also need reliable reaction rate constants marking the chemical paths between the molecule species in the reaction network. As we need reaction rate constants for every imaginable chemical reaction the determination of them can only be computational. The mentioned reliability can certainly be understood in many ways, however, our goal was not to calculate exact reaction rates, as this requires quite clever and detailed physical experiments and measuring, but instead we strove for deriving consistent reaction rates among the reactions. This means that rates of reactions that run at double speed in nature, should also have a factor of two between their calculated rates.

We achieve this by using quantum mechanical energy calculations [26] as they are supposed to incorporate physical details about molecular structure. As outlined in chapter 2.5 the reaction rate constant can be estimated once we know the molecule energies contributing in the reaction. The Transition State Theory TST was established in order to provide a more accurate description of consistent reaction rates than the Arrhenius equation can do [27] and is therefore used in our application.

Nevertheless the basis for the calculation of a reaction rate constant via the transition state theory is of course the knowledge of the transition state of a chemical reaction. This problem is adopted in the following subsections.

3.5.1. Transition State

In order to use the Transition State Theory for calculating reaction rate constants we need to derive the transition state of a particular reaction. As mentioned previously the transition state is not yet fully understood and therefore can only be estimated. We do so by following Fujita's proposal in [28] and derive the imaginary transition state[1] ITS from our chemical reactions. Since our chemical reactions are modeled as graph rewrite rules the ITS is like all contributing molecules represented as a graph.

Within the reaction network expansion process the intrinsic step is done in the application of the graph rewrite rule. There the educt molecules matching the left hand side of the rewrite rule are taken to create the product molecules according to the right hand side of the rule. Exactly in this moment all participating molecule species are known and, probably even more important, the atom-atom mapping is known. The latter means that for each atom on the educt side of the reaction we know which atom on the right hand side it corresponds to.

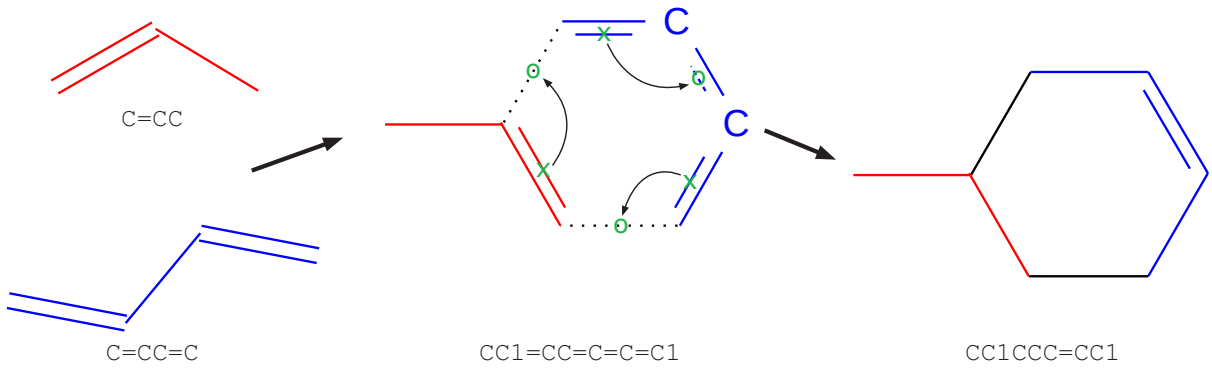


Figure 36: Example for a calculated imaginary transition state ITS of a Diels Alder reaction of the depicted SMILES strings. In the ITS the transition of the atom bonds is outlined via x symbols for broken bonds and o symbols for created bonds.

Taking all this information into account we can now build up the imaginary transition state by overlaying the educt molecules and the product molecules according to the rewrite rule so that every involved atom also exists in the ITS. Furthermore our algorithm sets up bonds of the highest occurring covalence between two atoms as proposed by Fujita. The result of this algorithm is exemplified in the above figure 36.

If atoms a_1 and a_2 are connected by a single bond within the reactants and the same atoms are connected by a double bond within the products the atoms a_1 and a_2 will also be double bonded in the ITS because the double bond has a higher covalence than the single bond.

[1] Fujita originally used the term *Imaginary Transition Structure*. However, we use the *state*-nomenclature since the mentioned theory TST is also called Transition *State* Theory.

3.5.2. Atom mapping

We have already learned about the atom mapping that exists during the application of the graph rewrite rules to the molecule graphs in the reaction network when the network is expanded. It appears in the moment when matching subgraphs of the left hand side molecule graphs for the particular rewrite rule are found and put together according to the rule to form the right hand side molecule graphs.

However, for reasons of software modularity we completely separated the reaction rate calculation from the remaining algorithmic parts network expansion and simulation. Thus we do not know anything about the atom-atom mapping when taking a particular chemical reaction as input. Therefore the reaction rate calculation has to derive the atom mapping for the given reaction on its own. For this task we use the subgraph isomorphism algorithm proposed by Ullmann [17] to find isomorphic matches between educts and products of the reaction. As shown in the following figure 37 the atom mapping leads to the information of corresponding atoms of both reaction sides.

This information is required when running quantum mechanical energy calculations because we advise the program `jaguar` to optimize the molecule and transition state geometry and `jaguar` identifies corresponding atoms through the given atom-atom mapping.

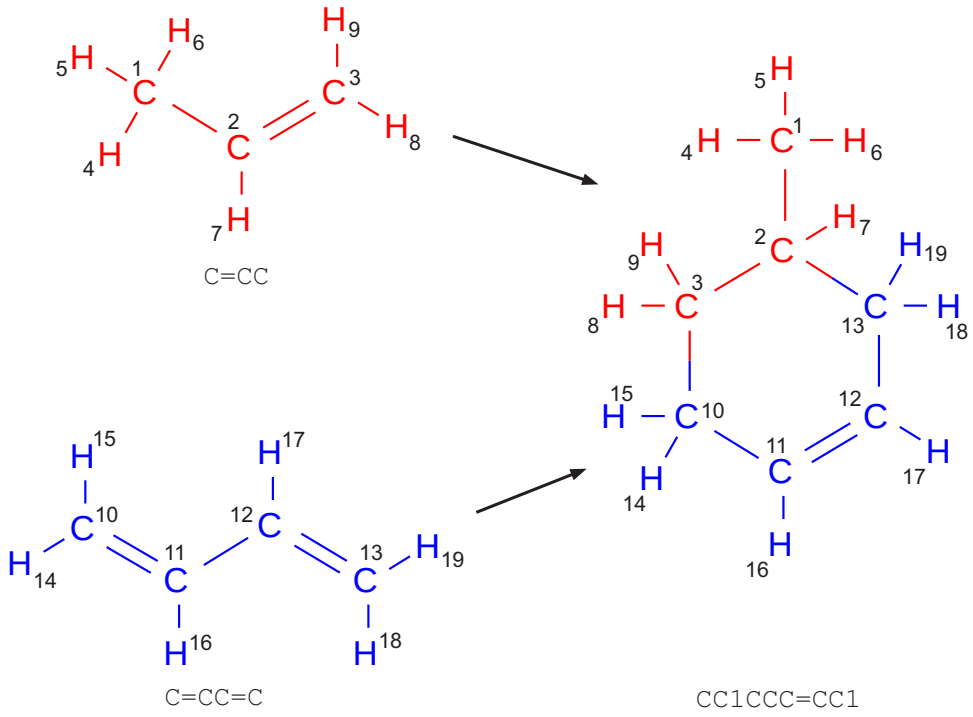


Figure 37: Example for an atom-atom mapping in a Diels Alder reaction of depicted educts and products. The molecules and atoms respectively are colored red and blue to indicate their overlay as seen in the product molecule according to the particular graph rewrite rule.

4. Interfaces & Data structures

In this chapter we describe the software interfaces to external components used in our application. The technologies and programming languages of these interfaces range from C++ libraries to command line calls of Perl scripts.

4.1. Graph Grammar Library

The graph grammar library GGL [10] provides various functionalities regarding molecular graphs and chemical graph transformations, such as parsing SMILES strings to graphs or subgraph isomorphism or applying graph transformations to molecular graphs. Internally it uses data structures of the Boost Graph Library BGL for storing and handling graphs and furthermore it uses the subgraph matching library sgm (internally VFLib 2.0) for finding rewrite rule patterns in host graphs. The package also contains the application `toyChem` [6], that implements parts of a chemical toy-universe in terms of expanding sets of molecule species within a chemical reaction network. Furthermore the library extensively uses the template mechanism of C++ which makes it flexible and adaptable for a great variety of related problems. The GGL is written in C++ and is mainly developed by Martin Mann (Albert-Ludwigs University Freiburg) and Christoph Flamm (University of Vienna).

Class ggl::chem::Molecule

The Molecule class is one of the core classes and holds the information about the molecular graph. It is defined as boost:: adjacency_list representing an undirected graph and storing properties for vertices and edges. The GGL consists of methods e.g. for generating a Molecule object out of a SMILES string or generating a unique/canonical SMILES string from a Molecule object.

```
typedef boost::adjacency_list<
        boost::vecS,                // store edges
        boost::vecS,                // store vertices
        boost::undirectedS,         // is an undirected graph
        Graph_NodeProperties,       // atom symbols etc
        Graph_EdgeProperties        // edge symbols etc
      > Molecule;
```

Class ggl::chem::Reaction

A Reaction object describes the essential information about a chemical reaction. Besides the reactant and product molecules in terms of their SMILES strings, the Reaction class also defines properties for the imaginary transition state and the reaction rate of the chemical reaction. Furthermore the string ID of the corresponding graph rewrite rule

that has led to the particular reaction, is stored as outlined in the following listing.

```
class Reaction {

public:
    /*! the identifier of the ggl::chem::ChemRule rule that was
     *  applied in this reaction. */
  std::string rule_id;

    //! the SMILES of the molecules involved in this reaction
  std::multiset< std::string > metabolites;

    //! the SMILES of the produced molecules of this reaction
  std::multiset< std::string > products;

    //! the rate of the reaction
  double rate;

    //! The SMILES of the transition state along the reaction.
    //! It is used to calculate appropriate reaction rates.
  std::string transState;
};

typedef std::set< Reaction > Reaction_Container;
```

Class ggl::chem::ChemRule

The ChemRule class represents a graph rewrite rule in terms of boost-graphs. It describes the left and the right hand side and the context of the rule due to deriving from superclass Rule. Static methods allow for checking chemical reason of the transformation.

```
template<> class ChemRuleT : public RuleT<> {

public:
    //! checks consistency of the rewrite rule (i.e. balanced
    //! electron changes and bond/atom labels)
  size_t isConsistent( void ) const;

    //! access the minimal imaginary transition state of this rule
  const TransitionState& getTransitionState() const;
};

typedef ChemRuleT<> ChemRule;
```

Class ggl::chem::LeftSidePattern

The class LeftSidePattern implements an interface of the subgraph matching library sgm to search for the left hand side of a given rewrite rule. Moreover it gives access to the graph automorphism of a certain rule and to the specified constraints that need to be considered when matching the pattern.

```
template < class RULE = Rule >
class LeftSidePatternT : public sgm::Graph_Interface {

public:
  virtual sgm::GA_OrderCheck getGraphAutomorphism( void ) const;

  virtual const std::vector< RuleConstraint > &
  getMatchConstraints( void ) const;
};

typedef LeftSidePatternT<> LeftSidePattern;
```

Function applyRules

The method applyRules is the main entry point for expanding a reaction network of specified molecule species and reactions in terms of specified graph rewrite rules. The declaration of the methods signature is outlined in the listing below.

```
void
applyRules( const RulePatternMap & rules
          , const SMILES_container & initialMolecules
          , SMILES_container & producedMolecules
          , Reaction_Container & producedReactions
          , const ReactionRateCalculation * rateCalc
          );
```

The method takes a set of initial molecules, a set of rewrite rules and optionally an object specifying a reaction rate calculation method as input and applies all rules to the set of molecules. The resulting molecule species as well as the thereby produced reactions are provided as output.

4.2. StochKit

StochKit [24] is an efficient, extensible stochastic simulation framework developed in the C++ language with the aim to make stochastic simulation accessible to practicing biologists and chemists. It provides Gillespie's popular stochastic simulation algorithm

SSA and a few optimized derivatives such as the tau-leaping methods. As a high-quality source for pseudo-random numbers the *Scalable Parallel Random Number Generator* library SPRNG is used internally. Furthermore the framework contains tools that make stochastic simulation more convenient. Among these tools are a java converter from an SBML file to the StochKit input format and MATLAB scripts to quantify differences in statistical distributions to verify results of the stochastic solver. StochKit is developed by Linda Petzold's group at the University of California, Santa Barbara.

Class Vector

The class Vector is a base class of StochKit that represents a linear array of real values. All mathematical operators, like the sum of two arrays or dividing all elements of an array by a constant value, are implemented to provide full usability in terms of mathematics. Vector objects are for example used to represent a state of molecule species numbers.

```
class Vector
{
public:
  Vector(int size, double initVal);

private:
  int mSize;
  double* mData;
};
```

Class Matrix

Class Matrix in the StochKit framework represents a two-dimensional array of real values. Again all mathematical Scalar-Matrix and Matrix-Matrix operators are implemented. Moreover not only single fields of the matrix can be accessed, but also whole rows or columns. Matrix objects are for example used to represent the stoichiometry matrix ν as explained in chapter 3.4.2.

```
class Matrix
{
public:
  Matrix(int rows, int cols, double initVal);

private:
  int mRows;
  int mCols;
  double* mData;
};
```

Class ReactionSet

A ReactionSet object contains all possible reaction channels in a system. Each reaction channel is characterized by the propensity function (a function pointer - see chapter 3.4.3 for details) and by the state change vector, that is calculated in each step according to the propensity function and the stoichiometric matrix ν.

```
class ReactionSet
{
public:
  ReactionSet(const Matrix& nu,
              const Matrix& dg,
              PropensityFunc prop,
              PartialPropensityFunc pprop,
              PropensityJacobianFunc propJac,
              EquilibriumFunc equilibrium);

  ReactionSet(const Matrix& nu,
              PropensityFunc prop);
};
```

Structure SolverOptions

By setting up a SolverOptions object the actual stochastic simulation is parameterized. The configuration is done by selecting simulation and output methods, like step-size selection, and setting various simulation parameters. Predefined values can be used by calling method ConfigStochRxn(**int** fixedoption) to create a SolverOptions object.

```
struct SolverOptions
{
  StepsizeSelectorFunc   stepsize_selector_func;
  SingleStepFunc         single_step_func;
  StoreStateFunc         store_state_func;
  double                 absolute_tol;
  double                 relative_tol;
  double                 epsilon;
  double                 initial_stepsize;
  int                    progress_interval;
  int                    StepControl;
};
```

Function StochRxn

The method StochRxn computes one realization of the stochastic simulation. The parameters are the initial vector, finish and end time, a ReactionSet and of course the options for the simulation. After the simulation has finished, a simulation history is returned, that can also directly be written into a text file.

```
SolutionHistory StochRxn(const Vector& x0,
                         double t0,
                         double tf,
                         const ReactionSet& reactions,
                         const SolverOptions& options);

void WriteHistoryFile(const SolutionHistory& hist,
                      const std::string& fileName);
```

Function CollectStats

The method CollectStats computes possibly many realizations of the stochastic simulation. Therefore the additional parameter runs has to be considered. After all realizations of the simulation have finished, a list of endpoint vectors[1] per run, is returned, that can also directly be written into a text file.

```
EndPtStats CollectStats(unsigned int runs,
                        const Vector& x0,
                        double t0,
                        double tf,
                        const ReactionSet& reactions,
                        const SolverOptions& options);

void WriteStatFile(const EndPtStats& stats,
                   const std::string& fileName);
```

4.3. OpenBabel

Open Babel [25] is a chemical toolbox designed to *speak* and *understand* the wide range of languages and formats of chemical data. It is an open source project that provides ready-to-use programs and libraries to handle molecular chemistry related data. Main achievements are the conversions between many different chemical file formats and the existing ports of the library and tools for actually all popular programming languages, such as C++, Java, Python, ...

[1] The endpoint vector only contains the last states in molecule species populations, not the whole history.

Class OpenBabel::OBConversion

The purpose of the OBConversion class is to convert chemical objects (derived from OBBase) from one format to another. It defines methods to directly convert whole files or strings and also methods to read a chemical object of a specified format first, manipulate it (e.g. add implicit hydrogen atoms) and afterwards write it in another format to some stream object.

```
OBConversion conv(&in,&out);    // istream/ostream objects

// setup conversion from SMILES to MOL format
if(conv.SetInAndOutFormats("SMI","MOL"))
{
  OBMol mol;
  if(conv.Read(&mol))
  {
    // ...manipulate molecule
  }
  conv.Write(&mol);
}
```

Class OpenBabel::OBMol

The OBMol or just molecule class is said to be the most important class in Open Babel. It is designed to store all information associated with a molecule that can also be read from some formats. OBMol defines quite lot methods for getting and setting information about a molecule and for adding and accessing included atoms and bonds.

```
OBMol mol;
double molWeight = mol.GetMolWt(true); // add hydrogens implicitly
OBAtom* atom = mol.GetAtom(0);         // access atoms
```

Class OpenBabel::OBForceField

Class OBForceField is the base class for molecular mechanics in Open Babel. Classes derived from OBForceField implement specific force fields (Ghemical, MMFF94, UFF) with specific working field and applicability. After selecting a force field it can be used to minimize a molecule's structure and calculate the energy of the structure.

```
OBMol mol;
OBForceField* pFF = OBForceField::FindForceField("Ghemical");
if (!pFF)  // exit...

if (!pFF->Setup(mol))
  cerr << "ERROR: could not setup force field." << endl;
double molEnergy = pFF->Energy();
```

4.4. CORINA

CORINA is a 3D structure generator for small and medium sized molecules. Its robustness, comprehensiveness, speed and performance helped CORINA to establish as a world-wide recognized quasi-standard in industry and academia. This significance in generating three-dimensional molecular models of high quality has also led to widespread usage in well-known pharmaceutical and chemical companies, such as Symyx or the NCI/NIH. The latter also provides web services utilizing CORINA for various conversions and generation of molecular structures.

The web services of the National Cancer Institute NCI are centralized as *CADD Group Chemoinformatics Tools and User Services* CACTUS. The web service using CORINA internally is called *Chemical Identifier Resolver*[1] and allows one to convert some chemical structure identifier into another representation. The structure identifier might be a SMILES string, a Standard InChI or a chemical name like *sodium chloride* or *benzene*. Output representations are for instance graphic images or the MDL structure-data file SDF. A SDF consists of the 3D coordinates and adjacency information of a set of atoms forming one or more molecules. To convert now a SMILES string into the three-dimensional molecular structure of a molecule, which we suppose to be the main use case, the following URL calls the corresponding NCI web service.

The SMILES string for the small molecule methanol is simply CO. Therefore the query URL is:

`http://cactus.nci.nih.gov/chemical/structure/CO/sdf`

which results in a SD File with the following content:

```
CH4O
APtclcactv06011015392D 0    0.00000      0.00000

  6  5  0  0  0  0  0  0  0  0999 V2000
    3.4030    0.2500    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000    0.0600    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.5369   -0.2500    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.7130   -0.2869    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.9399    0.5600    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.0930    0.7869    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  3  1  0  0  0  0
  2  3  1  0  0  0  0
  1  4  1  0  0  0  0
  1  5  1  0  0  0  0
  1  6  1  0  0  0  0
M  END
$$$$
```

[1] Web page `http://cactus.nci.nih.gov/chemical/structure/documentation`

4.5. Jaguar

Jaguar is a high-performance *ab initio* electronic structure package for both gas and solution phase simulations [26]. The particular strength in treating metal containing systems make it one of the most practical quantum mechanical tools for solving real-world problems.

Even though molecular mechanical methods, like the force field calculations, have improved in the last decades, they are still approximations and their calculation results extremely depend on the particular model and parameterizations. Therefore important research questions remain and cannot be answered without examining in detail a molecule's electronic structure. In conclusion high-level quantum mechanics is still the most accurate and most direct way to study even challenging systems, despite the increased computational cost. Therefore an efficient quantum mechanical program is indispensable for research in reactive chemistry, chemical reaction systems or phenomena that require precise energetics such as reaction rate calculations.

The `jaguar` package consists of tools and scripts performing the actual calculation. At first the following environment variables need to be set on the UNIX system:

```
export SCHRODINGER=/path/to/the/directory/of/your/jaguar/installation
export SCHRODINGER_TMPDIR=/path/to/some/scratch/directory
```

In the next step the input files for `jaguar` have to be created, which is described in detail in appendix D. In general an input file starts with a section of parameters adjusting the quantum mechanical calculation and telling `jaguar` what exactly to calculate, followed by the specification of the molecular data. The molecular data `jaguar` needs is only a list of all involved labeled atoms with 3D coordinates – adjacency information, bond types and all other information are extracted from the spatial arrangement of the atoms.

Depending on the commands specified in the input file `jaguar` can then minimize/optimize a molecular structure, derive a transition state for a chemical reaction or calculate various energy types and other physical parameters for the given set of molecules. The following commands can be used to run `jaguar` and extract specific results from the calculation:

```
$SCHRODINGER/jaguar run –WAIT <input filename>
$SCHRODINGER/jaguar results <option> <input filename>.out
```

The input filename has to be specified without an extension when calling `jaguar`, i.e. suppose the input file is called *methane.in*, then the run-command has to be executed with parameter *methane* only. The "-WAIT" parameter tells the command to return not until finished. At the results command of `jaguar` an option has to be specified, for instance "-gibbs" to get the Gibbs free energy or "-coords" to get the optimized geometry, and the output file of the calculation has to be given as *methane.out* in our example.

5. Software

5.1. StochGraphLab

5.1.1. Architecture overview

Hereby we briefly describe the overall architecture implemented in the presented work before going into detail in the following sections. As depicted in figure 38 our main application covers the parts: reaction network generator, reaction rate calculation and links to a web service. Whereas the web service for embedding molecular structures with CORINA and the rate calculation server are only peripheral services, the reaction network generator represents the actual application of simulating chemical reaction networks. As describes earlier this simulation further consists of expanding the reaction network by applying graph transformations to a set of molecule species, simulating the network dynamics in terms of concentration distribution among the molecule species and the essential network pruning to shrink the network size before expanding again.

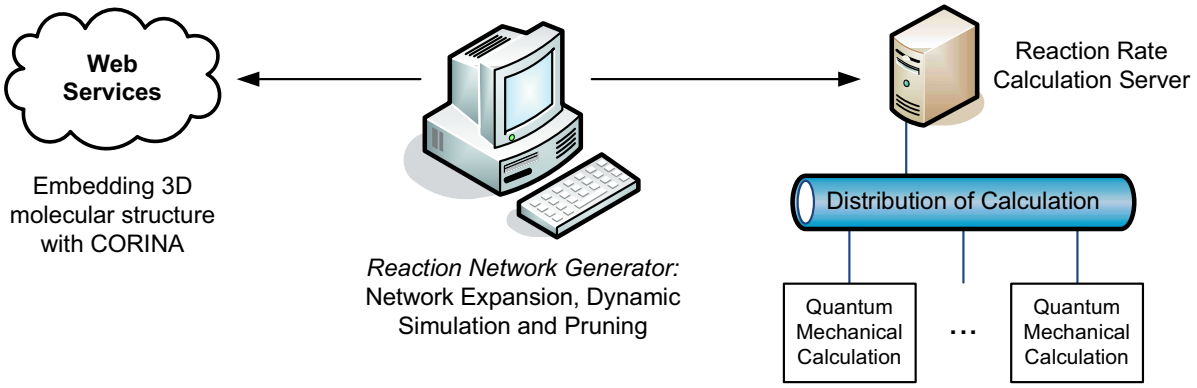


Figure 38: Rough overview of the whole application consisting of the reaction network generator, web services and a reaction rate calculation server.

5.1.2. Application

Our main application is called StochGraphLab, which stands for *Stochastic Graph based artificial Laboratory*. The single parts of the abbreviation already anticipate the core ideas and approaches of the software. In generalization of the aims of the presented thesis, we do not limit our software to special prebiotic reaction networks, but we indeed support every imaginable reaction network. Therefore our software represents some kind of a *laboratory* were chemists put some initial molecules into a vessel, create some environmental circumstances and look at the outcome. Because we model molecules as graphs and chemical reactions as graph transformations, we further call the software *graph based*. Finally to demonstrate real chemical and physical behavior the application runs stochastic simulations on the reaction networks – that is where the term *stochastic*

comes from. Altogether the presented software represents a fully featured chemical reaction network generator as proposed by Faulon and Sault [4]. Additionally our software analyzes cycles occurring in the reaction network, helping to identify possible autocatalytic cycles. In the figure below the overall application flow of the StochGraphLab is outlined.

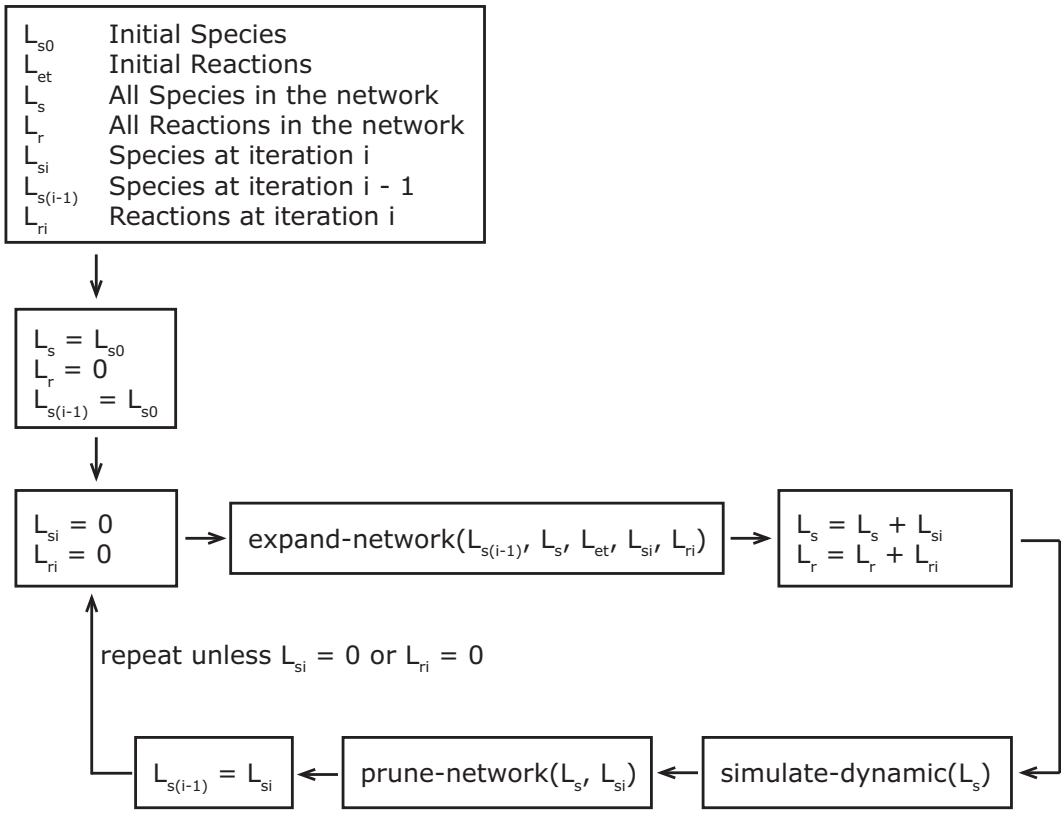


Figure 39: Main application flow of the Concentration Reaction Network Generator in the StochGraphLab. After initializing the used variables the main loop of expanding, simulating and pruning the reaction network is entered. If no new molecule species or reactions are produced, the application run has finished.

StochGraphLab is written in C++ and is developed using the Eclipse Platform 3.4.0 with the CDT (C/C++ Development Tools) PlugIn and Cygwin Tools on Windows XP with the GNU Compiler Collection GCC Version 4.3.2.

In the subsequent sections the individual modules and functionalities of the StochGraphLab are described. On the one hand to present each contained module separately and on the other hand to give the reader a more detailed insight in the applications parts.

5.1.3. Modules

Species

The class Species represents a molecule species and stores some chemical properties, such as the molecule weight. Furthermore the corresponding SMILES string and the molecular graph are stored at a Species object.

```
class Species
{
public:
  std::string name;                 // SMILES string
  ggl::chem::Molecule molecule;     // molecular graph
  double concentration;             // species concentration

  double getWeight() const;
  double getEnergy() const;

  // Parse species list from input file.
  static void
  ParseSpeciesList(const std::string &input,
                   SpeciesList &retList);

  // Convert a list of Species into a list of SMILES strings
  static void
  SpeciesListToSmilesContainer(const SpeciesList &species,
                               SMILES_container &smiles);
}
```

NetworkOperator

Class NetworkOperator defines static methods for operations on reaction networks. The main methods are depicted in the following listing.

```
class NetworkOperator
{
public:
  static void ExpandNetwork(...);
  static void PrePruneNetwork(...);
  static void PostPruneNetwork(...);
  static void AnalyzeNetwork(...);
}
```

The method ExpandNetwork expands a network by applying graph rewrite rules to a set of molecule graphs of the network. As outlined in the listing, the rewrite rules are converted to a RulePatternMap object before calling the GGL-method to expand the reaction network. Parameter newRules is currently always empty, because at the moment no new graph rewrite rules are derived from the species in the network.

```
void ExpandNetwork(SpeciesList &allSpecies,
                   RuleCollection &rules,
                   SpeciesList &newSpecies,
                   RuleCollection &newRules,
                   Reaction_Container &producedReactions)
{
  SMILES_container targetSmiles;
  SMILES_container producedSmiles;

  Species::SpeciesListToSmilesContainer(allSpecies, targetSmiles);

  // holds the left side patterns of the given rules
  RulePatternMap rulePattern;

  // call to the graph grammar library GGL
  applyRules(rulePattern, targetSmiles, producedSmiles,
             producedReactions, NULL);
}
```

The purpose of both methods PrePruneNetwork and PostPruneNetwork is actually the same, i.e. shrinking the network in terms of size by removing molecules species. *Note, that also reaction objects are removed from the network, if a species, that needs to be removed, is a reactant of this reaction. Hence, other species that are created only by this reaction are also removed from the network.* The methods implement different pruning strategies and are, for the reason of optimizations, called before the stochastic simulation (PrePruneNetwork) and afterwards (PostPruneNetwork) respectively.

```
void PrePruneNetwork(SpeciesList &network,
                     SpeciesList &newlyAdded,
                     Reaction_Container &reactions);
void PostPruneNetwork(SpeciesList &network,
                      SpeciesList &newlyAdded,
                      Reaction_Container &reactions);
```

The method AnalyzeNetwork searches for cycles in the reaction network in terms of reaction paths starting at some molecule species and finally getting back to the origin. The algorithm for this search is a brute-force-type and requires the reaction network to be transformed into the *substrate graph* form in which only molecule species (vertices) and reactions (edges connecting molecule species) exist, but no extra vertices for reactions

themselves, as depicted in figure 17. The algorithm minds minimum and maximum steps for a cycle in the search and is able to ignore or consider directions of the reactions. In the code a cycle is represented as list of unique molecule species and a corresponding list of all reactions this species is involved in the particular cycle: map<Species*, set<int> >.

```
void AnalyzeNetwork(SpeciesList &network,
                    const Reaction_Container &reactions,
                    vector< map<Species*, set<int> > > &allCycles);
```

QuantumMechanics

The class QuantumMechanics provides methods and constants useful for quantum mechanical calculations. As outlined in the following listing the only method is CalculateRate that takes a chemical reaction object as input and returns the reaction rate as real value.

```
class QuantumMechanics
{
public:
  static double CalculateRate(const ggl::chem::Reaction &reaction);
  static const double R = 8.314472;    // [R] = J / (mol*K)
}
```

We have already mentioned that all reaction rates are calculated on a separate calculation server and therefore need not be computed in the StochGraphLab application itself. Because the rate calculation server is a TCP server the communication between the calculation server and the main application uses the TCP/IP-protocol.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

double CalculateRate(const Reaction &reaction)
{
  ...
  // open TCP socket
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  // connect to server
  connect(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
  // post request to server
  int n = write(sockfd, request.c_str(), request.size());
  // read answer from server (string encoded reaction rate)
  n = read(sockfd, buffer, sizeof(buffer));
  ...
}
```

The format of the request string was chosen to contain educt-side, transition state and product-side of the reaction in terms of SMILES strings delimited by a space " ", and the single molecules in a group delimited by a dot. The request string for a simple Diels Alder reaction is therefore:

CC(=C)C=C.CC=C ␣ C[C]1=[C][C]=[C]=[C](C)=[C]1 ␣ CC1CCC=C(C)C1

The reaction rate returned by the calculation server is simply a real value that only needs to be parsed in the application.

DynamicSimulator

Class DynamicSimulator defines the method SimulateNetwork for running a stochastic dynamics simulation on the given reaction network. Therefore the reaction network, represented by the list of known species and reactions, first has to be converted into the data structures usable by the StochKit methods described in chapter 4.2. According to the algorithm outlined in 3.4.2 the stoichiometry matrix nu is derived from the given reaction network. Within this method for every reaction in the reaction network also the reverse reaction is considered and furthermore simulated, in order to simulate also the backward direction of a reaction path. The private method GetPropensity serves as function pointer defining the propensity functions of the reaction network, as described in chapter 3.4.3. The initial vector x0 specifying the particle numbers at the beginning of the simulation is filled according to the simulated concentration of the last iteration of network generation if the species is already known. Otherwise the initial number is set to 0 for species being only reaction products in the whole network or to concentration 1 respectively if the species is a reactant in at least one reaction.

```
class DynamicSimulator
{
public:
  static void SimulateNetwork(SpeciesList &network,
                              const Reaction_Container &reactions,
                              int iteration);

private:
  static Vector GetPropensity(const Vector& x);
}
```

Before starting the actual simulation run, the stochastic solver of StochKit is configured to run in the *adaptive mode* of explicit tau-leaping, which means automatic step-size selection and switching between exact SSA and tau-leaping, depending on the speed of changes in the reaction network.

```
SolverOptions opt = ConfigStochRxn(0);    // 0 ... adaptive mode
```

Furthermore the final time for the simulation is set to "-1", which indicates StochKit to end the simulation when detecting a steady state. This behavior was implemented in StochKit within this thesis and is **no** standard feature.

```
ReactionSet rxns(nu, GetPropensity);
SolutionHistory hist = StochRxn(x0, 0, TimeFinal, rxns, opt);
```

Resources & Miscellaneous

The Resources class holds static members allowing the StochGraphLab application to be parameterized. Moreover the parameterization data can be accessed from all modules in the application. All members are initialized in the main method of StochGraphLab. Module Miscellaneous defines general purpose helper methods for the StochGraphLab, such as checking the rewrite rules for chemical correctness or reading the various input files.

5.1.4. Input file formats

In this section the different input file formats to the StochGraphLab application are described. Every format is described briefly and exemplified for clarification.

Initial species list

The initial species list is an ASCII file specifying each molecule species represented by its SMILES string in an own line. The following listing exemplifies two alkene molecules.

```
C=C(C)C=C
C=C(C)
```

Initial species concentration

An ASCII file for specifying an initial concentration for molecule species used for the stochastic simulation. Each line contains of the canonical SMILES string for identifying the species and, separated by a tabulator character, the corresponding initial concentration as real value.

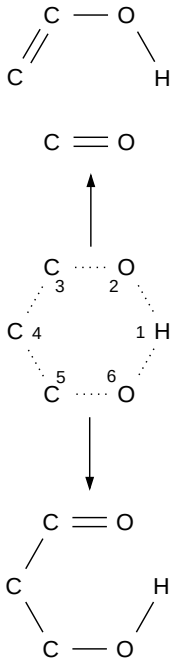
```
C=C(C)C=C \t 0.54
C=C(C) \t 0.02
```

Graph rewrite rules

As file format for representing graph rewrite rules the *Graph Modeling Language* or *Graph Meta Language* GML was chosen [11]. GML is a hierarchical ASCII-based file format for

describing graphs. As GML supports attaching arbitrary information to graphs, nodes and edges, it can also be used to store information about graph transformations.

```
rule [
  ruleID "aldol addition"
  left [
    edge [ source 1 target 2 label "-" ]
    edge [ source 2 target 3 label "-" ]
    edge [ source 3 target 4 label "=" ]
    edge [ source 5 target 6 label "=" ]
  ]
  context [
    node [ id 1 label "H" ]
    node [ id 2 label "O" ]
    node [ id 3 label "C" ]
    node [ id 4 label "C" ]
    node [ id 5 label "C" ]
    node [ id 6 label "O" ]
  ]
  right [
    edge [ source 1 target 6 label "-" ]
    edge [ source 2 target 3 label "=" ]
    edge [ source 3 target 4 label "-" ]
    edge [ source 4 target 5 label "-" ]
    edge [ source 5 target 6 label "-" ]
  ]
  constrainNode [
    id 5
    op =
    count 1
    nodeLabels [ label "O" ]
  ]
]
```



The above listing outlines the notations for left and right hand side and context of a rewrite rule. Furthermore a constraint, as suggested in the theory chapter, is defined, restricting the number of oxygen atoms adjacent to the carbon atom with ID 5 to be exactly one (i.e. the one oxygen atom with ID 6).

Reaction rate table

The reaction rate table file serves as a mapping of rewrite rule IDs to rates and reverse rates. As only rules can be addressed as a whole (i.e. every reaction a rule generates) and not single instances of reactions, this possibility of setting reaction rates only makes sense for testing purpose.

```
aldol_addition \t 0.314 \t 1.3e-12
```

5.2. Reaction Rate Calculation Server

The reaction rate calculation counts among the core modules of the whole application because of its importance for running relevant stochastic simulations. As outlined in the rough overview of figure 38 the rate calculation is implemented as TCP/IP server in the script language Perl. The interface of the rate calculation is the SMILES string encoded reaction including the educts, products and optionally the transition state as described in the section QuantumMechanics of the StochGraphLab. Additionally also the reaction temperature can be specified, but is currently not considered as mentioned later on. For the functionality the Perl script uses particular Perl modules besides the standard modules warnings and strict.

```
use LWP::Simple;
use IO::Socket;
use Net::hostent;

use Chemistry::File::SDF;
use Chemistry::InternalCoords::Builder 'build_zmat';
use Chemistry::ReactionGrammar::Mol;
use Chemistry::ReactionGrammar::Reaction;
use Chemistry::ReactionGrammar::Mapper::CSL;
use Chemistry::ReactionGrammar::RewriteRule;
```

Module LWP::Simple contains methods for posting web requests and fetching answers from web services. TCP server functionality is covered by the modules IO::Socket and Net::hostent and is setup with the following constructor code.

```
my $server = IO::Socket::INET->new(Proto     => 'tcp',
                                   LocalPort => $PORT,
                                   Listen    => SOMAXCONN,
                                   Timeout   => 60*60*24,  # secs.
                                   Reuse     => 1);
```

The modules contained in the Chemistry package are provided by my colleague Heinz Ekker and do on the one hand represent wrappers around the well known PerlMol package [52] and on the other hand provide representations for reactions and the atom-atom mapping in particular reactions.

After receiving the input reaction string the single molecule SMILES strings are parsed and converted to the respective SD File content, containing the 3D structural information, using the CACTUS web service of the NCI. To post the web request the SMILES string of a molecule has to be URL-escaped.

```
my $content = get(
    "http://cactus.nci.nih.gov/chemical/structure/$SMILES/sdf");
```

As the molecules are converted separately, due to errors in the CORINA web service[1], but nonetheless needed in a common spatial context for the latter quantum mechanical calculations, they are simply placed along one axis at a distance of 100 Angstrom Å per 50 atomic mass units amu. The atoms in the resulting molecules are relabeled using the elementary symbol and a consecutive number, required by the quantum mechanics program `jaguar`.

The previous step is applied to the reactants and the products first. If a transition state of the particular reaction is passed to the calculation server as well, it can also be converted to the structural data. Otherwise the transition state is deduced by deriving a graph rewrite rule from the reaction and converting the completely expanded context of the rule back to a molecule. The following listing shows the conversion of SMILES string encoded molecules to Chemistry::Mol objects.

```
my @gml_educts = Chemistry::Mol->parse($educts_SMILES_string,
   mol_class => 'Chemistry::ReactionGrammar::Mol',
   format => 'smiles')->separate;
my @gml_products = Chemistry::Mol->parse($products_SMILES_string,
   mol_class => 'Chemistry::ReactionGrammar::Mol',
   format => 'smiles')->separate;
```

Using the converted molecule objects a reaction object is created. During the determination of the mapping the atom identifiers of the reactant atoms are written to the corresponding atoms of the product molecules.

```
my $gmlreaction = Chemistry::ReactionGrammar::Reaction->new(
     educts    => \@gml_educts,
     products => \@gml_products
   );

# determine the atom-atom mapping of the reaction
my $mapper = new Chemistry::ReactionGrammar::Mapper::CSL(
                    remember_splits => 1);
$gmlreaction = $mapper->map($gmlreaction);
```

If the finally derived transition state has the same number of atoms as the educts or products it can be considered for further computations. Now the atom-atom mapping between educts and transition state and between transition state and products is examined. If the transition state is unusable the mapping is only calculated between educts and products. This mapping information is needed for the quantum mechanics calculation when creating the input files for `jaguar`, because it requires corresponding atoms of the molecules to occur in the same order in the input file (see appendix D for examples).

[1]The CACTUS web service places separate molecule to close to each other if they are larger than about 200 amu. The potentially leads to errors in calculations of `jaguar` regarding physically unreasonable atomic coordinates.

A `jaguar` input file contains a general section containing job commands and parameters and sections for the molecular data represented by enumerated atoms and their 3D coordinates. Within the whole rate calculation various input files are created in order to compute the following information:

- *Optimize geometry of the transition state*
- *Optimize geometry of the educts*
- *Calculate the Gibbs free energy of the transition state*
- *Calculate the Gibbs free energy of the educts*

Finally the reaction rate constant is calculated applying the Transition State Theory as outlined in 2.5.2. Additionally the rate constant is divided by the Avogadro constant N_A to regard the *system volume*. Before sending the result back to the client application the rate is also cached in memory and in a persistent file to reduce the server response time for another request of an already calculated reaction rate constant.

5.3. Tools

Besides the main application StochGraphLab to run the actual simulations of a chemical reaction network, there are developed a few additional tools for the purpose of preparing test networks and helping to finally evaluate results. Those tools are briefly described in the subsequent sections.

5.3.1. ChemkinLib

CHEMKIN is a commercial Cheminformatics software developed by the Sandia National Laboratories [54]. It is used to solve and describe reaction kinetics and is mainly used for combustion processes. CHEMKIN also defines a thermodynamic database and a file format in which reaction networks can be expressed.

The ChemkinLib is written in Java and provides access to CHEMKIN files through a parser and corresponding classes representing the file content. It also contains a test application to show the usage of the library (described in detail in appendix B). Internally the ChemkinLib uses the Chemical Development Kit library CDK [53] for chemical data representations.

The class ArrheniusEquation represents the kinetic parameters in a CHEMKIN file and allows to calculate the reaction rate constant from the given parameters.

```
public class ArrheniusEquation
{
  public double A, T, b, E;
  public EnergyUnit unit;

  public double GetReactionRateConstant();
}
```

The class KineticReaction extends the Reaction class defined in the Chemistry Development Kit CDK package [53] and adds the reaction rate and a string representation.

```
public class KineticReaction extends Reaction
{
  protected ArrheniusEquation rate;
  protected String OriginalStringRepresentation;
}
```

A CHEMKIN file is represented as ChemkinFile object that covers all information available in the particular file. A ChemkinFile can be initialized directly from a CHEMKIN file by using the parser.

```
public class ChemkinFile
{
  public String FileComment;
  public Vector<IElement> Elements;          // Chemical elements
  public Map<String, IMolecule> Species;     // Molecule species
  public Vector<KineticReaction> Reactions;

  public ChemkinFile(String fromFile) throws Exception
}
```

The ChemkinParser uses the Java Scanner to parse the CHEMKIN file. Furthermore the single parts of the file are converted to appropriate objects as outlined in the listing above. Molecule species are written as chemical formulae in the file and therefore have to be converted to SMILES strings before being able to create CDK IMolecule objects. The conversion to SMILES strings is again done using the CACTUS web service from the National Cancer Institute NCI.

```
public class ChemkinParser
{
  public ChemkinFile Parse(String filename) throws Exception

  private IMolecule getMoleculeFromSmiles(String smiles);
  private String getSmilesFromChemFormula(String chemFormula);
}
```

With the test application ChemkinTestApp the usage of the ChemkinLib is demonstrated by parsing a CHEMKIN input file and writing one file containing all molecule species in terms of SMILES strings and one file containing the string representations of all reactions as output.

5.3.2. Chemkin GML Generator

The Chemkin GML Generator is a Perl script that takes an input file that was created with the previously mentioned ChemkinTestApp and generates GML graph rewrite rules from the reactions. Because the input file was originally generated from some CHEMKIN file this script is also named after CHEMKIN. The listing below show an example line in such an input file:

```
[H] + [H]C([H])=C=O <-> [H][C]([H])[H] + C#O :: 1.511104421141E-11 J/mole
```

After parsing the separate molecule SMILES strings and parsing them to molecule objects, a reaction object is created, for which the atom-atom mapping is calculated.

```
my $reaction = Chemistry::ReactionGrammar::Reaction->new(
  educts    => \@educts,
  products => \@products
);

my $mapper = new Chemistry::ReactionGrammar::Mapper::CSL();
$reaction = $mapper->map($reaction);
```

After the atom mapping information is stored at the reaction object, the rewrite rule can be derived from the latter. The rewrite rule is than written in GML format to the output file.

```
my $gml = RewriteRule->new_from_reaction($reaction);
# expand rule context to contain all atoms of the reaction
$gml->expand_rules;
```

5.3.3. PostAnalyze

PostAnalyze is a Java tool for the graphical analysis of results of a StochGraphLab run. It parses an output file of the StochGraphLab application and visualizes the simulated reaction networks as substrate graphs. Molecule species can be depicted as SMILES strings or molecule graphs. Besides some different graph/tree layouts the simulated concentration of a species and the reaction rate constants of the networks reactions can be displayed additionally.

PostAnalyze is a Java SWING application and uses the CDK library [53] for parsing SMILES strings, the JChemPaint library [56] for generating molecule graphs, the JUNG library [55] for visualizing graphs/trees and the FreeHEP VectorGraphics library [57] for exporting those graphs in various formats. A detailed description of the usage of PostAnalyze and the handling in the user interface is given in appendix B.

Because a StochGraphLab run produces possibly many iterations during a reaction network generation, PostAnalyze reads every iteration separately and therefore holds a separate graph for each iteration. In the graphical user interface of PostAnalyze each iteration of the network simulation can be displayed extra.

```
// create a new JUNG graph object
// vertices and edges are of type String
Graph<String, String> g =
  new DirectedSparseMultigraph<String, String>();
...
g.addVertex(reactant);
g.addVertex(product);
g.addEdge(reactionIdentifier, reactant, product);
```

A JUNG graph is eventually displayed by creating a Layout<Vertex, Edge> object with the particular graph and further creating a VisualizationViewer<Vertex, Edge> object with the layout object. Layout<Vertex, Edge> only defines an interface and therefore has to be instantiated using some specific layout class such as CircleLayout or RadialTreeLayout.

```
Layout<String, String> layout =
  new CircleLayout<String, String>(g);
VisualizationViewer vv = new VisualizationViewer(layout);
```

The purpose of the VisualizationViewer<Vertex, Edge> object is the actual delineation of the graph and furthermore provides functionalities for fine adjustments, labeling the graph, scaling and user interactions.

Beside the SMILES string representation of the molecule species in the network graph, PostAnalyze is able to depict species as molecule graph. This molecule graph is created using the JChemPaint library [56].

```
private static Icon getMoleculeIconFromSmiles(String smi)
{
  IMolecule mol = null;
  SmilesParser sp = new SmilesParser(...);
  mol = sp.parseSmiles(smi);
  Image image = new BufferedImage(...);
  Graphics2D g2 = (Graphics2D)image.getGraphics();
  StructureDiagramGenerator sdg = new StructureDiagramGenerator();
  Renderer renderer = new Renderer(...);   ...
  IMoleculeSet molset = new MoleculeSet().addMolecule(mol);   ...
  renderer.paintMoleculeSet(molset, new AWTDrawVisitor(g2));
  return new LayeredIcon(image);
}
```

6. Results

6.1. Stochastic simulation

Before running stochastic simulations of whole reaction networks within our application StochGraphLab, we wanted to verify the stochastic simulator of the StochKit library [24]. Therefore we utilized the Discrete Stochastic Models Test Suit DSMTS [19]. Because of the probabilistic nature of stochastic approaches, testing them is so much the harder, since two identical simulators will never produce exactly the same set of results and exact solutions can only be calculated for very small reaction sets. Thus the only sensible method is to run the simulation lots of times and check the correctness of the distribution of the outcomes. The DSMTS contains several documented and well-defined test models. Each test model consists of a description in the SBML format, CSV files for mean values and standard deviations of the simulation results and plots of the latter against time. The following two figures describe the positive verification by means of a *Dimerisation* model and an *Immigration-Death* model.

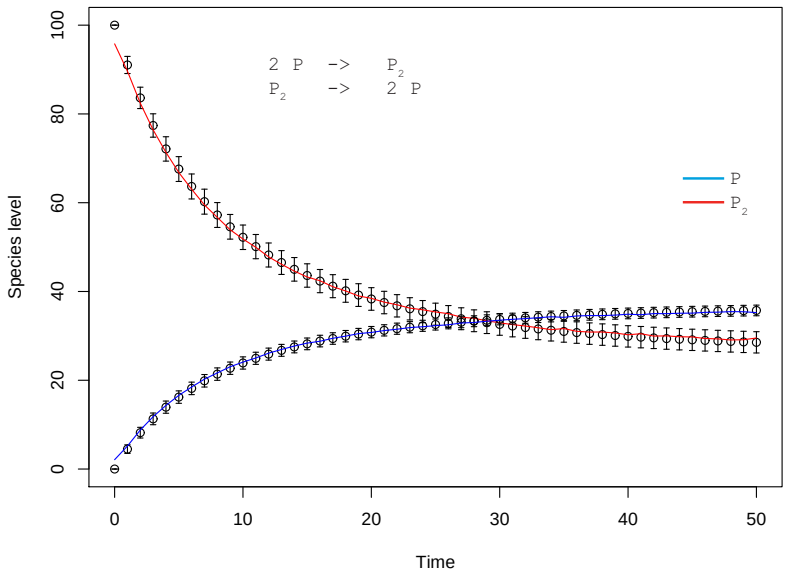


Figure 41: This dimerisation process is denoted by species P and P_2. The dimerisation reaction $2P \rightarrow P_2$ has rate parameter 0.001. The dissociation reaction $P_2 \rightarrow 2P$ has rate parameter 0.01. The colored lines show the mean values of 1000 runs and the black error bars show the mean values and standard deviations of the DSMTS reference.

Both verification attempts resulted in simulated mean values that are within the reference distribution proposed by the DSMTS. The particular test models were chosen to cover both, some quite difficult process involving two species (the dimerisation process) and a not less interesting process because of the statistical spread of the simulation results (the immigration-death process).

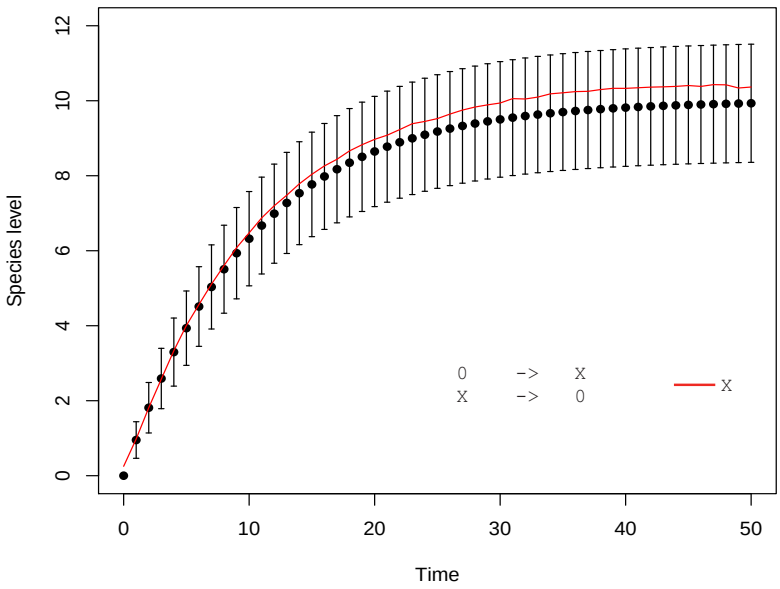


Figure 42: This immigration-death process is denoted by species X. The immigration reaction $\emptyset \rightarrow X$ has rate parameter 1. The death reaction $X \rightarrow \emptyset$ has rate parameter 0.1. The red line shows the mean values of 10000 runs and the black error bars show the mean values and standard deviations of the DSMTS reference.

6.2. Networks

In this chapter we describe the chemical reaction networks we simulated with our StochGraphLab application. These networks were used to test and successfully demonstrate our approach for a reaction network generator.

For all examined reaction networks the intra-molecular application of graph rewrite rules was allowed for reaction generation. Furthermore the verbosity level of StochGraphLab was set to at least three, in order to produce usable output for the PostAnalyze tool. Moreover the GML rewrite rules for each reaction network are listed in appendix C.

6.2.1. Diels-Alder

The Diels-Alder reaction is an organic cycloaddition between a conjugate diene and a substituted alkene and is named after Otto P.H. Diels and Kurt Alder [15]. As the cyclic product of this reaction also contains a double bond, it can server further as *alkene like* reactant. This repetitive character of the Diels-Alder reaction is the crucial factor for building up a reaction network on its mechanism. The reaction mechanism requires very little energy to create a cyclohexene ring, which is very useful in many other organic reactions. The diene component in the reaction can be open-chain or cyclic with the only limitation that it must be able to exist in the *cis*-conformation. This is against the normal tendency of two double bonds to arrange themselves as far away from each other as possible.

Below we describe simulation results of a reaction network starting with the two reactants CC=C and C=C(C)C=C. The simulation command for the first depiction in figure 43 is

```
StochGraphLab -rules=da_rule.gml -smiles=da1.smi -verbose=3
-pruneMaxMolSize=400 -autoScale -pruneConcThreshold=0.0001
```

whereas parameter `pruneMaxMolSize` limits the molecular mass of involved molecule species to 400 amu and `pruneConcThreshold` limits the concentration of species in the network to at least 0.0001.

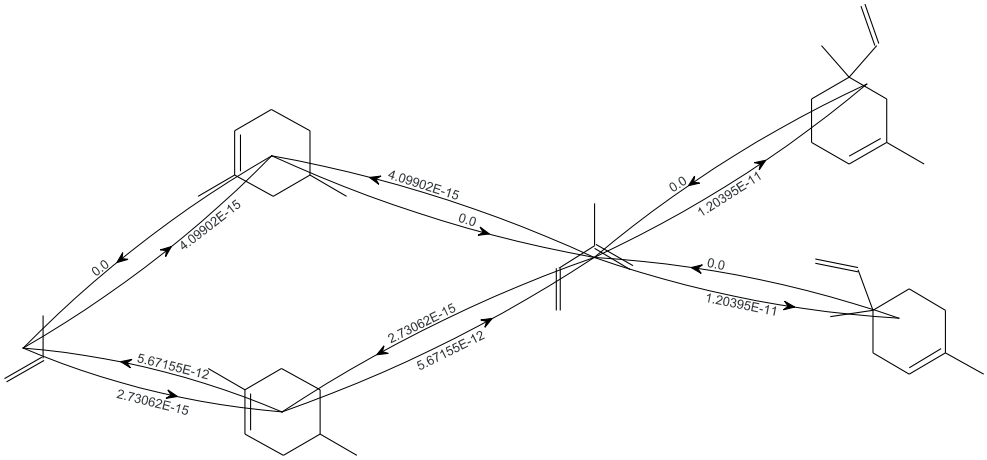


Figure 43: Simulation of a Diels-Alder reaction network. Starting with two molecule species this network soon stabilizes at six species because of the relative high concentration threshold.

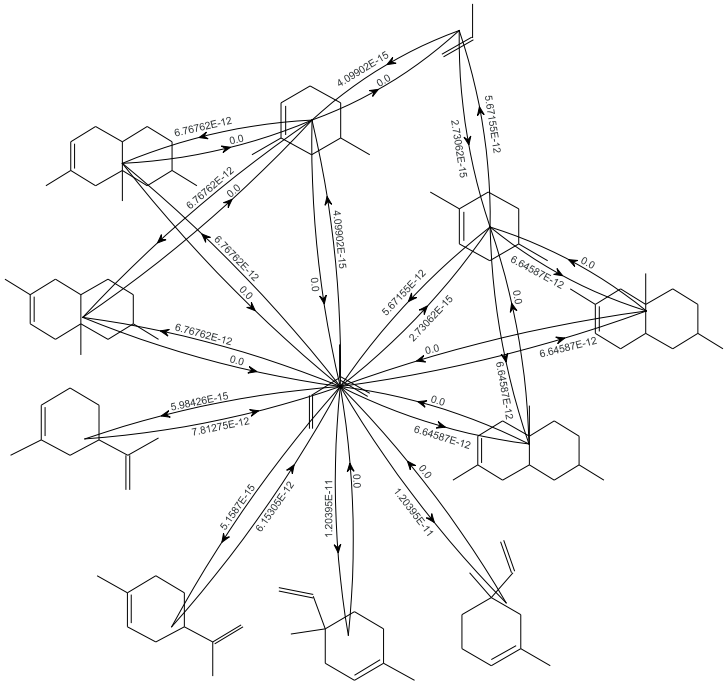


Figure 44: Iteration two of a simulation run with higher concentration threshold.

Both figures 44 and 45 are different depictions of the same iteration and the same simulation run. Only the representation of the molecules changes between a molecular graph and a colored circle proportionally indicating the simulated species concentration. The command can be given as

```
StochGraphLab -rules=da_rule.gml -smiles=da1.smi -verbose=3
-pruneMaxMolSize=400 -autoScale -pruneConcThreshold=0.000001
```

which simply sets a higher concentration threshold for pruning the reaction network. The obvious effect is, that the number of species in the stabilized network doubles to twelve. By setting parameter `autoScale` the species concentrations are scaled after each simulation of the network dynamics such that the highest concentration is 1.0. All figures of the Diels-Alder reaction network also outline the particular reaction rate constant for each reaction.

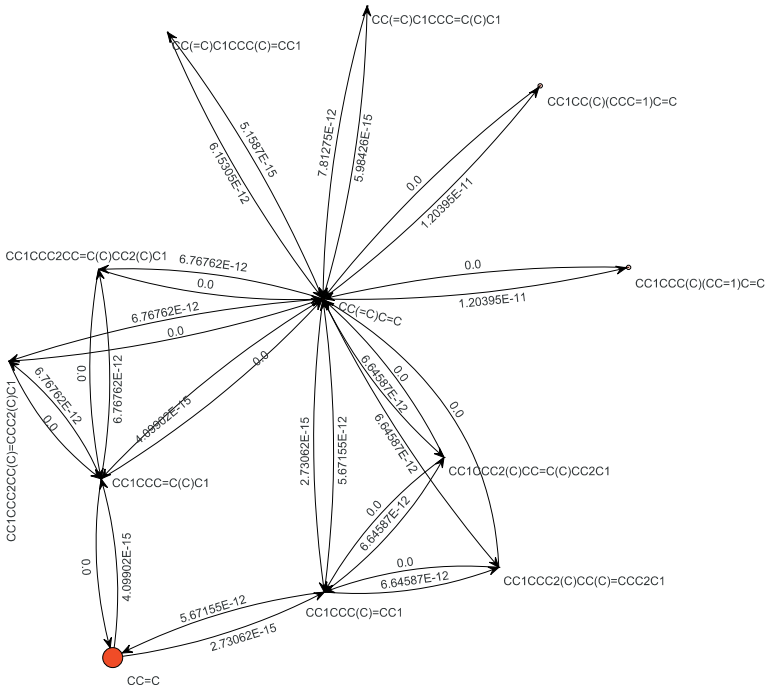


Figure 45: Depicting molecule species as colored circles. Fill color opacity and circle radius are linear proportional to the species concentration.

6.2.2. Formose reaction

Formose is an alloy of various sugar molecules that arises through self-condensation of formaldehyde on basic catalysts [46]. Due to the capability of the formose reaction to establish an auto-catalytic cycle, it is supposed to have played an important role in the emergence of the first biomolecules [3] and might therefore be a candidate for the origin of life scenario. The formose reaction also explains parts of the path from simple

formaldehyde to complex sugars like ribose and from there to RNA. Because of this potential it is also a perfect candidate to be tested and simulated with our application StochGraphLab.

To simulate a reaction network on base of the formose reaction we first picked the separate reaction mechanisms (listed in appendix C), contributing to the complete formose reaction, from literature and derived graph rewrite rules from them. Some reactions like the *Keto Enol tautomerism* are rather unspecific and do not require special constraints in the graph transformation or an almost completely expanded rule context. However, there are also reactions like the *Cannizzaro disproportionation*, which occur only under specific circumstances and therefore need quite specific rewrite constraints and an accordingly specific rule context.

In the following we describe simulation results of a reaction network starting with the two reactants formaldehyde [H]C(=O)[H] and water [H]O[H]. The simulation command for the first depiction is

```
StochGraphLab -rules=formose.reaction.allHs.in.gml
-smiles=formose.educts.smi -verbose=3 -pruneMaxMolSize=200 -noSimulation
```

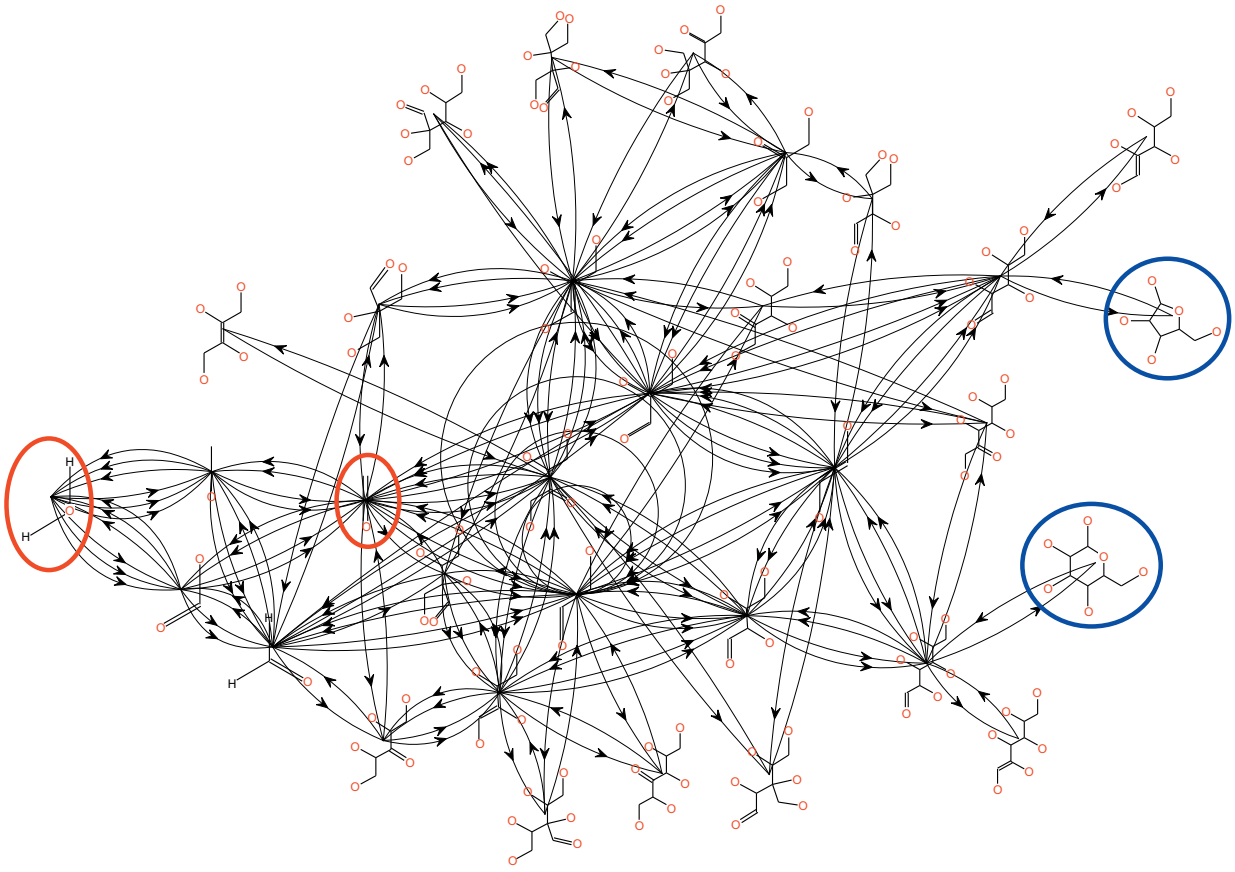


Figure 46: Iteration 5 of the simulated formose reaction network. The start molecules are emphasized by red circles. The blue circles mark two sugars (hexose, pentose) that

were generated during the simulation, which shows, that the mechanism of the formose reaction is implemented and covered correctly, since sugars represent an observed and predicted outcome of the reaction. There are no reaction rates or species concentrations because of parameter `noSimulation`.

The next figure shows reaction rate constants, achieved by the simulation command

```
StochGraphLab -rules=formose.reaction.allHs.in.gml
-smiles=formose.educts.smi -verbose=3 -pruneMaxMolSize=200 -autoScale
```

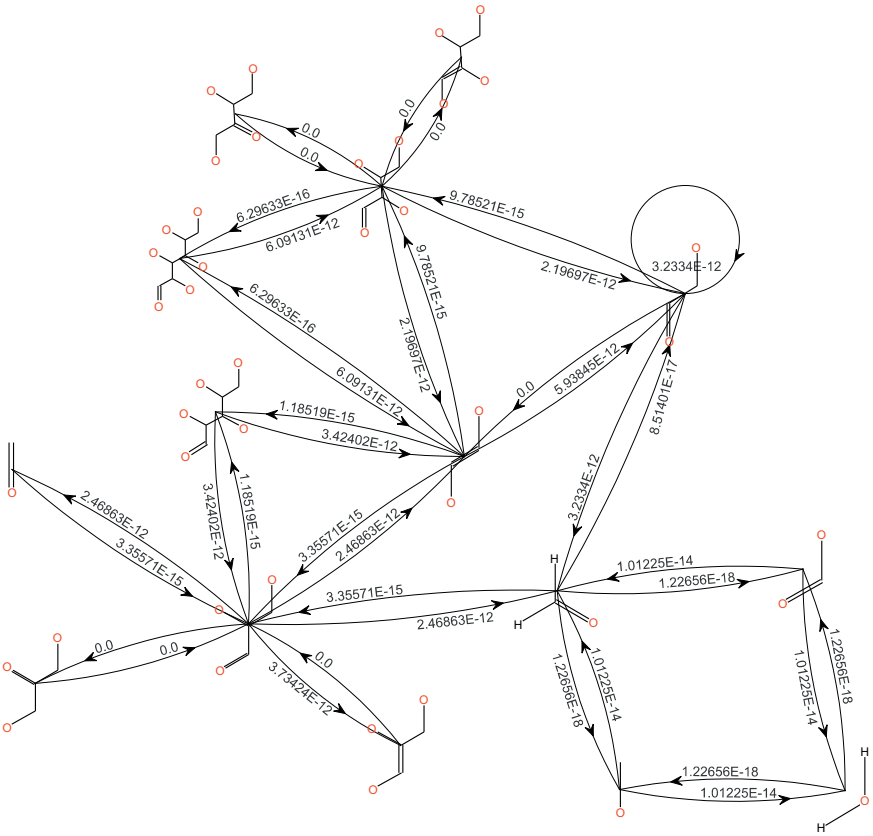


Figure 47: Simulation of the network kinetics of the formose reaction network.

Besides the kinetic simulation we also analyzed reaction cycles in the formose reaction network. With the following command cycles of path length 5 to 6 were searched in the directed network substrate graph:

```
StochGraphLab -rules=formose.reaction.allHs.in.gml
-smiles=formose.educts.smi -verbose=3 -pruneMaxMolSize=200 -noSimulation
-anaCycles=1 -minCycleSize=5
```

In iteration 5 of the network generation procedure a few cycles of length 5 to 6 were found. The following figure shows one of those, i.e. the famous auto-catalytic cycle of the formose reaction found in nature and reproduced in experiments. This again shows the capability of simulating reasonable reaction networks within our application.

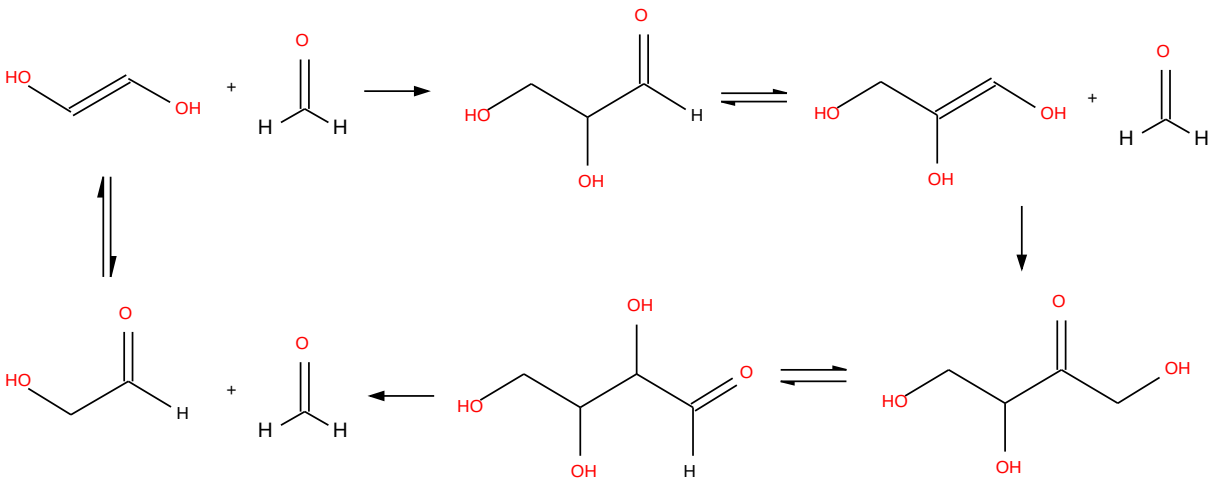


Figure 48: A cyclic reaction path found in the simulated Formose reaction network.

All simulations of the Formose reaction were parameterized for pruning molecule species with more than 200 atomic mass units amu, because larger molecules are not supposed to play a role in scenarios concerning the origin of life.

6.3. Restrictions & Comments

In addition to the results we want to list some restrictions and drawbacks of the current implementation of StochGraphLab and its environment:

- ⋆ Chemical radicals are generally not supported by the underlying libraries and can therefore not be used in reaction networks. As a result, for instance, combustion reactions and atmospheric high energy reactions can not yet be processed by our application.
- ⋆ The imaginary transition state ITS, calculated by the graph grammar library GGL, in some cases can not be used for the reaction rate calculation because the hydrogen count in the ITS is not necessarily equal to the number of hydrogens in the reactants or products. Generally the transition state is a rather blurred state on the chemical transition from educts to products, which means there might be bonds that are already broken or bonds that are even established although in the product molecules the bonds are again differently formed and positioned. However, the quantum mechanics program `jaguar` (appendix D) can only handle equal atom numbers on educts site and products site as well as in the transition state and its calculations will not converge otherwise. To circumvent this difficulty we calculate an own transition state within the reaction rate calculation or let `jaguar` estimate it, which takes longer in any case.
- ⋆ Using StochKit as stochastic simulation toolkit we soon realized a missing feature. Normally one has to exactly tell StochKit the time the simulation should

end. However, our use cases are to create possibly new reaction networks and for unknown scenarios it is not realistic to estimate or guess the time when further simulation can be stopped. It is just the equilibrium in the molecule network we are interested in when asking for the molecules concentration distribution. As it is not feasible to setup an arbitrarily high end time for every simulation just to make sure every imaginable reaction network reaches its steady state we contacted the developers group of StochKit. Thereby we learned about a new version StochKit2 of which the release date was too far away. Thus we implemented an own heuristic within StochKit to guess the quasi equilibrium of the simulated reaction network. In the meanwhile a beta version of StochKit2 was released in March 2010 but it still misses the desired feature so a switch-over to the new version was not necessary.

⋆ Molecule species in the reaction network might be dissipated to a large extent for producing other molecule species and therefore reach only low concentrations in the simulation of network dynamics. Furthermore this may lead to expelling those species from the network according to network pruning although they have doubtlessly been important and maybe even required for the production of other species. To resolve such wrong decisions in the algorithm, new and additional pruning strategies should be investigated and implemented.

⋆ The temperature at which a particular chemical reaction takes place is closely connected to the rate constant, as deduced by the Arrhenius equation and the Transition State Theory. Nevertheless our quantum mechanics calculation program `jaguar` is currently not able to optimize molecule structures for a given temperature but only for the default of 300 Kelvin. This implies that further also the Gibbs energy of the involved molecules is always determined regardless of the temperature.

7. Conclusion

7.1. Summary

As mentioned in the results chapter, the presented approach for generating chemical reaction networks using graph transformations and stochastic simulations is able to simulate chemically reasonable results.

At the beginning the Diels-Alder reaction network was used to test the mechanisms for expanding the network, simulating the dynamic and the various pruning strategies, since the part of the graph transformation is well-understood for the Diels-Alder reaction. When diverse errors were fixed and the stochastic simulation was enhanced to detect equilibrium states, we proceeded to derive graph rewrite rules for further reaction networks. As we are especially interested in prebiotic chemistries, we have chosen the Formose reaction, which resulted in successfully reproducing important outcomes such as hexose and pentose sugars and the formation of reaction cycles within the reaction network.

Nevertheless, there are enough investigations we did not manage to carry out. Therefore we dedicate the next chapter to ideas and suggestions for future work.

7.2. Outlook

The following list contains possible improvements and ideas we want to suggest for further work on this topic:

- ⋆ Because of the inherent probabilistic nature of stochastic simulators, their results are the better the more often the simulation is run. To speed up a simulation the separate runs should therefore be parallelized. A computer component capable of running many computations in parallel is the graphics processing unit GPU. The stochastic simulation library StochKit we are using has already been tested on a graphics board [23]. Still the developed code for the tests on the GPU is not available as a library.

- ⋆ At the moment the StochKit library is not able to detect steady-state conditions in the simulated reaction network, which is why we implemented our own heuristic. Anyhow a well-founded strategy for stopping the stochastic simulation on reaching equilibrium would be appreciated.

- ⋆ A complete generation and simulation of a reaction network consists of one *fixed* set of rewrite rules, passed to the StochGraphLab application. To simulate even more realistic and natural conditions concerning evolutionary aspects, the reaction network should be scanned for new rewrite rules, applicable to the contained molecule species, after each iteration to extend the set of known rules. This could be done for instance by considering and examining overlapping molecule orbitals when simulating the collision of two or more species, i.e. testing a possible reaction.

- ⋆ Present pruning strategies consider only the species concentration, which might be a problem as mentioned in section 6.3. Therefore new pruning strategies could be implemented, e.g. to consider reaction rates leading to higher or lower expressed molecule species.

- ⋆ Considering the current capability of analyzing cycles in reaction paths a new mode for network simulation could be developed. In this mode reaction rates are only calculated for reactions participating in a cycle. Furthermore only the dynamic of these cycles would have to be simulated.

- ⋆ Cycle analysis in the reaction network currently uses a brute force algorithm. Of course a more sophisticated subgraph isomorphism algorithm would make sense.

- ⋆ Currently the reaction rate calculation server contains both, the TCP/IP server and the actual rate calculation. For more general usage a server farm for parallel rate calculation and one TCP server for communicating with client applications and further dispatching the calculation jobs to the server farm.

- ⋆ Although the reaction rate calculation is rather robust in terms of receiving SMILES strings of various *dialects*, fetching the structural data from the CACTUS web service and deriving the transition state of a reaction, there are still reactions resulting in a rate of 0.0. In most cases this is due to an error in `jaguar` when optimizing the molecule geometry.

- ⋆ Furthermore the reaction rate calculation should be verified and calibrated utilizing measured reaction rates of physicochemical experiments.

- ⋆ As mentioned in the introduction of this work Eschenmoser proposed some reaction mechanisms and networks that are supposed to form auto-catalytic reaction paths and to potentially prebiotic character [3]. For detailed modeling of the proposed reactions an own diploma thesis could be announced for some organic chemist with sophisticated insights in prebiotic chemistries.

A. StochGraphLab usage

The usage of our main application StochGraphLab is as follows:

```
StochGraphLab -rules=...  -smiles=...  [...optional parameters]
```

Parameters `rules` and `smiles` are required and define the rewrite rule set and the set of molecule species respectively. Both parameters take a colon ':' separated list of file names. In the following the optional parameters are listed and described briefly:

`pruneConcThreshold` Removes species with smaller concentration than threshold from the network.
type=double

`pruneMaxNetworkSize` Maximum network size. Species with smallest concentration are removed from the network.
type=double

`pruneMaxMolSize` Removes species with higher molecule mass than the maximum from the network (in atomic mass unit amu).
type=double

`host` TCP host of reaction rate calculation server.
type=string

`port` TCP port of reaction rate calculation server.
type=integer

`verbose` Set verbosity level for outputs: 0 = off, 1 = output every iteration, 2 = output simulation input network, 3 = output reactions, 4 = output Reaction rate strings, 5 = output simulation input stoichiometry matrix, 6 = output propensity functions.
type=integer

`out` Output file name or STDOUT, if not given.
type=string

`rates` A ':' separated list of file names containing reaction rates per GML rule ID.
type=string

`simiter` Number of stochastic simulations to run in each iteration of the network generation. 1 if not given.
type=string

`temp` Surrounding temperature of the reaction network or 298.15 Kelvin if not given.
type=double

`iter` Number of network generation iterations.
type=integer

`simdetailout` Output file name to write history of stochastic simulations, i.e. the separate trajectories.
type=string

`initconc` A ':' separated list of file names containing initial concentrations for specified molecule species.
type=string

`noRuleCheck` Do not check the consistency of the rewrite rules.
type=boolean

`autoScale` Apply automatic scaling on every species' concentration (highest concentration will be 1.0).
type=boolean

`noSimulation` Do not actually simulate the reaction network.
type=boolean

`anaCycles` Analyze reaction network for cycles: 1 = use directed graphs, 2 = use undirected graph.
type=integer

`minCycleSize` Minimum size of cycles to find or 3, if not given.
type=integer

`maxCycleSize` Maximum size of cycles to find or 6, if not given.
type=integer

Reaction Rate Calculation Server

To start a reaction rate calculation server on a machine, simply the Perl script `calc_rate.pl` has to be started. If all output and error messages should be written to some log file and the process should be started in the background, the following command is appropriate:

```
perl calc_rate.pl > logfile.log 2>&1 &
```

B. Tools usage

ChemkinLib test application

The usage of the test application for the CHEMKIN II library is

```
ChemkinTestApp.jar input_file smiles_educts reactions_output
```

whereas the input file is a network defined in the CHEMKIN II format and `smiles_educts` and `reactions_output` are the output files, extracted from the network definition.

Chemkin GML Generator

As mentioned in chapter 5.3.2 the GML generator takes a reactions input file, generated with the ChemkinTestApp, and creates GML graph rewrite rules from the given reactions. The usage of the Perl script is

```
perl chemkin_rulegen.pl inputfile
```

which creates the two output files `inputfile.gml` containing the GML rules and `inputfile.rates.txt` containing the reaction rate constants for every rule parsed from the CHEMKIN input file.

PostAnalyze

The usage of the PostAnalyze tool is as follows:

```
PostAnalyze input_file [Smiles|Graphs] [Circle|FR|Tree] [molecule size]
```

The input file is some output file of the StochGraphLab with a verbosity level of at least 3. Second parameter is the representation of the molecule species. Third parameter is the network graph layout, where FR means Fruchterman-Reingold. The forth parameter is the size of the molecular graphs in pixel - only considered if molecules are represented as graphs.

Once the graphical user interface of PostAnalyze has started, following user interactions are possible:

- Menu *Graphs*: Select a particular iteration of the network simulation to display
- Menu *Tools*: Switch on/off labels, Zooming, Export to graphics file
- *Ctrl+A*: select all vertices
- *Mouse click*: pick or move vertices, press Shift for multi select, press Ctrl to fix the view
- *Mouse wheel and +/- keys* for scrolling

C. Rewrite rules

Diels-Alder

```
rule [
 ruleID "Diels Alder reaction"
 context [
   node [ id 0 label "C" ]
   node [ id 1 label "C" ]
   node [ id 2 label "C" ]
   node [ id 3 label "C" ]
   node [ id 4 label "C" ]
   node [ id 5 label "C" ]
 ]
 left [
   edge [ source 1 target 2 label "=" ]
   edge [ source 4 target 5 label "-" ]
   edge [ source 3 target 4 label "=" ]
   edge [ source 0 target 5 label "=" ]
 ]
 right [
   edge [ source 0 target 1 label "-" ]
   edge [ source 1 target 2 label "-" ]
   edge [ source 2 target 3 label "-" ]
   edge [ source 3 target 4 label "-" ]
   edge [ source 4 target 5 label "=" ]
   edge [ source 5 target 0 label "-" ]
 ]
]
```

Formose reaction

If we printed all fourteen GML rewrite rules of the formose rule set, it would fill a couple of pages. Thus we refer the committed reader to the provided source code, in which the rule set for the formose reaction network can be found in the file `/test/networks/formose/formose.reaction.allHs.in.gml`.

Nevertheless we want to list the reaction mechanisms, we derived the rule set from, in order to simulate the formose reaction network.

- *Cannizzaro disproportionation*
- *Keto Enol tautomerism* and reverse mechanism
- *Aldol addition* and reverse mechanism
- *Aldose Ketose isomerization* and reverse mechanism
- *Pentose cyclization*
- *Hexose cyclization*

D. Jaguar 7.5

Here we describe the various input files used for the calculations with `jaguar` 7.5 [26].

Transition state calculation

To determine or optimize the transition state of a reaction the listing below shows an example input file. Parameter `basis` is always set to basis set 6-31G* in our calculations. Parameter `molchg` is the molecules charge, `igeopt` has to be 2 for a transition state search, `iqst` denotes the search method and is set to 1, because we can provide the structural data for reactants, products and even for a guess of a transition state. The maximum number of optimization iterations is defined by `maxitg`. If we do *have* a transition state[1], we can write all three zmat-sections, whereas `zmat` contains the transition state, `zmat2` the reactants and `zmat3` the products. If we do not know the transition state reactants and products are written to sections `zmat` and `zmat2` respectively.

```
&gen
 basis=6-31g*
 molchg=<integer>
 igeopt=2
 iqst=1
 maxitg=10
&
&zmat
H1   3.0739  0.1550  0.0000
H2   2.0000  0.1550  0.0000
O3   2.5369  -0.1550  0.0000
&
&zmat2
 ...
&
&zmat3
 ...
&
```

Geometry optimization

To optimize the geometry of particular molecules the options in the gen-section have to be adapted as listed below. Parameter `igeopt` is set to 1 for only optimizing the structural data of one set of molecules. The latter is declared within the `zmat` section.

```
&gen
 basis=6-31g*
 molchg=<integer>
 igeopt=1
 maxitg=10
&
&zmat
 ...
&
```

[1] To *have* a transition state here always means to have at least a guess for it.

Energy Calculation

For the calculation of the Gibbs free energy of some molecules the input file has to declare following parameters:

`igeopt` is set to 0, since the geometry has been optimized in previous jobs

`ifreq` set to 1 activates the calculation of vibrational frequencies, needed for the energy calculation

`scalfr` is a scale factor for the frequency calculations (value 0.8953 for basis set 6-31G* was taken from [26])

`isolv` is 0 so `jaguar` performs the calculations in the gas phase

`eunit` = 2 sets the energy unit to kJ/mol

The structural data is again given in the single `zmat` section.

```
&gen
basis=6-31g*
molchg=<integer>
igeopt=0
ifreq=1
scalfr=0.8953
isolv=0
eunit=2
&
&zmat
...
&
```

E. Access to our software

All software developed and implemented within this thesis is available under the lesser GNU LGPL software license and can be accessed through the following git repository stored at the TBI web site of the University of Vienna:
`http://www.tbi.univie.ac.at/~dhoegerl/Diplomarbeit/Software/repository`

You can use the following command to check out from our repository:
`git clone ssh://user@repository`

It is also available as tarball:
`http://www.tbi.univie.ac.at/~dhoegerl/StochGraphLab.tar.gz`

List of Figures

References

[1] S. L. Miller, and H. C. Urey *Organic Compound Synthesis on the Primitive Earth, Science, 130, 245 (1959)*

[2] G. Wächtershäuser *Groundworks for an evolutionary biochemistry: The iron-sulphur world, Progress in Biophysics and Molecular Biology 58 (2): 85ff, doi:10.1016/0079-6107(92)90022-X (1992)*

[3] A. Eschenmoser *On a Hypothetical Generational Relationship between HCN and Constituents of the Reductive Citric Acid Cycle, CHEMISTRY & BIODIVERSITY Vol. 4: 554-573, 2007*

[4] J. L. Faulon, and A. G. Sault *Stochastic Generator of Chemical Structure. 3. Reaction Network Generation, J. Chem. Inf. Comput. Sci. 41:894-908, 2001*

[5] DT. Gillespie *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, J. Comput. Phys. 22:403-34, 1976*

[6] G. Benkö, C. Flamm and P. F. Stadler *A graph-based toy model of chemistry, J. Chem. Inf. Comput. Sci. 43:1085-1093, 2003*

[7] R. Heckel *Graph Transformation in a Nutshell, Electronic Notes in Theor. Comput. Sci. 148:187-198, 2006*

[8] C. Flamm and P. F. Stadler *Topology of Chemical Reaction Networks, 2006*

[9] C. Flamm, A. Ullrich, H. Ekker, M. Mann, D. Högerl, M. Rohrschneider, S. Sauer, G. Scheuermann, K. Klemm, I. L. Hofacker and P. F. Stadler *Evolution of Metabolic Networks - A Computational Framework, 2010*

[10] M. Mann, C. Flamm *Graph Grammar Library GGL, 2010*
`http://www.tbi.univie.ac.at/TBI/software.html`

[11] M. Raitner *GML file format, 1996*
`http://www.infosun.fim.uni-passau.de/Graphlet/GML/`

[12] J. Gunawardena *Chemical reaction network theory for in-silico biologists, 2003*

[13] M. Frenklach *Modeling of Large Reaction Systems, Springer Series in Chemical Physics; Springer-Verlag: Vol. 47:pp 2-16, 1987*

[14] W. Fontana *Algorithmic Chemistry. In Artificial Life II. 159-210, 1992*

[15] O. Diels and K. Alder *Synthesen in der hydroaromatischen Reihe, Liebigs Ann. Chem. 460:98-122, 1928*

[16] L. P. Cordella, P. Foggia, C. Sansone, M. Vento *An improved algorithm for matching large graphs. In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, 149-159, 2001*

[17] J. R. Ullmann *An Algorithm for Subgraph Isomorphism. Journal of the Association for Computing Machinery, 23:31-42, April 1976*

[18] DT. Gillespie *Stochastic Simulation of Chemical Kinetics, Annu. Rev. Phys. Chem. 58:35-55, 2007*

[19] T. W. Evans, C. S. Gillespie and D. J. Wilkinson *The SBML discrete stochastic models test suite. Bioinformatics, 24:285-286, 2008*

[20] Y. Cao, DT. Gillespie and L. R. Petzold *Avoiding negative populations in explicit Poisson tau-leaping. J. Chem. Phys. 123 (5), Article 054104, 2005*

[21] DT. Gillespie *Approximate accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. 115:1716-1733, 2001*

[22] DT. Gillespie *Deterministic Limit of Stochastic Chemical Kinetics. J. Phys. Chem. B. 113:1640-1644, 2009*

[23] L. Hong and L. R. Petzold *Efficient Parallelization of Stochastic Simulation Algorithm for Chemically Reacting Systems on the Graphics Processing Unit. Int. J. of High Perf. Comput., Vol. 24, No. 2, 107-116, 2010*

[24] Y. Cao, A. Hall, H. Li, S. Lampoudi and L. Petzold *StochKit, 2010* `http://engineering.ucsb.edu/~cse/StochKit/`

[25] OpenBabel community *Open Babel: The Open Source Chemistry Toolbox* `http://openbabel.org/wiki/Main_Page`

[26] *Jaguar, version 7.5, Schrodinger, LLC, New York, NY, 2008*

[27] W. H. Miller *Beyond Transition State Theory: Rigorous Quantum Approaches for Determining Chemical Reaction Rates, 1995*

[28] S. Fujita *Description of Organic Reactions Based on Imaginary Transition Structure. J. Chem. Inf. Comput. Sci. 26:205-212, 1986*

[29] M. Feinberg *The Chemical Reaction Network Toolbox,* `http://www.che.eng.ohio-state.edu/~feinberg/crnt/`

[30] N. Soranzo and C. Altafini *ERNEST: a toolbox for chemical reaction network theory. Bioinformatics, 25(21):2853-2854, 2009*

[31] F. Horn and R. Jackson *General mass action kinetics. Arch. Rational Mech. Anal. 47:81-116, 1972*

[32] M. Feinberg and F. Horn *Chemical mechanism structure and the coincidence of the stoichiometric and kinetic subspace. Arch. Rational Mech. Anal. 66:83-97, 1977*

[33] M. Feinberg *Chemical Reaction Network Structure and the stability of complex isothermal reactors– I. The deficiency zero and deficiency one theorems. Chem. Eng. Sci. 42:2229-2268, 1987*

[34] Wikipedia contributors *Cheminformatics. Wikipedia, The Free Encyclopedia, 1 April 2010*
`http://en.wikipedia.org/w/index.php?title=Cheminformatics&oldid=353379210`

[35] Wikipedia contributors *Chemical database. Wikipedia, The Free Encyclopedia, 27 March 2010*
`http://en.wikipedia.org/w/index.php?title=Chemical_database&oldid=352312478`

[36] D. Weininger *SMILES, a chemical language and information-system. 1. Introduction to methodology and encoding rules. J. Chem. Inf. Comput. Sci. 28:31-36, 1998*

[37] D. Weininger, A. Weininger, J. L. Weininger *SMILES 2: Algorithm for generation of unique SMILES notation. J. Chem. Inf. Comput. Sci. 29:97-101, 1989*

[38] Daylight Chemical Information Systems, Inc. *Daylight Theory Manual, Daylight Version 4.9, Release Date 02/01/08*
`http://www.daylight.com/dayhtml/doc/theory/`

[39] M. K. Yadav, B. P. Kelley and S. M. Silverman *The Potential of a Chemical Graph Transformation System. ICGT 2004, LNCS 3256, pp. 83-95, 2004*

[40] F. Rossello and G. Valiente *Graph Transformation in Molecular Biology. LNCS 3393, pp. 116-133, 2005*

[41] F. Rossello and G. Valiente *Chemical Graphs, Chemical Reaction Graphs, and Chemical Graph Transformation. Elec. Notes in Th. Comput. Sc. 127, 157-166, 2005*

[42] Wikipedia contributors *Hypergraph. Wikipedia, The Free Encyclopedia, 16 April 2010*
`http://en.wikipedia.org/w/index.php?title=Hypergraph&oldid=356359271`

[43] Campbell McInnes *Virtual screening strategies in drug discovery. Current Opinion in Chemical Biology, Volume 11, Issue 5, Analytical Techniques / Mechanisms, pp. 494-502, ISSN 1367-5931, doi:10.1016/j.cbpa.2007.08.033 (2007)*

[44] Wikipedia contributors *Quantitative structure-activity relationship. Wikipedia, The Free Encyclopedia, 17 March 2010*
`http://en.wikipedia.org/w/index.php?title=Quantitative_structure-activity_relationship&oldid=350385714`

[45] Wikipedia contributors *Graph theory. Wikipedia, The Free Encyclopedia, 8 May 2010*
http://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=360869681

[46] Wikipedia contributors *Formose reaction. Wikipedia, The Free Encyclopedia, 10 April 2010*
http://en.wikipedia.org/w/index.php?title=Formose_reaction&oldid=355057347

[47] J. Newman *Leonhard Euler and the Königsberg Bridges. Sci. Amer. 189, 66-70, 1953*

[48] O. Ore *A Note on Hamiltonian Circuits. American Mathematical Monthly 67, 55, 1960*

[49] D. L. Applegate, R. E. Bixby, V. Chvátal and W. J. Cook *The Traveling Salesman Problem: A Computational Study, Princeton University Press, ISBN 978-0-691-12993-8, 2006*

[50] E. W. Weisstein *Isomorphic Graphs. From MathWorld–A Wolfram Web Resource.*
http://mathworld.wolfram.com/IsomorphicGraphs.html

[51] K. J. Laidler *A glossary of terms used in chemical kinetics, including reaction dynamics (IUPAC Recommendations 1996). Pure and Appl. Chem., Vol. 68, No. 1, pp. 190, 1996*

[52] I. Tubert-Brohman *PerlMol - Perl Modules for Molecular Chemistry.*
http://www.perlmol.org/

[53] CDK Community *Chemistry Development Kit CDK.*
http://cdk.sourceforge.net/

[54] Sandia National Laboratories *CHEMKIN.*
www.sandia.gov/chemkin/

[55] JUNG Framework Development Team *JUNG 2.0 – the Java Universal Network/ Graph Framework.*
http://jung.sourceforge.net/

[56] Steinbeck Group *JChemPaint JCP 3.0*
http://sourceforge.net/apps/mediawiki/cdk/index.php?title=JChemPaint

[57] FreeHEP Project *FreeHEP VectorGraphics.*
http://java.freehep.org/vectorgraphics/

Curriculum Vitae

Contact information

Name	Högerl Daniel
Date of Birth	October 25th, 1985
Gender	Male
Place of Birth	Eisenstadt, Austria
Citizenship	Austria
Address	Rechte Hauptzeile 9, A-7053 Hornstein
Country	Austria
Cell Phone	+43 650 47 01 960
Email	daniel@hoegerl.org

Education

2008 Cambridge Certificate Advanced English
2007−2010 University FH Campus Wien, Bioinformatics
2003 Cambridge First Certificate English
2000−2005 Institution of higher education for Telecommunications Mödling
1996−2000 Secondary school Wolfgarten Eisenstadt
1992−1996 Elementary school Hornstein

Professional experience

2009 Diploma thesis at the Institute for theoretical chemistry, University Vienna *Simulation of prebiotic chemistries*

Since March 2006 ETM Prof. Control GmbH, Software development

A-level thesis *Electronic Anemometer*

Skills

Software	MS Visual Studio, MSSQL, Oracle, Office applications, Eclipse, Cygwin, GCC
Programming languages	Delphi, C, C++, C++ CLI, C#, Java, Perl, JavaScript
Languages	German (expert), English (fluent), Croatian (intermediate)

Publications

C. Flamm, A. Ullrich, H. Ekker, M. Mann, D. Högerl, M. Rohrschneider, S. Sauer, G. Scheuermann, K. Klemm, I. L. Hofacker and P. F. Stadler *Evolution of Metabolic Networks - A Computational Framework, 2010*