universität wien

MASTERARBEIT

Titel der Masterarbeit

„*De-novo* enzyme design for olefin metathesis“

verfasst von

Michael Gastegger BSc

angestrebter akademischer Grad

Master of Science (MSc)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 066 862
Studienrichtung lt. Studienblatt:	Masterstudium Chemie
Betreut von:	Univ.-Prof. Dr. Leticia González Herrero

ABSTRACT

One of the most tantalizing challenges in computational chemistry and bioinformatics is the design of artificial enzymes, tailored specifically to catalyse any desired chemical transformation. The goal of this thesis was the *de-novo* computational design of a metalloenzyme capable of catalysing the ring closing metathesis of diallylether, a reaction with no natural counterpart. The "inside-out" enzyme design protocol was employed for this task, where a theoretically engineered novel active site is introduced into the scaffold of an existing protein. The active site was modelled after motifs present in the popular second generation Grubbs ruthenium-based metathesis catalyst. In order to efficiently search for active site configurations with high catalytic activity, a genetic algorithm was newly implemented, combining high throughput quantum-chemical computations on DFT-level with a stochastic optimisation procedure. The Rosetta suite of programs was then used to screen a database of protein crystal structures for scaffolds where these model active sites could be realized and to optimise catalytic interactions via a subsequent step of sequence design. This procedure resulted in four promising designed metalloenzymes, based on the protein scaffolds with the Protein Data Bank IDs 1JQ5, 3E3P, 1O8V and 3C9U. To assess the metathesis activity of these designs, they will be expressed and characterized experimentally by the group of Prof. Dr. C. Becker (University of Vienna).

ABSTRACT

Die Entwicklung von Enzymen mit neuartiger Funktionalität ist eines der vielversprechendsten Anwendungsgebiete von computergestützten bioinformatischen und quantenchemischen Methoden. Im Rahmen dieser Masterarbeit wurde ein Metalloenzym entwickelt, welches die Ringschlussmetathese von Diallylether katalysiert, eine in der Natur unbekannte Reaktion. Der Designvorgang wurde entsprechend dem "inside-out" Protokoll für *de-novo* Enzym-Design gestaltet, welches vorsieht, ein neuartiges reaktives Zentrum in das Grundgerüst eines bereits bekannten Proteins einzubringen. Die Modellierung eines Reaktionszentrums für Metathese erfolgte am Beispiel des hochaktiven, rutheniumbasierten Grubbs-Katalysators der zweiten Generation. Um eine effiziente Suche nach katalysefördernden Reaktionszentrumsmodellen zu gewährleisten, wurde ein stochastischer Optimierungsalgorithmus in der Form eines genetischen Algorithmus entwickelt, welcher quantenchemische Berechnungen auf DFT-Basis als Selektionskriterium verwendet. Anschließend wurde das ROSETTA-Programmpaket benutzt, um eine Datenbank von Proteinkristallstrukturen nach für die Unterbringung der Modellgeometrien geeigneten Proteingerüsten zu durchsuchen und um die katalytischen Wechselwirkungen dieser neu eingeführten aktiven Zentren mittels Sequenzdesign zu optimieren. Die Anwendung dieser Prozedur führte zum Neudesign vier vielversprechender Metalloenzyme auf Basis der Proteine mit den Proteindatenbank IDs 1JQ5, 3E3P, 1O8V und 3C9U. Um diese Designs auf ihre Metatheseaktivität hin zu untersuchen, wird ihre experimentelle Exprimierung und Charakterisierung in der Gruppe von Univ.-Prof. Dr. C. Becker (Universität Wien) erfolgen.

..

CONTENTS

FREQUENTLY USED ACRONYMS

ATP	Adenosine-triphosphate
CATH	Class architecture topology homologous superfamily
DFT	Density functional theory
GGA	General gradient approximation
HF	Hartree–Fock
LSDA	Local spin density approximation
MARI-J	Multipole accelerated resolution of identity
MD	Molecular dynamics
NAD+	Nicotinamide Adenine Dinucleotide
PDB	Protein data bank
PES	Potential energy surface
QM/MM	Quantum mechanics/molecular mechanics
RCM	Ring closing metathesis
RI	Resolution of identity
TD-DFT	Time-Dependant Density Functional Theory

LIST OF SYMBOLS

∇	Nabla operator
M	Nuclear mass
Z	Nuclear charge
N	Number of nuclei
n	Number of electrons
$\mathbf{R}$	Nuclear coordinates
R_A	Coordinate of nucleus A
$\mathbf{r}$	Electronic coordinates
r_i	Coordinate of electron i
R_{AB}	Internuclear distance
r_{Ai}	Nucleus-electron distance
r_{ij}	Interelectron distance
E_{total}	Total energy
E_{el}	Electronic energy
$\lvert\Psi\rangle$	Wavefunction
$\lvert\Psi_{\text{nu}}\rangle$	Nuclear wavefunction
$\lvert\Psi_{\text{el}}\rangle$	Electronic wavefunction
$\hat{H}$	Quantum mechanic Hamiltonian operator
$\hat{H}_{\text{el}}$	Electronic Hamiltonian operator
$\hat{T}_{\text{el}}$	Kinetic energy operator of the electrons
$\hat{T}_{\text{nu}}$	Kinetic energy operator of the nuclei
$\hat{V}_{\text{el,el}}$	Interelectronic potential energy operator
$\hat{V}_{\text{nu,nu}}$	Internuclear potential energy operator
$\hat{V}_{\text{nu,el}}$	Nuclei-electrons potential energy operator
γ	Local spin density approximation scaling factor
a,b,c	Fitting parameters for hybrid and double hybrid functionals
ρ	Electron density
ρ_{tr}	Trial electron density
τ	Kinetic energy density
ε^{KS}	Eigenvalue of one-electron Kohn–Sham operator
$\hat{h}^{\text{KS}}$	One-electron Kohn–Sham operator
$\lvert\chi^{\text{KS}}\rangle$	Kohn–Sham orbital
E_0	Ground-state electronic energy
E_{xc}	Exchange-correlation energy
$E_{\text{xc}}^{\text{DFT}}$	Density functional theory exchange-correlation energy
$E_{\text{x}}^{\text{LSDA}}$	Local spin density approximation exchange energy

E_x^{GGA}	General gradient approximation exchange energy
E_c^{GGA}	General gradient approximation correlation energy
E_x^{HF}	Hartree–Fock exchange energy
E_c^{PT2}	Second order perturbation theory correlation energy
E_x^{B88}	Exchange energy of B88 functional
E_c^{VWN}	Correlation energy of VWN functional
E_c^{LYP}	Correlation energy of LYP functional
E_{xc}^{B3LYP}	Exchange-correlation energy of the B3LYP functional
$\bar{E}$	Electronic energy functional
$\bar{E}_{xc}$	Exchange-correlation energy functional
$\bar{E}_x$	Exchange functional
$\bar{E}_c$	Correlation functional
$\bar{E}_{xc}^{LSDA}$	Local spin density approximation exchange-correlation functional
$\bar{E}_{xc}^{GGA}$	General gradient approximation exchange-correlation functional
$\bar{T}$	Electronic kinetic energy functional
$\bar{T}_{ni}$	Kinetic energy functional of noninteracting electrons
$\bar{V}_{nu,el}$	Nuclei-electrons potential energy functional
$\bar{V}_{el,el}$	Interelectronic potential energy functional
$\Delta\bar{T}$	Correction for electron interaction
$\Delta\bar{V}_{el,el}$	Correction for nonclassical electron repulsion
N_{BF}	Number of basis functions used to represent the Kohn–Sham orbitals
N_{AUX}	Number of atom-centered auxiliary basis functions used for fitting the electron density
$\mathbf{D}$	Density matrix
D_{ab}	Element of the density matrix
$\mathbf{c}$	Coefficients vector of atom-centered auxiliary basis functions
c_α	atom-centered auxiliary basis function coefficient
J	Coulomb contribution to electronic energy
Φ	Atom centered basis functions representing the Kohn–Sham orbitals
ϕ	Atom-centered auxiliary basis function used for fitting the electron density
$\tilde{\rho}$	Approximated electron density
$(ab\|cd)$	Four centered Coulomb integral in chemists notation
α^A	Polarizability of fragment A
C	Dispersion coefficient
s	Scaling parameter for dispersion correction

ω	Vibrational frequency
f_{damp}	Damping function
ψ	Dispersion integration kernel
E_{disp}	Energy contribution due to dispersion effects
$E_{\text{xc}}^{\text{KS}}$	Kohn–Sham exchange-correlation energy
$E_{\text{disp}}^{\text{APW}}$	Atom pairwise dispersion energy
E_{c}^{NL}	Non-local correlation energy
$\mathbf{H}$	Hessian matrix of second derivatives
H_{AB}	Hessian matrix element
h	Planck's constant
$\hbar$	Planck's reduced constant
ν	Vibrational quantum number
$\boldsymbol{\theta}$	Coupled mass weighted spatial coordinates
θ_A	Coupled mass weighted spatial coordinate of nucleus A
$\boldsymbol{\Theta}$	Decoupled mass weighted spatial coordinates
Θ_A	Decoupled mass weighted spatial coordinate of nucleus A
$\mathbf{K}$	Transformation matrix
K_{AB}	Element of the transformation matrix
$\mathfrak{H}$	Diagonalised Hessian matrix
$\mathfrak{H}_{AB}$	Element of the diagonalised Hessian matrix
V_{pot}	Potential energy
ε	Dielectric constant
k_{B}	Boltzmann's constant
N_{A}	Avogadro's constant
R	Universal gas constant
I	Molecular moment of inertia
I_A, I_B, I_C	Principal moments of inertia
N_{part}	Number of particles
$\mathcal{M}$	Molecular mass
P	Pressure
P_0	Standard pressure of 1 atm
$\mathfrak{s}$	Spin multiplicity
σ	Rotational symmetry number
T	Temperature
V	Volume
φ	System state
E_ϕ	Energy of system state
G	Gibbs free energy
H	Enthalpy
S	Entropy
S_{el}	Electronic contribution to entropy
S_{trans}	Translational contribution to entropy

...

$S_{\mathrm{rot}}^{\mathrm{linear}}$	Entropy contribution of a linear rotor
S_{rot}	Rotational contribution to entropy
S_{vib}	Vibrational contribution to entropy
U	Total internal energy
U_0	Internal energy at zero Kelvin
U_{el}	Electronic internal energy
U_{trans}	Translational internal energy
$U_{\mathrm{rot}}^{\mathrm{linear}}$	Rotational internal energy of linear rotor
U_{rot}	Rotational internal energy
U_{vib}	Vibrational internal energy
Q	Partition function of system
q	Molecular partition function
q_{el}	Electronic molecular partition function
q_{trans}	Translational molecular partition function
q_{rot}	Rotational molecular partition function
q_{vib}	Vibrational molecular partition function
c_1, c_2, c_3, c_4	Cutoff radii of enzyme design procedure
d_{t}	Offset for tournament size interpolation
f_{fit}	Fitness function
g_{size}	Number of genes in steady-state genome
g_{min}	Minimum number of genes in generational genome
g_{max}	Maximal number of genes in generational genome
k	Genomes substituted by steady-state replacement
k_{t}	Linear slope for tournament size interpolation
n_{eval}	Number of fitness evaluations
n_{new}	Number of new genomes
n_{off}	Number of offspring
n_{tour}	Tournament size
p_{curr}	Current population size
p_{max}	Maximal population size
r_{cx}	Crossover rate
r_{mut}	Mutation rate
r_{rand}	Random number
r_{rec}	Recombination rate
t_{crit}	Genetic operation threshold
F_{min}	Minimum fitness of current population
F_{max}	Maximal fitness of current population
F_{avg}	Average fitness of current population
G_{TS}	Gibbs free energy of transition state
G_{react}	Gibbs free energy of reactants

1

INTRODUCTION

The processes underlying life at the smallest level are an intricate network of chemical transformations, be it the digestion of food, the replication of DNA or even visual perception. Most of the reactions associated with these processes are too slow to sustain the respective functions under normal conditions, calling for the aid of some kind of catalyst. Nature has provided those in the form of enzymes, a wide range of peptide or even RNA complexes, exhibiting many fascinating traits.[1] These biocatalysts accelerate the rates of chemical transformations by lowering the reaction barrier through favourable interactions with the participating substrates.[2] Moreover, three billion years of continuous evolution have transformed enzymes into true chemical powerhouses. Some of them accelerate the turnover rates of reactions to the point, where diffusion of the substrates becomes the rate determining step.[3] Additionally, almost all of them exhibit an astonishing specificity and selectivity towards certain reactions and substrates, both traits which are rarely found to this extent in synthetically prepared catalysts.[4]

Owing to these features, enzymes are also utilized in several different processes in industry and research[5], ranging from the preparation of food to the synthesis of drugs. For these applications, nature already provides a large amount of enzyme catalysed reactions, but human inventiveness and serendipity has lead to the discovery of a plethora of novel chemical transformations unknown in nature, further augmenting the set of possible chemical transformations. The successful creation of artificial enzymes capable of catalysing all these new reactions, while still exhibiting the excellent reactivity and specificity of natural enzymes, would open up a wide range of possibilities, be it in chemical synthesis, medicine or industrial applications.

In fact, this dream is on the verge of coming true. Different approaches for designing enzymes with new functionalities have been developed in the last decades: directed evolution[6], catalytic antibodies[7], host-guest chemistry[8] and many more. With new insights into the structure and mechanism of enzymes, the advent of more and more powerful computers and advances in bioinformatics and computational chemistry, computational enzyme design has been established as a promising technique for the creation of artificial enzymes. For good reviews on this topic, see References 9 and 10.

The roots of computational enzyme design can be traced back to the manipulation of amphipathic α-helices, resulting in the creation of novel enzymes such as "Helichrome"[11] and HP-7.[12] Although the design process of these molecules involved no computational models, they provided valuable insights into general sequence-structure relationships. One of the first examples of targeted structure design incorporating computations was the application of symmetry operations to an ideal α-helix, using retrostructural analysis as guidance[13], which lead to the creation of the *duo ferro* (DF) series of metalloenzymes.[14,15]

The DF proteins helped to demonstrate the viability of simulations for elucidating and predicting relations between sequence, structure and reactivity.[16] While the use of well defined α-helix elements limits the structural and chemical diversity, it allows to bypass the inverse folding problem. An alternative strategy is to craft novel active sites onto the scaffolds of existing proteins. Early examples of this approach are the metallazymes created with the METAL SEARCH[17,18] and DEZYMER[19] programs and the protozyme design (PZD) series of non-metal enzymes.[20] In the latter case, a composite sidechain was introduced at a suitable protein backbone location and the conformations of this moiety and adjacent sidechains optimized with Dead End Elimination.[21] structural and in turn often chemical diversity. The creation of structurally diverse enzyme scaffolds through computational means alone is unfortunately hampered by the inverse folding problem. An alternative, more manageable strategy, is to craft active sites incorporating novel functionalities onto the scaffolds of existing proteins. Early examples of this approach are the metallazymes created with the METAL SEARCH and DEZYMER programs and the PZD series of non-metal enzymes. In the latter case, a protein backbone was screened for positions capable of accommodating a composite sidechain composed of the catalytic amino acid and the substrate. To optimize the conformations of this moiety and adjacent sidechains, the Dead End Elimination algorithm was used. The approach employed for the PZD enzymes can be seen as a precursor for the "inside-out" design protocol by the groups of Baker and Houk[10,22], implemented in the ROSETTA suite of programs[23] for computational protein design. This strategy uses a geometric hashing algorithm to search a set of protein crystal structures for locations, where a model active site, generated by quantum-chemical methods, can be introduced. The resulting structures are then subjected to sequence/structure design in order to stabilize catalytic interactions. The inside-out methodology has already been applied successfully on several occasions, such as the *de-novo* design of retro-aldol enzymes[24], enzymes catalysing Kemp-elimination[25] and Diels–Alder enzymes[26], as well as the redesign of a zinc metalloenzyme for hydrolysis.[27] These successes reported for the computational *de-novo* design of enzymes prove the viability of this method.

The objective of this thesis is to apply this inside-out design protocol to the computational *de-novo* design of a metallazyme catalysing an olefin metathesis reaction. This approach was chosen because of its relative robustness and reliability, as demonstrated by the examples given above.

The creation of a model of the active site, a so called "theozyme"[28], is guided by a motif commonly found in enzymes and first suggested by Pauling in 1946[29], where the reaction barrier is lowered by favourable interactions of the transition state with the protein residues. This concept is used in this work to arrive at well-suited protein structures. In order to efficiently search for beneficial geometry arrangements, a genetic algorithm was newly developed, combining high throughput quantum chemistry methods and efficient metaheuristic optimisation algorithms.[30]

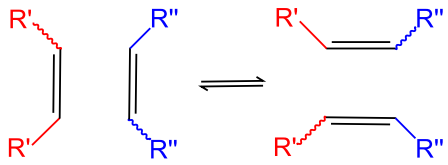


Figure 1.1: Olefin metathesis reaction. The exchange of the alkene moieties formally leads to the formation of two new double bonds.

In the second stage of the work, the Rosetta suite of programs by Baker and coworkers is used to incorporate the theozyme into the scaffold of an existing protein. To this goal, a geometric hashing algorithm is utilized to efficiently search a database of protein crystal structures for locations, where the novel active site residues can be introduced without impacting the overall conformation of the scaffold.

The modified scaffolds are then subjected to a sequence-design procedure, in order to tailor the novel active sites complementary to the substrate and stabilize the catalytic interactions. Finished designs are evaluated according to the retention of the catalytic geometry and quality of the catalytic interactions.

The target reaction in this thesis, olefin metathesis, is a formal redistribution of alkene fragments and thus one of the few organic reactions capable of creating new carbon bonds (Figure 1.1). Its overall efficiency and lack of undesired byproducts in combination with the advent of well-defined catalysts has increased the popularity of metathesis over the last decades[31], leading to a variety of applications in industry[32] and organic synthesis[33], as for example the shell higher olefins process, cross metathesis (CM), ring-opening metathesis polymerization (ROMP) and ring-closing metathesis (RCM). These properties and the fact that there is no analogous reaction in nature, make olefin metathesis the perfect target for introducing a novel functionality into an enzyme.

This thesis is conducted in the framework of a cooperation between two theoretical groups. The design of the active site was done in the quantum chemistry group of Univ.-Prof. Dr. L. González and guided by Dr. P. Marquetand. The search for adequate scaffolds and the subsequent sequence design was carried out in the bioinformatics group of Univ.-Prof. Dr. I. Hofacker with the help of Dr. C. Flamm, who also initiated the project. The most promising enzyme designs will be synthesized and characterized with regards to their functionality by the experimental group of Univ.-Prof. Dr. C. F. W. Becker.

2

ON DE-NOVO ENZYME DESIGN

The promise of creating artificial enzymes with attributes similar to their natural counterparts, but at the same time bearing novel functionalities is tantalizing. This would allow for the specific design of new molecular species, capable of catalysing every possible chemical reaction under "green" conditions – mild, aqueous environments – while exhibiting unprecedented reaction speed, proficiency and selectivity.

Unfortunately enzyme catalysis is a highly complex interplay of a wide range of different effects. The observed rate enhancement and selectivity of enzymatic reactions is the result of a combination of several fine tuned interactions between the substrate, the active site and the protein scaffold as a whole. Active sites exhibit a geometry highly complementary to the transition state of the substrate, stabilizing it through electrostatic, covalent or steric interactions.[29,34] Furthermore, reactants are often preorganized before catalysis[35], while second shell interactions lead to a fine tuning of the reaction energy landscape[36], inextricably coupling the active site to the outer regions of the enzyme. Some enzymes also exhibit allosteric effects, dynamically changing their conformations allowing for a further refinement of the enzymatic process. This intricate network of interactions renders the *de-novo* design of enzymes a formidable task.

To alleviate the encountered problems, different approaches are employed, including directed evolution[6], catalytic antibodies[7], host-guest chemistry[8], supramolecular chemistry[8], protein engineering[37] and many more. The one focused on here is computational enzyme design. This relatively young field of research has profited much from recent developments in hardware, new insights in bioinformatics and the enzymatic workings, as well as the advent of high throughput methods in quantum chemistry and has thus been established as a promising method in *de-novo* enzyme design. Of special interest is a variant of computational enzyme design pioneered by the groups of Baker and Houk, the so called *inside-out* design protocol, which has proven to be a reliable method of introducing new functionality into enzymes and will be discussed in more detail. For a good review on the *inside-out* methodology and enzyme design in general, see reference 10.

2.1 INSIDE-OUT APPROACH

Creating an entire enzyme from scratch would introduce the need of solving the inverse folding problem, the prediction of stable tertiary structures associated with certain amino acid sequences. Since protein structure prediction has to explore a vast combinatorial structure space, it requires immense computational resources and has thus far only been applied successfully in limited number of cases.

The reverse folding problem can be avoided by crafting novel active sites onto the scaffolds of already existing peptides.[22] As at least one sequence of amino acids folding into the required tertiary conformation is already known, extensive structure prediction is not required. Based on this observation, the design process can be separated into different stages, depicted schematically in Figure 2.1.

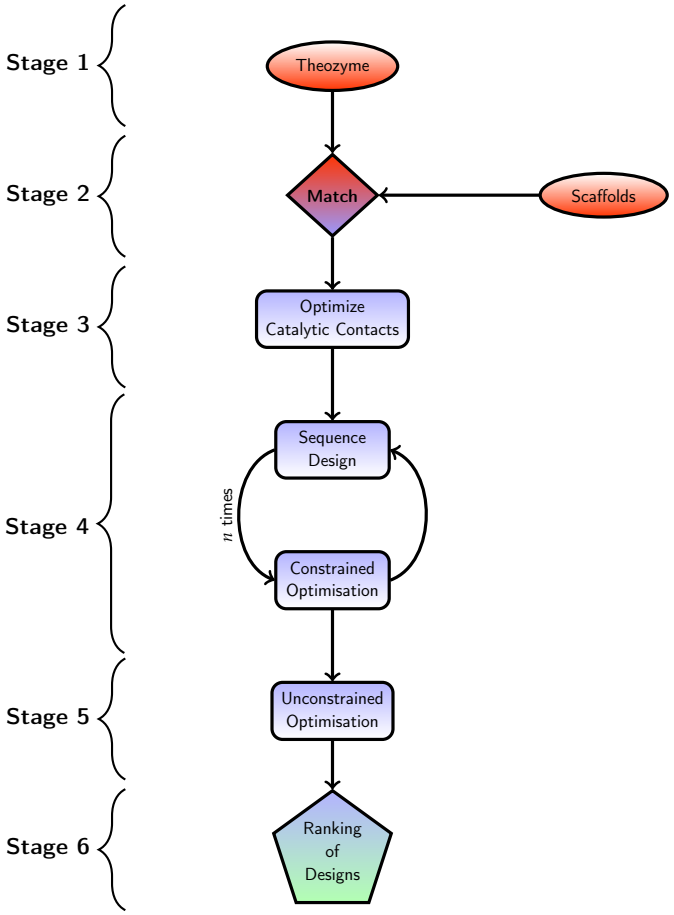


Figure 2.1: Different stages invoked during the inside-out design process, starting from the theozyme and a library of peptide scaffolds.

As a starting point for inside-out enzyme design serves a "theoretical enzyme", a so-called theozyme.[28] The theozyme consists of the transition state structure of the reaction to be modelled, surrounded by an array of stabilizing amino acid residues. For the nomenclature used throughout this thesis, see Figure 2.2.

This theozyme is typically generated using quantum chemistry calculations with the aim of optimising the interactions between the transition state and the protein by the geometrically favourable placement of different functional amino acid groups (see Figure 2.1, Stage 1).

The next step is the screening of existing protein scaffolds for backbone sites capable of accommodating the three-dimensional arrangement of amino acid residues in the theozyme (Stage 2). A rotamer library of possible conformations of the catalytic residues is generated and matched against a library of crystal structures of existing proteins. This screening can be done in an efficient manner by utilizing a geometric hashing algorithm.[22] Since the exact theozyme geometry

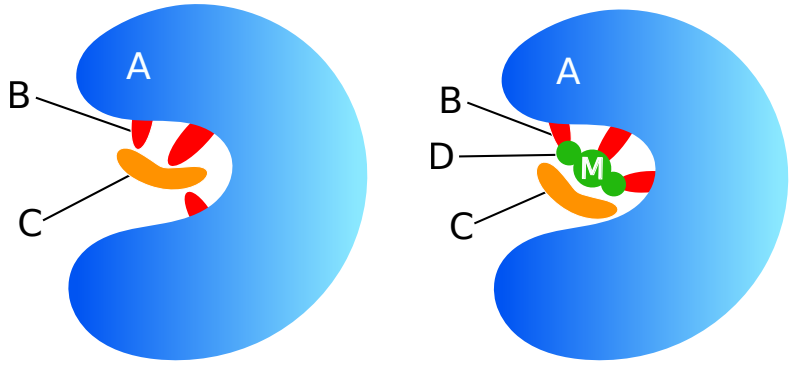


Figure 2.2: The left side shows the conventional classification used in enzyme design. The protein scaffold compromises the catalytic sidechains (B) and the remaining protein (A). The catalytic residues (B) and the part of A closest to the cavity form the active site capable of accommodating the substrate (C). The combination of substrate and catalytic residues is the theozyme. In order to provide a description for the theozymes encountered in this thesis, the scheme was augmented. The active site now bears a metal moiety (D), which will be referred to as the minimal catalyst. In this case the theozyme consists of the catalytic residues (B), the minimal catalyst (D) and the substrate (C). The combination of only the minimal catalyst and the substrate will be called the metal-substrate complex.

produces matches only in the rarest cases, tolerance values are usually assigned to the internal coordinates representing the catalytic contacts.

Once a match is found, the associated sequence structure pair is subjected to an optimisation procedure. First, the catalytic contacts between protein scaffold and substrate are optimised (Stage 3). Afterwards alternating steps of side-chain sampling and structure optimisation are applied in an iterative manner under retention of the catalytic restraints (Stage 4). This is then followed by an unconstrained minimisation cycle in order to check the integrity of the obtained artificial enzyme (Stage 5).

In a last step the obtained designs can be ranked according to different criteria (Stage 6), representing their capability of stabilizing the transition state. Typical descriptors are energy, ligand binding scores and active site geometry, but it is also possible to use molecular dynamic simulations for more detailed assessments.[38]

The power of the inside-out enzyme design protocol lies in its relative simplicity and efficiency, resulting in a robust methodology. It has in fact been employed successfully in several cases such as the *de-novo* design of an enzyme catalysing a Kemp elimination reaction[25], and many more.[24,26,27,39] This attests to the feasibility of the approach, despite the use of several approximations.

3

ON OLEFIN METATHESIS

Known since the middle of the last century, olefin metathesis has undergone a major renaissance recently. As this reaction describes a metal-catalysed rearrangement of olefinic double bonds (Figure 3.1), it is one of the few organic methods capable of forming covalent carbon-carbon bonds. Because of this ability, olefin metathesis has found widespread use in total synthesis and polymer chemistry.[33]

Examples for different applications of this reaction include acyclic diene metathesis, cross metathesis, ring-opening metathesis polymerization and ring-closing metathesis.

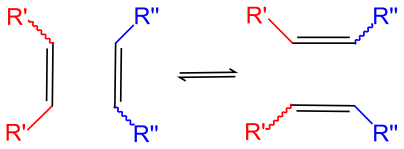
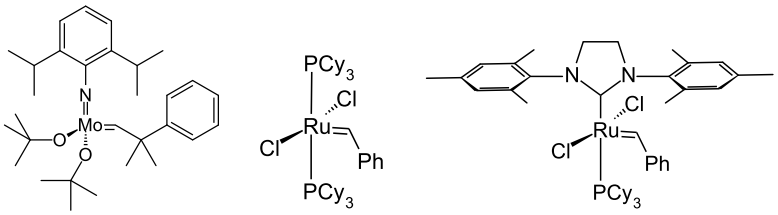


Figure 3.1: Olefin metathesis reaction. The exchange of the alkene moieties formally leads to the formation of two new double bonds.

3.1 OLEFIN METATHESIS CATALYSTS

One important reason for the rapid increase in popularity of olefin metathesis was the discovery of well-defined catalysts. These are typically transition metal complexes carrying an alkylidene moiety. Amongst the most prominent representatives of metathesis catalysts are Schrock-alkylidenes[40] and the so called Grubbs-type catalysts (Figure 3.2). The Grubbs-type catalysts are commonly further classified into first-[41] and second-generation complexes[42], of which the latter will be in the focus of this thesis.



(a) (b) (c)

Figure 3.2: Different catalysts facilitating olefin metathesis: The Schrock-alkylidene (a) and first-generation Grubbs (b) as well as second-generation Grubbs-type (c) catalysts.

Grubbs-type catalysts are ruthenium-based and exhibit improved stability towards environmental influences and tolerance towards different functional groups when compared to other catalysts, making them the standard choices for olefin metathesis reactions in synthesis.[43]

The first-generation of Grubbs catalyst carries two bulky phosphine ligands, typically cyclohexyl-phosphine, and a benzylidene group. The second generation of catalysts improve upon the first-generation Grubbs catalysts by replacing one of the phosphine ligands with a N-heterocyclic carbene species with mesityl groups, thus greatly increasing the activity and functional group tolerance by virtue of beneficial steric and electronic effects.[43–45]

3.2 MECHANISM

A reasonable mechanism of olefin metathesis was first proposed by Herisson and Chauvin in 1970[46] and is thought to proceed via a series of formal [2+2] cycloadditions followed by cycloreversions. Extensive theoretical and experimental studies accompanying the rise in popularity of the olefin metathesis reaction have helped to further elucidate certain aspects of the reaction pathway.

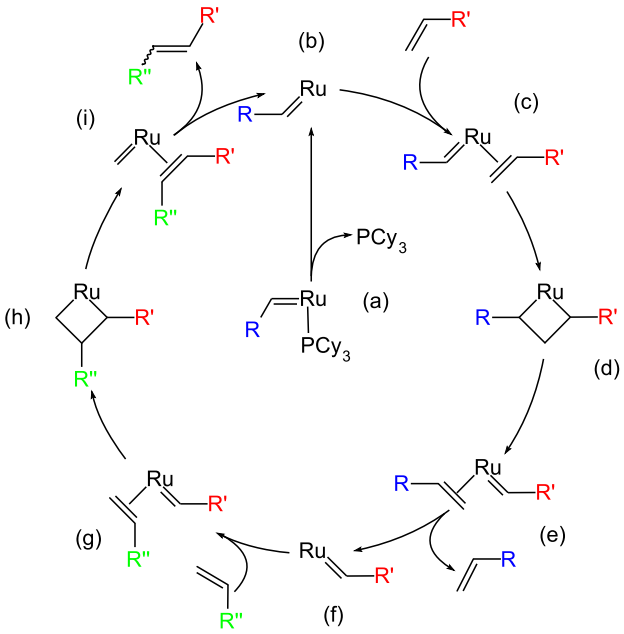


Figure 3.3: General mechanism of the olefin metathesis reaction for the Grubbs-type catalysts. The reaction cycle starts with the dissociation of the phosphine ligand from the initial catalyst (a). The main ligand and the chlorines are omitted for reasons of clarity

In case of the Grubbs-type catalysts (see Figure 3.3 for a full reaction cycle), the first step of the reaction is now commonly accepted to be the dissociation of a phosphine ligand (a), leading to a 14-electron species (b).[47,48] Next, the olefin coordinates *trans* to the ruthenium atom, resulting in a 16-electron π-complex (c).[49] The metathesis reaction then proceeds by insertion of the olefin into the metal-alkylidene bond, forming a metallacycle intermediate (d), thus resembling a [2+2] cycloaddition. In a final step the ruthenacycle undergoes ring opening and transforms into the products (f) via a second π-complex (e) and dissociation of the newly formed olefin-fragment.[49,50] This last step is generally favoured for entropic reasons. Repetition of these steps (g-i) yields the methylidene analogue of the initial active catalyst, as well as the final product and constitutes a full metathesis cycle. In

the case of ring-closing metathesis, the olefinic substrate bears two double bond moieties. The initial reaction proceeds normally, yielding the reactant-alkylidene complex. The subsequent steps, however, involve an intramolecular cycloaddition and cycloreversion, leading to the formation of a cyclic product, hence the reactions name (for an example see Figure 6.2).

The rate limiting step depends on the type of the catalyst and is usually either the initial dissociation of one phosphine ligand or the ring-closure.[49,51] The influence of the final transition state on the whole reaction profile is even more prominent in ring-closing metathesis, since the resulting species is subject to considerable ring strain effects.[52,53]

3.3 THE MODEL REACTION

The ultimate goal of this work is the development of an enzyme bearing a novel functionality. Olefin metathesis, and especially ring-closing metathesis are excellent target reactions, due to several reasons.

As mentioned above, the ability to form new carbon bonds makes this reaction a valuable tool for synthesis. The incorporation into a peptide scaffold would furthermore allow olefin metathesis to be carried out in aqueous environment, a property highly sought after in metathesis catalysts. In addition, no reaction resembling olefin metathesis is known in nature, rendering it indeed a novel functionality.

Because of these traits, the RCM of diallylether (Figure 3.4) was chosen as the model reaction for the *de-novo* enzyme design.

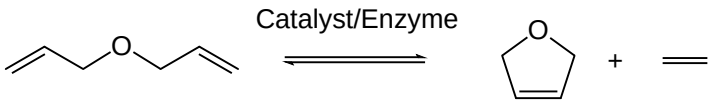


Figure 3.4: Ring-closing metathesis of diallylether forming dihydrofurane and ethene.

An advantage of this RCM is the formation of ethene, driving the reaction towards the products through the associated increase in entropy. Diallylether was chosen as a substrate due to its water solubility and size. The length of the molecule allows for the formation of a 5-membered ring, the smallest ring RCM is capable of creating. Since 5-membered rings have been shown to preferably adopt an envelope conformation during RCM[52], an intensive sampling of conformation space, as would be required for the different stereoisomers of larger rings, is not necessary. The lack of superfluous functional groups also makes it easier to assess the individual influences of the moieties present in the ether. One last benefit of using a substrate of smaller size is the saving in computational time, especially for larger systems, like the ether-catalyst complex. While the methods based on density functional theory (DFT) used in this study are quite efficient when compared to other quantum-mechanical methods, their scaling of $O(N^3)$ with the number of atoms can lead to significant reductions in computational effort for only small reductions in system size.

4

THEORETICAL BACKGROUND

The following chapter is meant to give an overview over the theoretical aspects underpinning this work. A short summary of the Schrödinger equation and the Born–Oppenheimer approximation will be provided, followed by a discussion of DFT. Geometry optimisation, normal mode analysis and computation of thermodynamic properties will be reviewed. Afterwards an introduction to genetic algorithms will be given. The final topic will be geometric hashing algorithms.

4.1 THE SCHRÖDINGER EQUATION

At the core of non-relativistic, nonadiabatic quantum mechanics lies the Schrödinger equation, proposed by Erwin Schrödinger in 1927.[54–56] In its time-independent form

$$\hat{H}(\mathbf{R},\mathbf{r})\,|\Psi(\mathbf{R},\mathbf{r})\rangle = E_{\text{total}}\,|\Psi(\mathbf{R},\mathbf{r})\rangle\,, \tag{4.1.1}$$

it describes the stationary states represented by the wavefunction $|\Psi(\mathbf{R},\mathbf{r})\rangle$ of a quantum mechanical system. E_{total} is the total energy of the respective state and $\hat{H}(\mathbf{R},\mathbf{r})$ is the quantum-mechanical Hamiltonian. This operator represents the interactions of the nuclei and electrons and thus depends on nuclear and electronic coordinates $\mathbf{R}$ and $\mathbf{r}$. It is composed of the terms

$$\hat{H}(\mathbf{R},\mathbf{r}) = \hat{T}_{\text{el}}(\mathbf{r}) + \hat{T}_{\text{nu}}(\mathbf{R}) + \hat{V}_{\text{el,el}}(\mathbf{r}) + \hat{V}_{\text{nu,el}}(\mathbf{R},\mathbf{r}) + \hat{V}_{\text{nu,nu}}(\mathbf{R}), \tag{4.1.2}$$

where $\hat{T}_{\text{el}}(\mathbf{r})$ and $\hat{T}_{\text{nu}}(\mathbf{R})$ are the electronic and nuclear kinetic energies, $\hat{V}_{\text{el,el}}(\mathbf{r})$ and $\hat{V}_{\text{nu,nu}}(\mathbf{R})$ account for the electron and nuclear Coulomb repulsions and $\hat{V}_{\text{nu,el}}(\mathbf{R},\mathbf{r})$ describe the interactions between nuclei and electrons.

For a molecule with n electrons and N nuclei the contributions of the kinetic energies to the overall Hamiltonian in atomic units can be written as

$$\hat{T}_{\text{el}}(\mathbf{r}) = -\sum_{i=1}^{n}\frac{1}{2}\nabla_i^2, \tag{4.1.3}$$

$$\hat{T}_{\text{nu}}(\mathbf{R}) = -\sum_{A=1}^{N}\frac{1}{2M_A}\nabla_A^2, \tag{4.1.4}$$

where M_A is the mass of nucleus A. The potential energy terms can be expressed as

$$\hat{V}_{\text{el,el}}(\mathbf{r}) = \sum_{i=1}^{n}\sum_{j>i}^{n}\frac{1}{r_{ij}}, \tag{4.1.5}$$

$$\hat{V}_{\text{nu,nu}}(\mathbf{R}) = \sum_{A=1}^{N}\sum_{B>A}^{N}\frac{Z_A Z_B}{R_{AB}}, \tag{4.1.6}$$

$$\hat{V}_{\text{nu,el}}(\mathbf{R},\mathbf{r}) = -\sum_{A=1}^{N}\sum_{i=1}^{n}\frac{Z_A}{r_{Ai}}, \tag{4.1.7}$$

where $r_{ij} = |r_i - r_j|$ is the interelectronic distance, $R_{AB} = |R_A - R_B|$ the internuclear distance and $r_{Ai} = |R_A - r_i|$ the distance between nucleus A and electron i.

4.2 BORN–OPPENHEIMER APPROXIMATION

Solving equation 4.1.1 for molecules poses an insurmountable obstacle. It constitutes a many-body-problem and thus lacks analytical solutions for systems containing more than two particles.

To overcome this hurdle, Born and Oppenheimer introduced an approximation in 1927, which allows for the separate treatment of nuclei and electrons.[57] Because of their smaller mass, electrons are much faster than nuclei. Therefore, electrons can be considered as moving in the field of the stationary nuclei. The wavefunction of the system can then be written as

$$|\Psi(\mathbf{R},\mathbf{r})\rangle = |\Psi_{\mathrm{nu}}(\mathbf{R})\rangle\,|\Psi_{\mathrm{el}}(\mathbf{r};\mathbf{R})\rangle\,, \tag{4.2.1}$$

where $|\Psi_{\mathrm{nu}}(\mathbf{R})\rangle$ is the nuclear wavefunction and $|\Psi_{\mathrm{el}}(\mathbf{r};\mathbf{R})\rangle$ the electronic wavefunction depending parametrically on the nuclear coordinates $\mathbf{R}$. Since quantum chemistry is mainly concerned with electronic properties, the nuclear kinetic energy terms can be omitted leading to the electronic Schrödinger equation in the form

$$\left(\hat{H}_{\mathrm{el}}(\mathbf{r};\mathbf{R}) + \hat{V}_{\mathrm{nu,nu}}(\mathbf{R})\right)|\Psi_{\mathrm{el}}(\mathbf{r};\mathbf{R})\rangle = E_{\mathrm{el}}\,|\Psi_{\mathrm{el}}(\mathbf{r};\mathbf{R})\rangle\,. \tag{4.2.2}$$

$\hat{V}_{\mathrm{nu,nu}}(\mathbf{R})$ is a constant for a given set of nuclear coordinates and E_{el} is the effective potential energy of electrons moving in the field of the fixed nuclei. The multidimensional surface spanned by the values of E_{el} obtained for different nuclear positions is called the potential energy surface (PES), an important concept in theoretical chemistry. The electronic Hamiltonian $\hat{H}_{\mathrm{el}}(\mathbf{r};\mathbf{R})$ is given by the relation

$$\hat{H}_{\mathrm{el}}(\mathbf{r};\mathbf{R}) = -\sum_{i=1}^{n}\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{n}\sum_{j>i}^{n}\frac{1}{r_{ij}} - \sum_{A=1}^{N}\sum_{i=1}^{n}\frac{Z_A}{r_{Ai}}. \tag{4.2.3}$$

The *ab initio* methods, like Hartree–Fock-theory[58], are based on equation 4.2.2 and try to derive solutions of increasing quality for the electronic problem.

4.3 DENSITY FUNCTIONAL THEORY

A drawback of the wavefunction is its lack of a direct physical interpretation. Furthermore, it depends on $3N$ spatial and N spin coordinates and thus contains more information than strictly needed for the treatment of a system. This situation has motivated the search for an alternative function involving fewer variables, preferably based on a physical observable, which allows for the computation of the energy and other molecular properties.

Equation 4.2.3 shows the dependence of the electronic Hamiltonian on only the nuclear coordinates, atomic numbers and the total number of electrons. A suitable observable is therefore the electron probability density $\rho(\mathbf{r})$, a function of only three spatial variables. For a given $\rho(\mathbf{r})$,

it should then be possible to derive the energy and other properties of the electronic state associated with this density. The required formalism is provided by DFT.

4.3.1 *Hohenberg–Kohn Theorems*

A rigorous foundation for DFT was provided by Hohenberg and Kohn in 1964 with the so-called Hohenberg–Kohn theorems.[59]

According to the existence theorem, the ground-state electron density is uniquely related to the external potential of the nuclei and thus defines the Hamiltonian, the wavefunction and all other molecular properties. The ground-state electronic energy E_0 and other quantities can then be expressed as functionals of the electron density $\rho(\mathbf{r})$:

$$E_0 = \bar{E}[\rho(\mathbf{r})] = \bar{T}[\rho(\mathbf{r})] + \bar{V}_{\mathrm{nu,el}}[\rho(\mathbf{r})] + \bar{V}_{\mathrm{el,el}}[\rho(\mathbf{r})]. \tag{4.3.1}$$

The terms on the right hand side refer to the kinetic energy and the interaction potentials between nuclei and electrons and between electrons (compare to equation 4.1.2).

The Hohenberg–Kohn variational theorem states that the density obeys a variational principle. This infers, that every trial electron density $\rho_{\mathrm{tr}}(\mathbf{r})$ yields an energy equal to or above the exact ground state energy

$$\bar{T}[\rho_{\mathrm{tr}}(\mathbf{r})] + \bar{V}_{\mathrm{nu,el}}[\rho_{\mathrm{tr}}(\mathbf{r})] + \bar{V}_{\mathrm{el,el}}[\rho_{\mathrm{tr}}(\mathbf{r})] \geq \bar{E}[\rho(\mathbf{r})] = E_0. \tag{4.3.2}$$

This relation would allow for a systematic improvement of the energy. Unfortunately it only holds for the exact functional $\bar{E}[\rho(\mathbf{r})]$, a problem of DFT which will be discussed later on.

4.3.2 *Kohn–Sham Equations*

Despite their importance, the Hohenberg–Kohn theorems give no directions as how to compute the ground state energy E_0 or how to derive the density without knowledge of the wavefunction. An important step towards the practical application of density functional theory was therefore undertaken by the introduction of the Kohn–Sham formalism in 1965.[60]

By assuming a fictitious system of non interacting electrons with the same ground state density as the real system of interest, one can split the energy functional:

$$\begin{aligned}\bar{E}[\rho(\mathbf{r})] =& \bar{T}_{\mathrm{ni}}[\rho(\mathbf{r})] + \bar{V}_{\mathrm{nu,el}}[\rho(\mathbf{r})] + \bar{V}_{\mathrm{el,el}}[\rho(\mathbf{r})] \\ &+ \Delta\bar{T}[\rho(\mathbf{r})] + \Delta\bar{V}_{\mathrm{el,el}}[\rho(\mathbf{r})],\end{aligned} \tag{4.3.3}$$

where $\bar{T}_{\mathrm{ni}}[\rho(\mathbf{r})]$ is the kinetic energy of the non-interacting electrons and the last two terms are corrections due to the interacting nature of electrons and non-classical electron-electron repulsion. The kinetic energy can now be computed by taking the sum over the individual electronic contributions. After expressing the density as Kohn–Sham orbitals χ^{KS} according to

$$\rho(\mathbf{r}) = \sum_{i=1}^{n} |\chi_i^{\mathrm{KS}}\rangle \langle\chi_i^{\mathrm{KS}}|, \tag{4.3.4}$$

equation 4.3.3 can be rewritten for a system of n electrons and N nuclei as

$$\begin{aligned}\bar{E}[\rho(\mathbf{r})] = &-\frac{1}{2}\sum_{i=1}^{n}\left\langle \chi_i^{\mathrm{KS}}\right| \nabla_i^2 \left|\chi_i^{\mathrm{KS}}\right\rangle - \sum_{A=1}^{N}\frac{Z_A}{R_A}\int\frac{\rho(\mathbf{r})}{R_A}d\mathbf{r} \\ &+\frac{1}{2}\iint\frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}d\mathbf{r}' + \bar{E}_{\mathrm{xc}}[\rho(\mathbf{r})].\end{aligned} \tag{4.3.5}$$

The last two correction terms in equation 4.3.3 have been contracted into $\bar{E}_{\mathrm{xc}}[\rho(\mathbf{r})]$, the so-called exchange-correlation functional. Electron exchange describes a quantum-mechanical repulsive interaction arising due to the fermionic nature of electrons, while electron correlation is the dependency of one electron's probability density on the probability densities of all other electrons in the system and *vice versa*. The Kohn–Sham orbitals are found by solving the pseudo-eigenvalue equations

$$\hat{h}_i^{\mathrm{KS}}\left|\chi_i^{\mathrm{KS}}\right\rangle = \varepsilon_i^{\mathrm{KS}}\left|\chi_i^{\mathrm{KS}}\right\rangle, \tag{4.3.6}$$

where the one-electron Kohn–Sham operator satisfies

$$\hat{h}_i^{\mathrm{KS}} = -\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{N}\frac{Z_A}{r_{iA}} + \int\frac{\rho(\mathbf{r}')}{|\mathbf{r}_i-\mathbf{r}'|}d\mathbf{r}' + \frac{\partial\bar{E}_{\mathrm{xc}}[\rho(\mathbf{r})]}{\partial\rho(\mathbf{r})}. \tag{4.3.7}$$

Applying this formalism, the ground-state energy can be computed in a self-consistent field procedure.

Because Kohn–Sham theory involves no approximations, equation 4.3.5 yields exact ground state energies, provided the correct exchange-correlation functional is known. As this is unfortunately not the case, great endeavours are undertaken to find functions that allow reasonable approximations of $\bar{E}_{\mathrm{xc}}[\rho(\mathbf{r})]$.

4.3.3 *Exchange-Correlation Functionals*

Since $\bar{E}_{\mathrm{xc}}$ not only has to account for quantum mechanical electron exchange and correlation, but also for the difference in the kinetic energy of an interacting system, it is no trivial task to derive a suitable approximation.

Different approaches to this end have resulted in various expressions for the exchange-correlation functional. Perdew suggested grouping them in the form of a ladder, where each rung improves upon the former one in complexity, sophistication and, in general, accuracy.[61] A description of the individual "rungs" is provided in this section.

Local Spin Density Approximation

Local Density Approximation (LDA) and its open-shell analogue Local Spin Density Approximation (LSDA) are based on the analysis of the uniform electron gas. The term "local" indicates the dependence of $\bar{E}_{\mathrm{xc}}$ at a given position on only the local electron density.

The exchange energy of an uniform electron gas can be derived exactly and is given by

$$\bar{E}_{\mathrm{x}}[\rho(\mathbf{r})] = -\frac{9}{8}\left(\frac{3}{\pi}\right)^{\frac{1}{3}}\gamma\iiint\rho^{\frac{4}{3}}(\mathbf{r})d\mathbf{r}, \tag{4.3.8}$$

where γ is a parameter varying for different approaches.[62,63]

For the correlation part, no analytical solution based on the uniform electron gas exists. Correlation functionals are therefore usually constructed by fitting to computed densities of the uniform electron gas. A popular example is the VWN correlation functional by Vosko, Wilk and Nusair[64], obtained by fitting to data derived via quantum Monte Carlo methods.

General Gradient Approximation

As the electron density of a molecule is by no means spatially uniform, the LSDA approach is heavily limited in its applicability. A way to improve upon LSDA is making $\bar{E}_{\text{xc}}$ not only depend on the local density, but also on the change in density. This is facilitated by using a Taylor-expansion-like formalism and including the gradient of the density into the functional.

The resulting model is called the generalized gradient approximation (GGA). The new term is typically treated as a correction to a LSDA functional in the form

$$\bar{E}_{\text{xc}}^{\text{GGA}}[\rho(\mathbf{r})] = \bar{E}_{\text{xc}}^{\text{LSDA}}[\rho(\mathbf{r})] + \Delta\bar{E}_{\text{xc}}\left[\frac{|\nabla\rho(\mathbf{r})|}{\rho^{\frac{4}{3}}(\mathbf{r})}\right]. \tag{4.3.9}$$

Most GGA functionals incorporate empiric parameters.

Examples for exchange functionals include Becke's B88 functional[65] and the parameter-free PBE functional of Perdew, Burke and Ernzerhof.[66] Popular correlation GGA functionals are P86 by Perdew[67], PW91 by Perdew and Wang[68] and the LYP functional by Lee, Yang and Parr[69], derived from the Colle–Salvetti formula. In contrast to other GGA functionals LYP does not compute a correction to the LSDA expression, but the total correlation energy. For practical use exchange and correlation GGA-functionals are combined. This leads to functionals like BP86, using the B88 exchange and P86 correlation terms.

meta-GGA

Given the Taylor-expansion-like nature of equation 4.3.9, one might further enhance the GGA-formalism by including the Laplacian of the electron density.

This approach is usually referred to as meta-GGA, since it goes beyond the normal gradient approximation. However, numerically stable computations of the Laplacian are quite demanding. An alternative is to incorporate a dependence on the kinetic energy density τ into the exchange correlation functional, defined as

$$\tau(\mathbf{r}) = \frac{1}{2}\sum_{i}^{\text{occ.}} |\nabla\chi_i^{\text{KS}}(\mathbf{r})|^2, \tag{4.3.10}$$

where the index i runs over all occupied Kohn–Sham orbitals $\chi_i^{\text{KS}}(\mathbf{r})$. Examples for meta-GGA functionals are B95[70] and TPSS.[71]

Hybrid Functionals

The next step towards an accurate expression of E_{xc} is the partial inclusion of exact, nonlocal exchange, as computed by the Hartree–Fock (HF) method. These functionals have the general form of

$$E_{xc} = (1-a)E_{xc}^{DFT} + aE_x^{HF}, \tag{4.3.11}$$

where a is an empirical constant regulating the admixture of exact HF-exchange.

Introducing more parameters and extensive fitting to experimental and computational data has lead to a wide range of so called hybrid functionals. One of the most popular representatives is the B3LYP method[72] defined as

$$\begin{aligned} E_{xc}^{B3LYP} = (1-a)E_x^{LSDA} + aE_x^{HF} + bE_x^{B88} \\ +(1-c)E_c^{VWN} + cE_c^{LYP}. \end{aligned} \tag{4.3.12}$$

The parameters a, b and c were optimized to $a = 0.20$, $b = 0.72$ and $c = 0.81$ by fitting to empirical data. Other examples for hybrid functionals are PBEo[73] and TPSSh.[74]

Double Hybrid Functionals

The highest rung of Perdew's ladder is reached by introducing part of the exact correlation energy as an extension to the hybrid functional formalism. These so-called double hybrid functionals are defined as

$$E_{xc} = (1-a)E_x^{GGA} + aE_x^{HF} + bE_c^{GGA} + (1-b)E_c^{PT2}, \tag{4.3.13}$$

where E_c^{PT2} is the correlation contribution computed by second order perturbation theory. An example for double hybrid functionals is the B2PLYP functional of Grimme.[75]

4.3.4 *Resolution of Identity*

Despite its ultimately approximate nature, DFT is amongst the most popular electronic structure methods. One of the main reasons is efficiency, as attested by a scaling behaviour of $O(N^3)$. When compared to typical scalings of *ab initio* methods ranging from $O(N^4)$ to $O(N^7)$, DFT allows for the treatment of much larger systems at acceptable accuracy.

An additional increase in computation speed can be gained by utilizing the resolution of identity approach or RI-J, first introduced by Almlöf *et al.*[76] and further refined by Ahlrichs and coworkers.[77]

The bottlenecks in equation 4.3.5 are the evaluation of the exchange correlation functional and the Coulomb term

$$J = \iint \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}d\mathbf{r}'. \tag{4.3.14}$$

E_{xc} can be treated efficiently using numerical quadratures, which leaves J as the dominating term.

Due to the representation of the Kohn–Sham orbitals in N_{BF} atom-centered basis functions Φ:

$$\rho(\mathbf{r}) = \sum_{a=1}^{N_{BF}} \sum_{b=1}^{N_{BF}} D_{ab}\Phi_a(\mathbf{r})\Phi_b(\mathbf{r}), \tag{4.3.15}$$

where **D** is the density matrix, the evaluation of J gives rise to four-center two-electron repulsion integrals. The cost of resolving these integrals in dependence on the number of basis functions grows with the fourth power.

The resolution of identity approximation is based on the observation, that describing $\rho(\mathbf{r})$ in terms of equation 4.3.15 holds an surplus of information. It is therefore possible to approximate the density by a linear combination of N_{AUX} atom-centered auxiliary basis functions ϕ:

$$\rho(\mathbf{r}) \simeq \tilde{\rho}(\mathbf{r}) = \sum_{\alpha=1}^{N_{\mathrm{AUX}}} c_\alpha \phi_\alpha(\mathbf{r}). \tag{4.3.16}$$

The coefficients c_α are obtained by a fitting procedure according to

$$\iint \frac{\delta\rho(\mathbf{r})\delta\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r} d\mathbf{r}' = \text{min.}, \tag{4.3.17}$$

where $\delta\rho(\mathbf{r}) = \rho(\mathbf{r}) - \tilde{\rho}(\mathbf{r})$. Using the chemists notation to represent the integrals[58], it can be shown that expression 4.3.17 leads to a replacement of the four center integrals by a system of linear equations consisting of only two- and three centered integrals,

$$(ab|cd) = \sum_{\alpha\beta}^{N_{\mathrm{AUX}}} (ab|\alpha)(\alpha|\beta)^{-1}(\beta|cd). \tag{4.3.18}$$

This representation resembles the resolution of identity in Hilbert space, hence the method's name. Employing this approach leads to a 10-fold saving in computation time, while introducing only minor errors because of the finite auxiliary basis set.[77]

An extension of RI-J is the multipole accelerated resolution of identity method, for short MARI-J, developed by Ahlrichs and coworkers.[78] Here the Coulomb interactions are partitioned into near- and far-field terms. The near field part is then evaluated via RI-J, while the far-field contribution is treated using multipole expansions. For large systems, this separate treatment can lead to an additional saving in CPU-time of a factor of 6.5 compared to traditional RI-J.[78]

4.3.5 *Dispersion Correction*

One major shortcoming of DFT caused by the approximation of the exchange-correlation term is the neglect of long range electron-electron correlation. This omission results in the inability of standard functionals to correctly account for dispersion effects. Typical examples are the van-der-Waals (vdW) bound ground states of noble gas dimers and the benzene dimer, which DFT fails to locate.[79]

Since the magnitude of dispersion interactions can easily outrank all other interactions especially in larger biological and chemical compounds or complexes, an accurate description is necessary. This need has lead to the development of several dispersion correction schemes over the past few years. Most employ the general strategy

$$E_{\mathrm{xc}} = E_{\mathrm{xc}}^{\mathrm{KS}} + E_{\mathrm{disp}}, \tag{4.3.19}$$

where the dispersion term E_{disp} is added to the Kohn–Sham energy in a post SCF manner. They can be grouped into three classes: atom

pairwise additive schemes, charge density dependent approaches and extensive parametrization.

Atom pairwise (APW) additive schemes are based on the approximation that dispersion can be expressed as additive atom pair interactions. These corrections are dependent on the distance R_{AB} between the two atoms A and B. They typically contain a $1/R_{AB}^6$ term resembling dipole-dipole interactions and sometimes terms of higher power:

$$E_{\text{disp}}^{\text{APW}} = -\sum_{A>B}\sum_{n=6(8,10)} s_n \frac{C_n^{AB}}{R_{AB}^n} f_{\text{damp}}(R_{AB}), \tag{4.3.20}$$

where s_n is a fitted scaling parameter, C_n^{AB} are the dispersion coefficients and $f_{\text{damp}}(R_{AB})$ is a damping function employed to switch off the dispersion correction for regions where it is sufficiently described by the functional. The main differences of the various atom pairwise additive corrections are the choice of the damping function and the way the dispersion coefficients are obtained.

An early scheme is DFT-D2 by Grimme[80] which uses tabulated values for C_n^{AA} and computes C_n^{AB} as the geometric mean.

More advanced solutions introduce a system dependence of the dispersion coefficients. The DFT-D3 approach[81] interpolates C_n^{AB} between different hybridizations according to the system geometry. In this case, the pure dispersion coefficients were computed using the Casimir–Polder relation[82]

$$C_n^{AB} = \frac{3}{\pi}\int_0^{\text{inf}} \alpha^A(i\omega)\alpha^B(i\omega)d\omega. \tag{4.3.21}$$

The fragment polarizabilities α were obtained from time-dependent DFT (TD-DFT) calculations.

Another approach termed DFT+vdW[83] also derives C_n by the Casimir–Polder relation with TD-DFT results as a basis. To account for system dependence, the coefficients are then modified utilizing the Hirshfeld partitioned volume of the bound and free atom.

The exchange dipole moment scheme XDM by Becke and Johnson[84–86] employs second-order perturbation theory to compute the coefficients from the dipole moment between an electron and its exchange hole. The terms of higher order also incorporate the quadrupole and octopole moments. Like DFT+vdW, this scheme makes use of Hirshfeld partitioning. In this case, $f_{\text{damp}}(R_{AB})$ is the rational Becke-Johnson (BJ) damping function.

Steinmann and Corminboeuf[87] improve upon the XDM correction by switching to the damping function of Tang and Toennies.[88] This function introduces an additional dependence on the electron density. The coefficients are still obtained in a similar way, but make use of a computationally more efficient GGA-like formalism. The resulting approach is called dDsC.

While all aforementioned corrections are atom pair dependent, the DFT-NL method of Vydrov and van Voorhis[89] computes the dispersion interaction directly from the electron charge density. The non-local (NL) correlation energy is obtained via

$$E_c^{\text{NL}} = \frac{1}{2}\iint \rho(\mathbf{r})\psi(\mathbf{r},\mathbf{r}')\rho(\mathbf{r}')d\mathbf{r}d\mathbf{r}', \tag{4.3.22}$$

where $\psi(\mathbf{r}, \mathbf{r}')$ is an integration kernel accounting for the correct $1/R^6$ asymptotic behaviour.

The last type of dispersion correction discussed is the Minnesota class of functionals M0X by Zhao and Truhlar, where 0X stands for the year of development.[90] More than 30 parameters are employed in these type of functionals and extensive fitting to a training set composed of mainly non-covalent interactions allows for the recovery of a major fraction of the short-range correlation.

Exhaustive studies undertaken during the last years have shown the applicability of treating dispersion interactions in DFT with semiempirical corrections.[79,87,91–93] Most benchmark results for van-der-Waals compounds compare favourable to higher-level ab-initio methods. All of the introduced methods perform reasonably well and in some cases complement each other.

While DFT-D2 sometimes provides astonishingly accurate results due to error cancellation, the DFT-D3 method can be seen as an important improvement. Both DFT-D3 and DFT-NL yield similar results, with a slight tendency to overbinding in some cases. DFT-D3 has a better accuracy to computational cost ratio, as the evaluation of integral 4.3.22 introduces an overhead of approximately 50%. However, due to its direct dependency on the density, DFT-NL has an advantage when dealing with complicated electronic structures (e.g. metals), where the use of DFT-D3 is inherently limited. The DFT+vdW approximation is slightly less accurate and encounters serious problems when faced with many-body dispersion interactions. The performance of Becke's XDM method is similar, but both schemes are readily expandable to deal with anisotropic and polarizable systems. The DFT-dDsC correction of Steinmann and Corminboeuf is a serious improvement over XDM and excels at treating intramolecular dispersion interactions. The heavily parametrized functionals of Truhlar and Zhao provide viable results in most cases, especially for compounds similar to the training set, yet fail to correctly account for long-range dispersion effects.

4.3.6 *Advantages and Shortcomings of DFT*

The previous section dealt with the problem of DFT when faced with dispersion type interactions. While this is one of the best-known failures, other pathological cases due to limitations of the Kohn–Sham formalism or modern functionals exist.

Since the Kohn–Sham formalism essentially enforces a single determinant treatment, problems arise when dealing with multi-referential systems. This shortcoming may manifest itself in the inability to correctly predict the relative energies of triplet and singlet states.[94] While this issue can sometimes be amended by resorting to an unrestricted form of the Kohn–Sham procedure, great care has to be taken as this approach is also prone to errors.

Another drawback of DFT is "overdelocalization", an effect rooted in the inaccurate treatment of corrections to classical self-interaction energies. This leads to the tendency of overstabilizing systems with high delocalization effects and can result in erroneous predictions of some transition state geometries.[95–98]

The use of the electron density instead of a wavefunction approach can also be detrimental in some cases. Examples are dynamics, which depend on matrix elements between different wavefunctions. The lack of phases also leads to major issues when multistate resonance and interference effects are encountered. Additionally there is no concise way to derive a formalism in DFT as how excited states are to be described. A possible approach is to use time-dependant electron densities, as in TD-DFT.[99] Unfortunately the theoretical foundations of TD-DFT are not as methodologically sound as classical Kohn–Sham formalism.[100] Moreover, while the wavefunction formalism provides a wide range of well defined operators, only a few generic property functionals are known for DFT. This fact can be considered a major shortcoming as it enforces an approximate treatment of the exchange-correlation functional.

The lack of an exact exchange-correlation term poses a fundamental problem in DFT as – despite sophisticated schemes to emulate E_{xc} exactly – it remains an approximate method in the end. Furthermore, there is no protocol to systematically improve the quality of the approximation as is the case for *ab initio* methods. This deficit has given rise to several different models, which in turn lead to the creation of a variety of functionals. Since those often differ dramatically in their applicability and accuracy for different chemical systems, DFT should not be treated as a black-box method. Fortunately, a lot of benchmark studies are available, alleviating the choice of a suitable functional for the problem at hand.

Despite all these shortcomings, DFT has been established as an important method in electronic structure theory. It often yields results comparable to higher-level *ab initio* methods, but at only a fraction of the computational cost. The $O(N^3)$ scaling of DFT enables the treatment of much larger systems reasonable times. This efficiency is especially valuable for biochemical problems, as the typical system size encountered is often inhibitive to a treatment at the *ab initio* level. An additional speed-up is gained when the RI approximation is employed, making DFT feasible for the use in high-throughput screening. Recently, linear scaling implementations for computations on systems of biological scale have been developed.[101] It is also worth mentioning, that DFT is not restrained to the use of Gaussian-type orbitals and may employ Slater-type orbitals[102] or plane waves instead.[103] This liberty in the choice of basis functions can sometimes prove advantageous from a computational perspective. DFT also exhibits a faster convergence in regard to the size of the basis set used, often reaching maximum accuracy with basis sets several times smaller than required for *ab initio* methods.[104]

To summarize, DFT is a valuable tool for computational chemistry, but care should be taken to avoid the aforementioned pitfalls.

4.4 MOLECULAR PROPERTIES

DFT is a method to compute one point of the PES at a fixed geometry. Yet, most of the time the interest lies in special points of the PES, the geometries of local minima and transition states. In order to locate these extrema, several algorithms have been developed facilitating

geometry optimisation. Of similar importance are spectral properties, such as vibrational frequencies, as well as thermochemical properties, which can also be derived from a molecular geometry.

4.4.1 *Equilibrium Structures and Transition States*

As real molecules oscillate around their equilibrium structures, locating minimum energy geometries is crucial for the description of ground state and experimental properties. Since these structures are characterized by minima on the PES, geometry optimisation is intrinsically a mathematical problem of finding the next local minimum. Detailed knowledge of the PES would make this an easy task, but the computational cost renders an extensive computation for even small molecules impossible. Instead local properties of the PES are utilized. For an overview of the methods introduced here, see reference 105.

Gradient-based methods essentially follow the negative gradient in the direction of greatest downward slope in energy and thus are able to locate the nearest local minimum. These methods are usually referred to as steepest-descent methods. A major problem of optimisation algorithms based solely on the gradient is their slow convergence.

More sophisticated methods, such as the Newton-Raphson method, also consider the PES curvature in the search for local minima. By expanding the PES in a Taylor-series, truncating after the quadratic term and applying the condition that all first derivatives in a stationary point be zero, one arrives at the equation

$$\mathbf{R}_{k+1} = \mathbf{R}_k - \mathbf{H}_k^{-1}\nabla_k, \tag{4.4.1}$$

where ∇_k is the gradient of the nuclear coordinates at point k and $\mathbf{H}_k^{-1}$ is the inverse of the Hessian matrix of second derivatives at the same point. Unfortunately, the computation of the Hessian and its inversion are computationally expensive, so it is usually updated in an approximate way, using data obtained in previous points, instead of being recomputed every step. These methods are then called *quasi*-Newton methods.

Other critical points of interest on the PES are transition states, which connect two minimum energy structures and thus give important information on properties of the associated reaction. Transition states can be described as saddle points, where the gradient of all coordinates but the reaction coordinate is zero. Because of this property it is often difficult to locate the correct transition state connecting two molecules and several different methods exist to this end. Synchronous transit methods try to find the transition state by evaluating intermediate structures along a shortest path connecting two minima. Eigenvalue following algorithms maximise the energy in respect to one particular coordinate, while minimizing it in respect to all others. If the specified coordinate represents the normal mode describing the reaction, the correct transition state was found.

4.4.2 *Normal Mode Analysis*

Due to the quantum mechanical nature of the nuclei, real molecules are never found frozen at their respective equilibrium geometry, but

in constant motion, as dictated by the uncertainty principle. Normal mode analysis provides a method to compute these molecular vibrations and in turn makes it possible to obtain infrared spectra, derive thermochemical properties and identify transition states and minimum energy geometries. A detailed description of the associated formalism is given in reference 105.

Within the Born–Oppenheimer approximation nuclei can be thought of as moving in the potential of the PES. At a local minimum, this potential function can be expanded in a Taylor series. Truncation after the third term of the expansion leads to the harmonic oscillator approximation

$$\begin{aligned} V_{\text{pot}}(\mathbf{R}) =& V_{\text{pot}}(\mathbf{R}_{\text{eq}}) + \sum_{A=1}^{3N} (R_A - R_{A,\text{eq}}) \left(\frac{\partial V_{\text{pot}}}{\partial R_A} \right)_{\mathbf{R}=\mathbf{R}_{\text{eq}}} \\ &+ \frac{1}{2!} \sum_{A=1}^{3N} \sum_{B=1}^{3N} (R_A - R_{A,\text{eq}})(R_B - R_{B,\text{eq}}) \left(\frac{\partial^2 V_{\text{pot}}}{\partial R_A R_B} \right)_{\mathbf{R}=\mathbf{R}_{\text{eq}}} . \end{aligned} \tag{4.4.2}$$

The first term is a constant, the second term vanishes at an energy minimum, leaving only the third term depending on the Hessian matrix. Because of the difficulties arising when dealing with Cartesian coordinates, it is advantageous to introduce mass-weighted spatial coordinates $\theta_A = \sqrt{M_A}(R_A - R_{A,\text{eq}})$. Equation 4.4.2 can then be expressed as

$$V_{\text{pot}} = \frac{1}{2} \sum_{A=1}^{3N} \sum_{B=1}^{3N} H_{AB} \theta_A \theta_B. \tag{4.4.3}$$

By diagonalizing the the Hessian matrix in equation 4.4.3, the function depending on $3N$ nuclear coordinates can essentially be decoupled into $3N$ eigenvector equations depending on just one coordinate

$$\Theta_A = \sum_{B=1}^{3N} K_{AB} \theta_A. \tag{4.4.4}$$

In other words, equation 4.4.3 can be transformed to

$$V_{\text{pot}} = \frac{1}{2} \sum_{A=1}^{3N} \mathfrak{H}_{AA} \Theta_A^2, \tag{4.4.5}$$

where $\mathfrak{H}$ is the diagonalized Hessian matrix. The frequencies are then given by

$$\omega_A = \sqrt{\mathfrak{H}_{AA}}, \tag{4.4.6}$$

and the vibrational energy can be computed according to

$$E = \sum_{A=1}^{3N} \hbar \omega_A \left(\nu_A + \frac{1}{2} \right), \tag{4.4.7}$$

where ν_A is the vibrational quantum number. Using the transformation matrix $\mathbf{K}$ obtained in equation 4.4.4, the normal mode coordinates $\boldsymbol{\Theta}$ can be transformed back into their Cartesian equivalents $\mathbf{R}$, allowing for a Cartesian description and visualisation of the vibrations.

Certain care has to be taken when working with properties derived from normal mode analysis. The use of the harmonic approximation introduces intrinsic errors, like the inability to account for bond dissociation and false predictions when working with geometries displaced from the minimum energy structure. This effect is further amplified by the fact that the gradient in equation 4.4.2 only vanishes at stationary points of the PES.

4.4.3 *Thermodynamic Properties*

Experiments usually do not deal with individual molecules, but with large ensembles governed by the laws of thermodynamics. Thus great interest lies in the derivation of ensemble thermochemical properties, like enthalpy H, entropy S or free energy G, from the previously computed single molecule potential energies. Entropic contributions to the energy are of special importance in rearrangements involving dissociations and associations of substrates, as is the case for the metathesis reaction, or reactions exhibiting a prearrangement of reactants, a common feature in enzymatic transformations. The free energy is therefore a well-suited descriptor for characterizing this kind of chemical reactions. For a more detailed discussion of the topic and derivation of the formalism, see reference 106.

As discussed beforehand, the nuclei are subject to quantum mechanical oscillations, where a certain energy is associated with each vibrational mode. Using the relation 4.4.7 and summing the energies of all molecular vibrations gives the the zero-point vibrational energy (ZPVE). This in turn leads to an expression for the internal energy U_0 of a molecule at absolute zero, which can be written as

$$U_0 = E_{\mathrm{el}} + \sum_{\kappa=1}^{\mathrm{modes}} \frac{1}{2} h\omega_\kappa, \tag{4.4.8}$$

where E_{el} is the electronic energy. Based on this formula, it is then possible to derive further properties by applying the rules of statistical thermodynamics.

Assuming a canonical ensemble, the system is characterized by its partition function

$$Q(N_{\mathrm{part}}, V, T) = \sum_{\varphi=1} e^{-E_\varphi(N_{\mathrm{part}}, V)/k_{\mathrm{B}}T}, \tag{4.4.9}$$

where φ indicates all possible system states with the corresponding energy E_φ, k_{B} is the Boltzmann constant and T the temperature N_{part} is the total number of particles, which is in general chosen to be Avogadro's constant N_{A} and V is the volume occupied by the ensemble. By introducing the assumption of the ensemble being an ideal gas, it is possible to reduce the problem to finding the molecular partition function, which can be separated into electronic, translational, rotational and vibrational terms, leading to the expression

$$q(V, T) = q_{\mathrm{el}}(T) q_{\mathrm{trans}}(V, T) q_{\mathrm{rot}}(T) q_{\mathrm{vib}}(T). \tag{4.4.10}$$

Thus the different contributions to the molecular partition function and in turn to the internal energy, as well as to the entropy, can be computed individually.

Electronic Contributions

Provided the energetic separation of the ground state and the next excited state is high enough and introducing the convention of defining E_{elec} as the relative zero of energy in equation 4.4.8, the electronic partition function can be greatly simplified. The electronic component of U is reduced to

$$U_{\text{el}} = 0 \tag{4.4.11}$$

and the electronic contribution to the entropy computes as

$$S_{\text{el}} = R\ln(2\mathfrak{s}+1), \tag{4.4.12}$$

where R is the universal gas constant and $\mathfrak{s}$ is the spin multiplicity.

Translational Contributions

The translational partition function is obtained by treating the molecule as a particle in a three dimensional box. The translational contributions to U and S are then given by

$$U_{\text{trans}} = \frac{3}{2}RT \tag{4.4.13}$$

and

$$S_{\text{trans}} = R\left\{\ln\left[\left(\frac{2\pi\mathcal{M}k_{\text{B}}T}{h}\right)^{\frac{3}{2}}\frac{RT}{P_0}\right]+\frac{3}{2}\right\}, \tag{4.4.14}$$

where $\mathcal{M}$ is the molecular mass and the pressure is usually set to be $P_0 = 1$ atm, in order to be able to compare different thermodynamic results.

Rotational Contributions

Regarding the rotational contributions, a distinction must be made for linear and non-linear molecules. In the linear case, the rotational components of the inner energy and entropy can be formulated as

$$U_{\text{rot}}^{\text{linear}} = RT \tag{4.4.15}$$

and

$$S_{\text{rot}}^{\text{linear}} = R\left[\ln\left(\frac{8\pi^2 Ik_{\text{B}}T}{\sigma h^2}\right)+1\right], \tag{4.4.16}$$

where I is the molecular moment of inertia and σ a rotational symmetry number.

For a non-linear molecule a quantum mechanical approximation to the classical rigid-rotor problem has to be applied and gives

$$U_{\text{rot}} = \frac{3}{2}RT \tag{4.4.17}$$

for U and

$$S_{\text{rot}} = R\left\{\ln\left[\frac{\sqrt{\pi I_A I_B I_C}}{\sigma}\left(\frac{8\pi^2 k_{\text{B}}T}{h^2}\right)^{\frac{3}{2}}\right]+\frac{3}{2}\right\} \tag{4.4.18}$$

for the entropy, with I_A, I_B and I_C being the principal moments of inertia.

Vibrational Contributions

The vibrational partition function can be expressed as a sum of the individual modes. Approximating the modes as quantum mechanical harmonic oscillators allows further simplifications. Provided the vibrational frequencies are known, the vibrational components of U and S can be computed according to the equations

$$U_{\text{vib}} = R \sum_{\kappa=1}^{\text{modes}} \frac{h\omega_\kappa}{k_B(e^{h\omega_\kappa/k_\text{B}T} - 1)} \tag{4.4.19}$$

and

$$S_{\text{vib}} = R \sum_{\kappa=1}^{\text{modes}} \left[\frac{h\omega_\kappa}{k_\text{B}T(e^{h\omega_\kappa/k_\text{B}T} - 1)} - \ln\left(1 - e^{-h\omega_\kappa/k_\text{B}T}\right) \right]. \tag{4.4.20}$$

Free Energy

Having derived the contributions to U and S in such a manner, it is possible to calculate the total inner energy according to

$$U = U_0 + U_{\text{el}} + U_{\text{trans}} + U_{\text{rot}} + U_{\text{vib}}. \tag{4.4.21}$$

The enthalpy may be written as

$$H = U + PV. \tag{4.4.22}$$

The free energy of the molecule can then be obtained using relation

$$G = H - TS, \tag{4.4.23}$$

where S is simply the sum of the different contributions to the entropy.

Caveats

Unfortunately using this formalism to characterize the thermochemical properties exhibits pathological behaviour when confronted with vibrational modes of low frequency. While this effect is minor with regards to the internal energy and in turn the enthalpy, as every individual frequency contributes only a factor of RT, it greatly affects the molecular entropy and thus the free energy of the system. The reason for this deficiency can be found by examining the second term of equation 4.4.20, which approaches infinity as the corresponding frequency reaches zero. This trend can result in very large errors to the entropy for even minor errors in low frequencies. Since the harmonic oscillator approximation is typically poor for these low frequency modes, great care has to be taken when dealing with entropic properties of a molecule exhibiting low frequency vibrations. Typical examples are torsions around single bonds with only small barriers, commonly referred to as free or hindered rotors. For a detailed discussion see reference 105.

4.5 GENETIC ALGORITHMS

While classical deterministic optimisation algorithms are an important tool for finding the extrema of functions, they quickly become intractable when faced with search spaces of high dimensionality. Moreover there is no guarantee to locate the global optimum if additional

local minima are present, as the classic optimisation procedures tend to converge towards the nearest extremal point. In the last decades metaheuristic search algorithms have been established as valuable tools for dealing with this type of problems. One prominent class of stochastic optimisation methods are genetic algorithms, modelled after the mechanisms of Darwinian evolution.[30]

In genetic algorithms, candidate solutions to a problem are represented by a genome. This genome consists of individual genes, which describe certain aspects of the solution. Since this kind of search method is inspired by natural selection, a measure for the fitness of a potential solution has to be provided in the form of the so-called fitness function. This function maps a real value representing the quantity to be optimized, usually referred to as fitness, to every candidate's genome. Genetic algorithms then operate on a set of potential solutions, the population, subjecting their respective genomes to genetic operations such as recombination, mutation and selection with the aim of continuous improvement towards a global optimum. The candidate solutions are also called individuals in the context of population.

A typical example for a genetic algorithm consists of the following steps (Figure 4.1):

INITIALIZATION: An initial set of random solutions spanning the relevant search space is created.

EVALUATION: The fitness of individual genomes is evaluated according to the fitness function.

SELECTION: Based on the relative fitness, candidate solutions are chosen for genetic operations and extinction events, introducing the notion of survival of the fittest into the genetic algorithm.

RECOMBINATION: The genomes of two or more promising solutions are recombined in the hope of arriving at offspring solutions of higher fitness by a favourable mixing of beneficial traits.

MUTATION: The mutation operator modifies a single solution by manipulation of individual genes. This is equivalent to performing a local search near the original candidate solution.

REPLACEMENT: Certain individuals of the original population are replaced by the offspring solutions or entirely new genomes. Different schemes exist to this end.

Starting with a re-assessment of the fitness, the above steps are then repeated until a desired terminating criterium is met.

The greatest difficulty in implementing a genetic algorithm usually lies in the efficient balancing of exploration and exploitation. While global search due to the diversity of the population allows for an efficient sampling of the entire fitness landscape, it can hinder the convergence towards a common solution. Local optimisation on the other hand guarantees convergence, but is in danger of getting stuck in a local minimum. Because of this, several different schemes for the genetic operations used in a genetic algorithm have been developed.

Examples for the different genetic operators commonly used and a summary on the advantages and disadvantages of genetic algorithms will be given in the following sections.

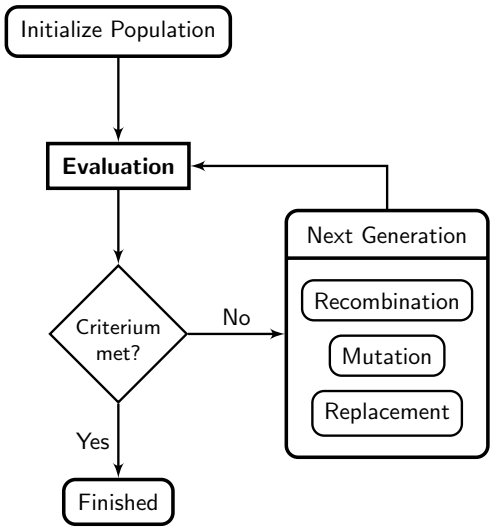


Figure 4.1: General operation scheme of a genetic algorithm.

4.5.1 *Selection*

The main requirement of selection operators is to introduce enough selection pressure to drive evolution of the system towards an optimum, while avoiding premature convergence. These operators can be grouped into two main classes.

As the name suggests, fitness proportionate selection draws direct information from the fitness of each individual when selecting potential candidates. Typical examples are roulette wheel selection[30,107] and stochastic universal sampling.[108]

Ordinal selection methods derive the selection criteria from the relative ordering of each candidate with respect to its fitness. A popular ordinal selection method is tournament selection[107], where genomes are chosen randomly to participate in tournaments of a predefined size. The fittest individual is then chosen as the winner of each tournament.

4.5.2 *Recombination*

As mentioned above, the recombination procedure is the main driving force in exploring the search space at a global level. Because of its importance a wide range of different approaches have emerged, many of which are tailored to specific problems.[107,109]

In general, two individuals are selected by one of the aforementioned procedures and will be subjected to a recombination event according to a recombination rate r_{rec}. A random number r_{rand} is then drawn from a uniform distribution. If $r_{\text{rand}} \leq r_{\text{rec}}$ recombination will take place, otherwise the children are chosen to be identical copies of the parents. Two basic recombination schemes exist.

In a n-point crossover, n crossover sites in the genome are selected at random. The genes in each candidate are then exchanged according to the chosen sites.

An alternative is uniform crossover[110], where each gene is chosen for recombination individually and the respective bits are switched according to a crossover rate r_{cx}.

4.5.3 *Mutation*

Mutation operators are important for local exploration of the fitness landscape, as well as the introduction of new genetic motifs. This provides a mechanism against population stagnation.

A common example for a mutation strategy is the bit-flip mutation. Here each bit constituting the genome of a selected individual is flipped to its counterpart with a probability r_{mut}, also called the mutation rate.

4.5.4 *Replacement*

The offspring genomes created during recombination and mutation have then to be reintroduced into the original population. It is generally assumed, that the newly generated individuals improve upon the solutions of the previous generation. Several methods exist for updating the population.

The delete-all scheme completely replaces the old population by the created offspring.

Steady state replacement is a deletion method and substitutes only a certain number k of old members by new genomes. It introduces flexibility by the choice of k and the selection procedure used to determine the individuals to be replaced.

4.5.5 *Advantages and Disadvantages of Genetic Algorithms*

The main feature of genetic algorithms is their ability to deal with optimisation problems with high search-space complexity and tailored objective functions. Since this class of algorithms does not require detailed *a priori* knowledge of the fitness landscape or the form of the fitness function, they possess great generality.[111–113] This trait is attested by their successful application in a wide range of different research fields. In addition, they offer means of locating the global optimum despite the presence of local extrema.

A drawback of genetic algorithms is their use of several different parameters, like the population size, the crossover rate, the mutation rate, etc. In most cases only rules of thumb exist for the choice of these parameters and the whole process is essentially a trial-and-error procedure. A suboptimal set of parameters can in turn severely restrict the utility of the algorithm employed. Nevertheless, the last years have seen the development of several subclasses of genetic algorithms alleviating parts of this problem by subjecting the parameters to Darwinian evolution too.[114–119]

4.6 GEOMETRIC HASHING

Initially developed for the recognition of geometric features in computer vision, geometric hashing is now used in several other areas, e.g. medicinal chemistry and molecular biology. This increase in popularity is due to the efficiency of the method and its low polynomial scaling. For a good review on geometric hashing and further information, see Ref. 120 and references within.

The goal of geometric hashing is the identification of a shape on the basis of its unique combination of individual geometric features. The geometric hashing process can essentially be separated into two phases.

In the preprocessing phase, a database of the shapes to be recognized is created. To this end, the geometric features (e.g. points, edges,...) are extracted (Figure 4.2).

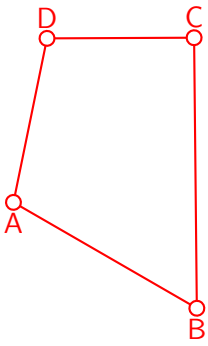


Figure 4.2: In the first step of the preprocessing phase, the features of a shape are extracted. In this two-dimensional case, the points A, B, C and D are sufficient.

A subset of these features is then used to define a new basis, as demonstrated in Figure 4.3.

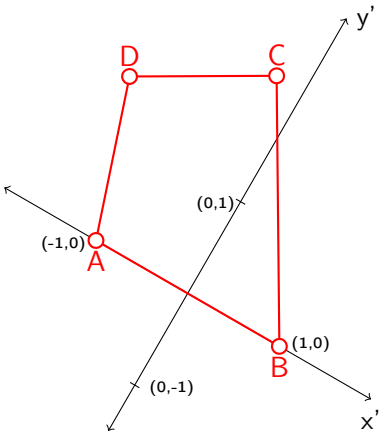


Figure 4.3: The two points A and B are used to establish a basis for a new reference coordinate system.

The coordinates of the remaining features, e.g. points, relative to the basis are computed. In order to make recognition robust to noisy data, the coordinates are usually not computed exactly but rather discretised on a grid (Figure 4.4). These discretised coordinates associated with a geometric feature, are called a "key". Based on these keys, a special two-column table is constructed, where every key is stored in the first column of a row and used to access the field in the second column of the same row, which is often referred to as an "entry". In this entry, the basis used to compute the feature and, if more shapes are present, an index for the shape, are stored. The discrete coordinates/keys constituting every basis are not stored in the table, as they are always

set to the same values (-1,0 and 0,1). Their entries would thus hold all the bases which is of little use if shapes are to be distinguished. This process of generating relative discretised coordinates and using them as a key to store information about the associated basis in the table is then repeated for every possible basis. If a key already exists, the new basis is simply appended to the data already present in the entry specified by the key (Table 4.1).

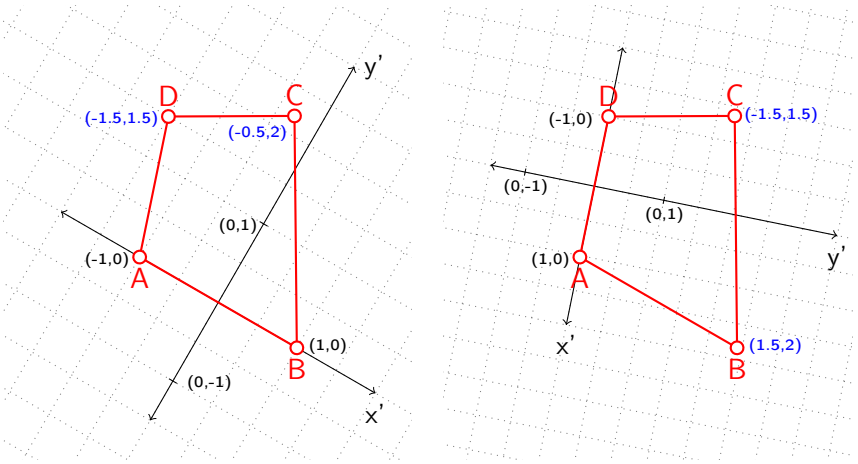


Figure 4.4: On the left, the coordinates of the remaining points C and D are computed relative to the coordinate system formed by A and B. The right side shows the same process for the basis DA and the points B and C. In order to reduce errors due to noise in the input data, the coordinates are discretised, in both cases using a grid of spacing 0.5. The coordinates are then rounded down to the values of the nearest grid points.

This data-storage process is termed hashing and the resulting table is referred to as a hash table. Since the discretisation of the coordinates leads to coordinates within a certain range being grouped together, the table entries are also called hash bins.

Table 4.1: A hash-table containing a selection of discretised relative coordinates/keys in the first column and the associated bases, as computed for the sample shape, in the respective entries of the second column. The introduction of discretised coordinates leads to multiple bases being assigned to a single hash-bin.

Coordinates/Keys	Basis
(-1.5,1.5)	AB, DA
(-0.5,2.0)	AB
(1.0,1.0)	BC
(-0.5,1.0)	BC
(-1.0,4.0)	CD
(1.5,2.0)	CD, DA
⋮	⋮

To identify an unknown shape in the recognition phase, its geometric features are extracted. After choosing a basis, the relative

coordinates of the remaining points are calculated and discretised as before (Figure 4.5).

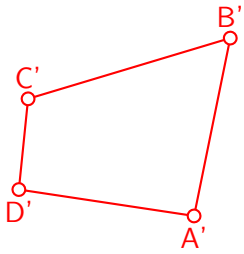


Figure 4.5: An unknown shape. In order to determine the orientation which exhibits the greatest similarity to the previous shape shown in Figure 4.2, its geometric features are extracted. In the same manner as above, the discretised relative coordinates are calculated with respect to different bases. For this example only two bases were used: A'B' leading to the coordinates/keys (-1.5,1.5) and (-0.5,2.0), as well as B'C' with the coordinates/keys (1.0,0.5) and (-0.5,1.5).

For every basis used in the computation of the discretised relative coordinates, the coordinates/keys are used to access the corresponding hash-table entries. A counter is then incremented for every base present in the bin. As an example, the process was carried out for the coordinates/keys of the shape in Figure 4.5 relative to the bases A'B' and B'C'. For the basis A'B', the hash table entry indexed by the key (-1.5,1.5) holds the bases AB and DA of the original shape, whose counters are then increased by one each. The same process is repeated for the remaining point and the coordinates/keys of the basis B'C'. Plotting the values of the counters for every basis of the original shape for each basis A'B' and B'C' of the new shape, two histograms are obtained (Figure 4.6).

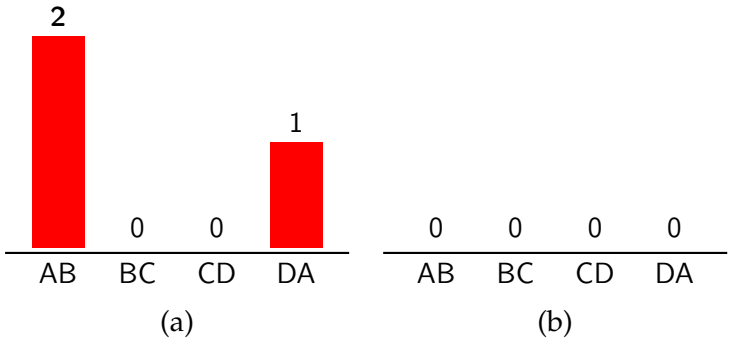


Figure 4.6: Sample histogram obtained for the hash-table 4.1 and the relative coordinates of the bases A'B' (a) and B'C' (b), extracted from the unknown shape.

The bases whose counters surpass a certain threshold of votes constitute the potential matches. These hits are then subjected to a more fine-grained recognition procedure. In the case of the example used in this chapter, the best match between orientations is identified for the orientations relative to basis AB in the original shape and basis A'B' in the unknown shape (Figure 4.7). This is the combination of

bases with the highest count of two, all other combinations possess counts of zero or one.

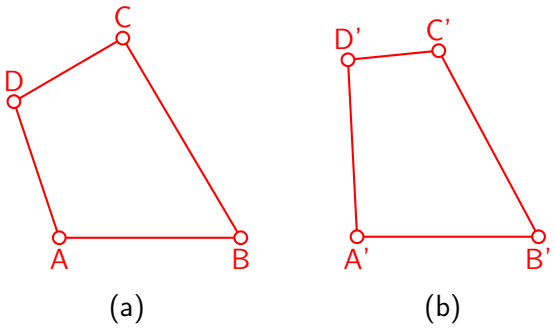


Figure 4.7: The histogram in Figure 4.6 shows, that the geometric hashing procedure correctly identifies the orientation of the new shape (b) along the basis A'B' to be the most similar to an orientation of the original shape (a) relative to the basis AB.

This protocol can be easily adopted for different shapes, by substituting the bases by their respective basis-shape combinations in order to discriminate between shapes, like e.g. the one given in Figure 4.2 and a circle with the same points A,B,C and D, on basis of their geometric features.

The strength of geometric hashing lies in its ability to recognize shapes even in cases where they have undergone transformations, like translation or rotation, or when only partial information is available.

5

COMPUTATIONAL METHODS

In this chapter, the different methods employed during the course of this work will be specified. Due to the nature of the research, it is feasible to subdivide this chapter into three parts, with the following sections dealing in turn with the quantum chemistry methods, the genetic algorithm and the enzyme design utilities.

5.1 QUANTUM CHEMISTRY METHODS

As the size of the studied systems renders wavefunction-based approaches impractical and an efficient and robust method is required for high throughput screening in the framework of a genetic algorithm, DFT was used for electronic structure computations. This choice is supported by the good agreement of computational and experimental observations reported by several DFT studies on Grubbs-type ruthenium complexes and transition metal complexes in general.[49,50] Moreover, DFT profits from a significant speed-up in terms of computation time when the RI-approximation is employed. The nature of the complexes studied also permits a closed-shell treatment, since they are known to adopt low-spin configurations[121], allowing several of the limitations of DFT to be bypassed (see end of section 4.3). As all reactions are thought to take place inside of peptide cavities, which are commonly accepted to possess relatively low dielectric constants ($\varepsilon \sim$ 3-6) compared to water ($\varepsilon \sim 80$)[122], it is furthermore admissible to neglect solvent effects.[123]

All computations were either performed with GAUSSIAN09[124] or TURBOMOLE-6.4.[125]

5.1.1 *Choice of Functional and Basis Set*

The utilized functionals were B3LYP[72] as implemented in GAUSSIAN[124] and BP86[62–65,67] as provided by the TURBOMOLE suite of programs.[125] The B3LYP hybrid functional was chosen for its general applicability for a wide range of different chemical species, whereas the GGA BP86 provides excellent computational efficiency since it can make full use of TURBOMOLE's RI-formalism and avoids the costly evaluation of Hartree–Fock exchange. Both functionals were augmented with empirical dispersion corrections to account for noncovalent attractive interactions present in this class of complexes, which DFT fails to predict correctly.[121,126] The Grimme dispersion corrections D2 and D3 were used for B3LYP and BP86, respectively.

In case of B3LYP the 6-31G* Pople basis set[127,128] was used on all non-metal atoms, as it usually gives the best results due to its use during the parametrisation of the functional. BP86 computations were carried out with the def2-SV(P) basis set of Ahlrichs and coworkers[129] in the form implemented in TURBOMOLE. In both cases, the Stuttgart relativistic effective core potential (ECP) was used for the central ruthenium atom

of the complexes.[130] If not stated otherwise explicitly, the combinations of B3LYP-D2/6-31G*/ECP and BP86-D3/def2-SV(P)/ECP+MARI-J will be simply referred to as B3LYP and BP86 henceforth.

5.1.2 *Geometry Optimisation and Transition State Location*

Geometry optimisations were carried out using both functionals in tandem. In order to speed up the process, the starting geometry was preoptimized in TURBOMOLE with BP86 and the multipole-accelerated RI-approximation (MARI-J).[131] The resulting structure was then subjected to a final optimisation in GAUSSIAN with B3LYP. Computational results obtained during a later stage of the thesis demonstrated, that BP86 alone is sufficiently accurate for the problem at hand (Section 6.4.2). In both cases, the standard optimisation algorithms as well as the default grids and convergence criteria of the programs were used.

Transition states connecting neighbouring geometries were located with the QST2 protocol[132] of GAUSSIAN, which employs the synchronous Transit-Guided Quasi-Newton method developed by Schlegel and coworkers. Alternatively, the eigenvalue-following algorithms of TURBOMOLE and GAUSSIAN were used to locate the transition state with BP86 and further refine the results on B3LYP-level.

Both ground state structures and transition states were characterized by frequency analyses in GAUSSIAN or TURBOMOLE respectively, employing the standard quantum-harmonic oscillator approximation.

5.1.3 *Thermochemical Properties*

Thermochemical properties like the free energy of the respective systems were obtained with the standard formalism introduced in section 4.4.3 as implemented in both programs. All formulae were evaluated at temperatures of 298.15 K and pressures of 1 atm.

5.2 IMPLEMENTATION OF THE GENETIC ALGORITHM

To arrive at active enzymes, theozymes with favourable reaction profiles are needed. Applying Pauling's postulate about the complementarity of the active site and the transition state representing the reaction[29], the problem can be reduced to finding the optimal placement and combination of amino acid residues, which lower the reaction barrier by stabilizing the theozyme. This problem is of combinatorial nature and thus lends itself to a stochastic optimisation approach. To this end, a genetic algorithm was developed over the course of the thesis. It was written in PYTHON[133], a programming language providing many useful utilities for scientific computing. The initial algorithm employed a classical generational approach, but insights obtained during the course of the work also lead to the creation of a steady-state variant. The details of implementation will be discussed in the following paragraphs and fundamental differences between the versions will be highlighted.

5.2.1 *General Algorithm Structure*

The generational genetic algorithm on the one hand follows the standard model depicted in Figure 4.1 and genetic operations are only initiated, when the fitness evaluation of every member constituting the current population has finished.

The steady-state implementation on the other hand constantly feeds completely evaluated genomes into the population and genetic operations occur every time a certain threshold is reached. This convention leads to an initial period of saturation, where the algorithm only operates on a subset of the population until the maximal population size is reached. A schematic of the steady-state genetic algorithm is shown in Figure 5.1, the source code can be found in Appendix A.2.

Advantage of the latter procedure is the ability to deal with the greatly varying calculation times encountered during the fitness evaluations, which can cause significant delays in the generational approach and lead to a suboptimal use of the available computational resources.

Both algorithms continue to operate until a certain stopping criterion is reached, in this special case the number of fitness evaluations n_{eval}.

5.2.2 *Population Representation*

Central to every evolutionary algorithm is the genetic representation of the individuals constituting the population. In this case a real coded representation was chosen, making use of real numbers instead of the bitstrings of zeroes and ones traditionally employed for genome encoding. Since the theozyme consists of a fixed central template, the metal-substrate complex, and only the arrangement and identity of the catalytic amino acid residues vary, the genome of an individual was defined to consist of a list of the residues present, as well as the associated atom types and Cartesian coordinates.

In order to increase efficiency and still be able to capture essential interactions, the residue templates were truncated at the C_β atoms of the amino acids. Theoretical studies show that this method is able to describe enzyme active sites with reasonable accuracy.[123] All 21 eukaryotic proteinogenic amino acids (Figure 5.2), with the exception of glycine, alanine and proline, were present in the residue library, as well as the different protonated states of histidine and cysteine.[134]

5.2.3 *Initialization*

For every individual in the initial population of size p_{max}, a certain number of residues were chosen according to a uniform random distribution. Due to the different structure of the crossover and mutation operators, a fixed value (g_{size}) was employed in the steady state algorithm, while the amount of residues per genome was allowed to vary within certain limits (g_{min}, g_{max}) in the generational approach. Spatial placement was restricted uniformly to the regions around the central template and collisions were avoided. Rotational freedom was sampled using the Marsaglia scheme[135] under the constraint that the positions leading to the backbone of the protein face away from the central template.

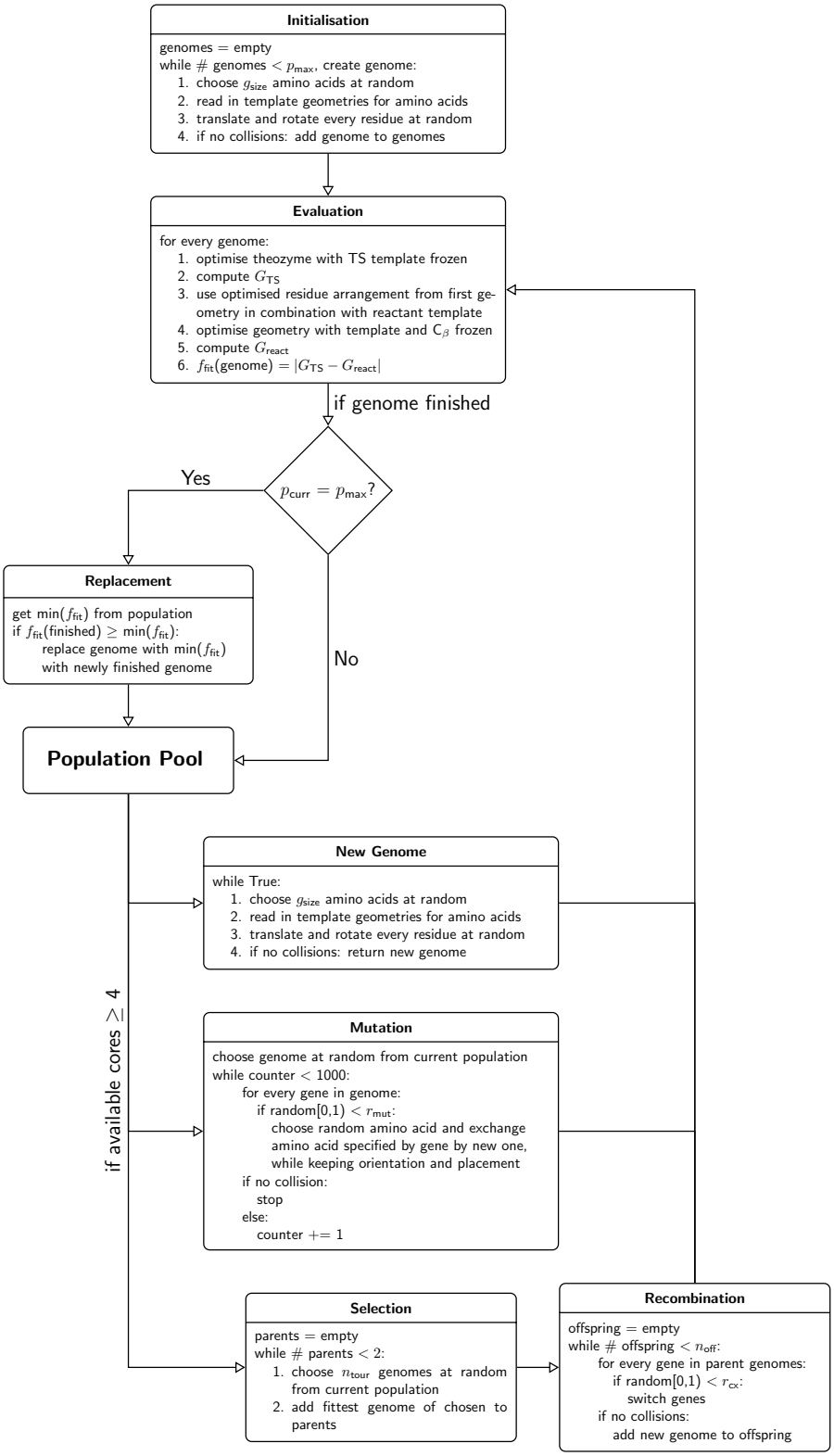


Figure 5.1: Workflow of the steady-state genetic algorithm used to search for theozymes. In this case genetic operations were carried out every time the evaluation of a genome finished and at least four of the processors designated for evaluation were available. This criterium was chosen in order to keep the number of evaluations approximately constant, as the genetic operations generate four new genomes in total (one new genome, one mutant and two offspring genomes).

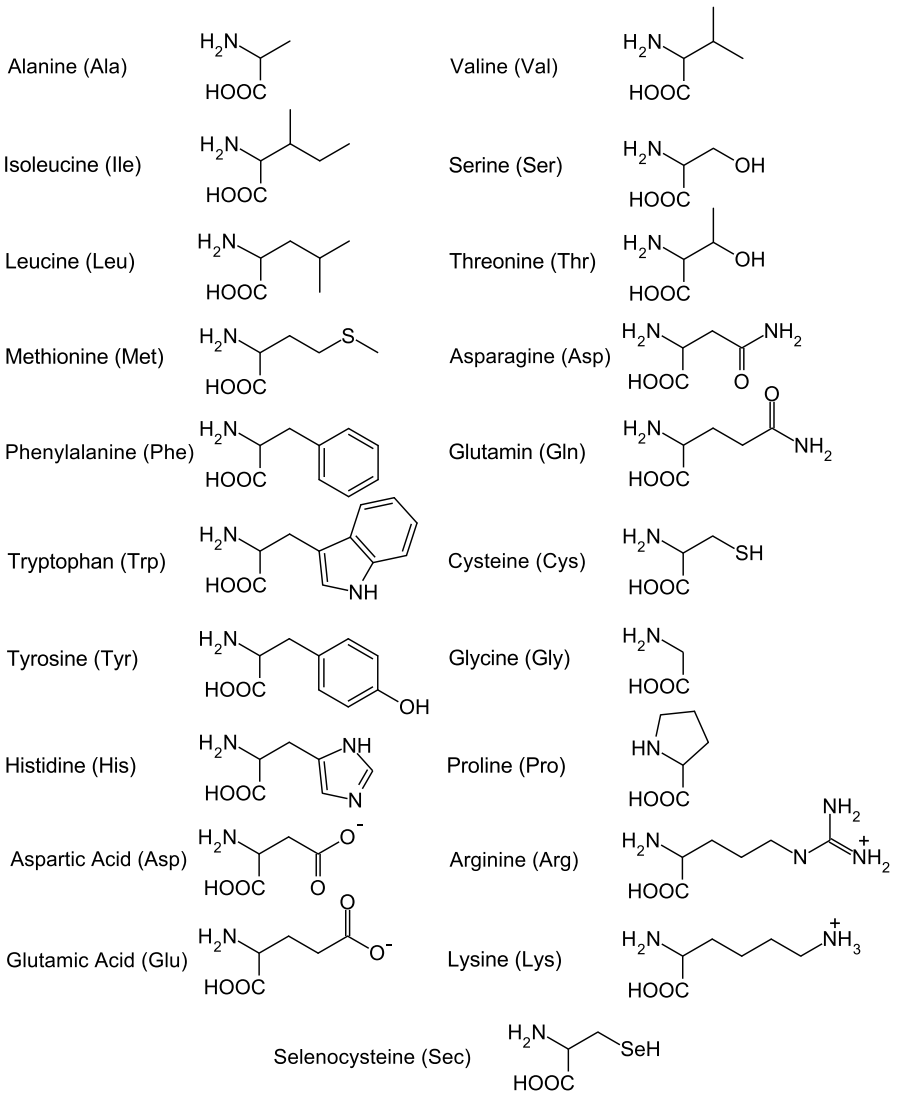


Figure 5.2: The 21 proteinogenic amino acids found in eukaryotic organisms. The standard three letter abbreviations are given in parentheses.

5.2.4 *Genetic Operators*

As stated in section 4.5, the genetic operators serve to globally explore or locally exploit the search space. These two modes of operation have to be balanced in order to avoid premature convergence to suboptimal solutions. It is therefore advantageous to tailor some operators specifically to the problem at hand.

Selection

In both versions of the genetic algorithm, tournament selection was chosen because of its robustness and the possibility to adjust selection pressure by varying the tournament size n_{tour}. This parameter was set to be constant in the generational variant, whereas in the steady state version it depended on the current population size p_{curr}, in order to account for the initial saturation period. It was modelled to vary linearly between a maximum and minimum value depending on the individuals in the current population pool, according to the relation:

$$n_{\text{tour}} = \text{rint}(k_t p_{\text{curr}} + d_t), \tag{5.2.1}$$

where rint rounds to the next integer. The values for $k_t = 0.14$ and $d_t = 1.4$ were obtained by linearly interpolating between a tournament size $n_{tour} = 2$ for $p_{curr} = 4$ and a tournament size of $n_{tour} = 10$ for a population of 60 individuals.

Recombination

The crossover between two parent genomes was facilitated by a modified uniform crossover operator in both cases, where one residue with all associated atom types and coordinates was treated as a single bit. For every residue present in both genomes a check was made against a predetermined crossover rate r_{cx} and if the check succeeded, the two "bits" were exchanged. The process was repeated in the case collisions were present in the offspring geometries until the desired number of offspring genomes n_{off} was created.

Mutation

Mutation was carried out by replacing one residue of a selected genome by a random amino acid, while keeping the overall position and orientation. To conserve orientation, the unit quaternions representing the rotation of the residue relative to the template are computed by a least-squares fitting procedure.[136,137] Using these quaternions, a rotation matrix is then generated and applied to the coordinates of the mutant amino acid. The affected residues are chosen by checking a uniform random number against a predefined mutation rate r_{mut}. In the generational implementation every individual was subjected to this procedure, the steady state variant on the other hand selected one genome of the population at random. The mutation rate was deterministic in both cases and modelled after the relation

$$r_{mut} = \frac{1}{(p_{curr} + 1)}. \tag{5.2.2}$$

In order to ensure the integrity of the mutants, collision checks were carried out and the process was repeated until a valid geometry was obtained or an internal threshold reached.

Replacement

Both algorithms made use of an extinctive replacement scheme, substituting the individuals with the worst fitness values by the newly generated offspring and mutants every generation. In order to ensure genetic diversity, a number of new genomes n_{new} is also introduced during this step.

5.2.5 *Fitness Evaluation*

The fitness of every individual is calculated as an absolute difference between the free energies of a stationary state and the subsequent transition state. Thus, this fitness essentially corresponds to an activation barrier. The free energy G is chosen as the energy descriptor, as

metathesis reactions are typically subject to substantial entropic contributions.[52,53] The two single point evaluations of G are performed by external programs.

For the first computation the transition state is used as the central template. The catalytic residues are placed as specified in the genome. The geometry is then optimised with only the template constrained and the free energy is computed afterwards.

For the second point, the optimised amino acid placements of the first geometry are kept, but the central species is exchanged for a template resembling the reactant. Keeping the C_β atoms of the residues and the central complex fixed, the geometry is once again optimised. Based on the final structure, G is computed.

Selection pressure was thus applied by means of the genetic algorithm towards a minimisation of every individuals fitness, *i.e.* to lower the required energy of activation. The absolute value was chosen in order to avoid overstabilisation of the metallacycle intermediate and provide a relatively smooth reaction profile.

One of the major differences between the generational and steady state version of the algorithm, are the quantum chemical methods and programs employed during fitness assessment. The generational approach makes use of the combination of GAUSSIAN and TURBOMOLE described in 5.1.2 for geometry optimisation and computes the free energy on the B3LYP level using GAUSSIAN.

Requirement for a speed-up of the process and results validating the use of BP86 for geometries (see 6.4.2), energies and vibrational frequencies[47,138–141] for transition metal complexes, lead to the implementation of an alternative fitness evaluation for the steady-state algorithm. In this case, all calculations and optimisations are carried out on the BP86 level using TURBOMOLE, making full use of the RI procedure and the programs efficiency.

The communication between the genetic algorithm and the external programs GAUSSIAN and TURBOMOLE was fashioned after a master-slave model, where a central process (master) controls several subprocesses (slaves) performing individual tasks. In order to parallelize the procedure and to make optimal use of the local computational infrastructure, one core was used for the fitness evaluation of every separate genome.

5.3 ENZYME DESIGN

After a theozyme was obtained, protein scaffolds were screened for sites capable of accommodating the designed geometry. Finally, the sequence of a potential match was redesigned to optimize catalytic interactions. Both tasks were carried out with the ROSETTA3 suite of programs developed by Baker and coworkers.[23]

5.3.1 *Matching*

ROSETTA employs a matching algorithm based on geometric hashing to search for scaffold candidates. In order to perform a search, a pdb-file of the scaffold, a pos-file listing the accessible backbone positions, a parameter file of the metal-substrate complex and a cst-file specifying

the geometric constraints of the theozyme are needed. These topics will be discussed in the following sections.

Scaffold Library

The scaffolds were obtained from the RCSB protein database[142] according to several criteria: X-ray resolution better than 2 Å, reported expression in *E. coli* to alleviate synthesis, structural diversity and no identical sequences. To ensure the use of different protein topologies, representatives of every major CATH[143]-architecture domain (**C**lass **A**rchitecture **T**ogpology **H**omologous superfamily) were chosen (Table 5.1). The criteria specified above resulted in a library of 5458 scaffolds.

Table 5.1: Major CATH-architectures present in the scaffold library.

CATH-classifier	Description
1.10	Orthogonal Bundle
2.40	Beta Barrel
2.60	Sandwich
3.10	Roll
3.20	Alpha-Beta Barrel
3.30	2-layer Sandwich
3.40	3-layer Sandwich
3.90	Alpha-Beta Complex

Accessible Backbone Positions

Accessible backbone positions of the protein scaffolds were located with the Rosetta-holes algorithm.[144] Only cavities with a volume larger than 30 $Å^3$ were considered. To provide sufficient shielding of potential active sites from environmental influences, while still allowing substrate coordination, protein pockets accessible by probes with a radius greater than 5 Å were pruned by the algorithm.

Geometric Constraint File

The constraint file essentially is a reduced representation of the theozyme geometry, where every catalytic interaction between a sidechain and the metal-substrate complex is described by a set of internal coordinates.

Every residue is represented by an entry in block form. This entry contains two triplets of atoms, required for the definition of the internal coordinates. One triplet of atoms describes the catalytic sidechain, the other one represents the metal-substrate complex. The coordinates themselves encompass the distance, two angles and three dihedrals, and describe the geometric relation between the metal-substrate complex and the catalytic amino acid. Since it is unlikely to find a protein site capable of accommodating the exact theozyme geometry, tolerance values for the sampling of the internal coordinates are assigned in the constraint file.

It is also possible to choose between two matching algorithms: the classic ROSETTA-match algorithm and a secondary matching algorithm. While the classic algorithm allows to enforce exact enzymatic constraints, the secondary algorithm is advantageous for softer restrictions.[22]

5.3.2 *Enzyme Design*

An enzyme design procedure was then applied to the matches in order to tailor the shape of their active sites to complement the substrate geometry and stabilize the arrangement of catalytic sidechains. The ROSETTA Enzyme Design utility was used for this task. The design process can be split into several parts: optimisation of the catalytic interactions, sequence design and a final repacking to asses enzyme quality.

Optimisation of Catalytic Interactions

In an initial step, the residues in vicinity of the metal-substrate complex were grouped into the categories "catalytic", "designable" and "repackable" according to four specified cutoff-radii. Catalytic residues define the theozymatic interactions and were not subjected to sequence design. Designable residues were allowed to mutate during the sequence design steps. Repackable residues remained unchanged, but were subject to the optimisation step invoked during each design cycle, along with the other two residue classes.

Every designable and repackable residue was then mutated into alanine to arrive at a minimal active site consisting only of the metal-substrate complex and the catalytic residues. The resulting geometry was optimised with the steepest descent method on a force field level under retention of the theozymatic constraints, in order to move the metal-substrate complex into the position providing the best catalytic interactions. For information on the general composition of the force field employed by ROSETTA and a discussion of its individual terms, see Ref. 22 and references within.

Sequence Design

In the next phase, the restored active site was subjected to a series of alternating steps of sequence design and constrained energy minimisations to arrive at an optimal sequence folding into the required tertiary structure, while stabilizing the catalytic sidechains.

During sequence design steps designable residues were mutated with the ROSETTA Monte Carlo algorithm.[145]

As an alternative to the default force field, a variant with shorter van-der-Waals distances can be used. The discretion in the sampling of dihedral angles, introduced by the rotamer library employed by ROSETTA, can lead to potentially good conformations being rejected by the Monte-Carlo algorithm, as only a small divergence from an optimal angle may result in suboptimal energy scores. The soft-repulsive force field variant overcomes this problem by allowing small deviations from the ideal angles, leading to the acceptance of more conformers and often resulting in a better overall side-chain packing.[146]

In the following minimisations step, the active site geometry was once again subjected to constrained geometry optimisation.

Unconstrained Repacking

In a last step, the constraints of the theozyme were lifted and the active site geometry optimised. If the theozyme geometry was retained, the design procedure was considered successful.

6

RESULTS

The following chapter presents the results obtained in this work. The *de-novo* design process culminating in the creation of an enzyme with novel functionality will be broken down into its elementary steps and each topic will be treated individually.

Starting with preliminary computations on the model system, the steps leading up to the final theozymes will be detailed. This will be followed by accounts on the search for adequate protein-scaffolds and the redesign procedure. Finally the most promising candidates for *de-novo* enzymes will be presented and their sequential and structural features discussed.

6.1 COMPUTATIONS ON THE 2ND GENERATION GRUBBS CATALYST

The successful design of a theozyme requires extensive knowledge on the energy profile of the reaction to be promoted. The identification of important intermediates and transitions states, as well as the rate limiting steps, allows to address the issue of fine-tuning and optimising performance of the future active site.

To this end, a full cycle of the model compound undergoing RCM in presence of a standard metathesis catalyst, (tricyclohexylphosphine) (1,3-dimesityl-4,5-dihydoimidazol-2-ylidene) methylidene ruthenium dichloride (Figure 6.1) was studied. The choice of a second-generation Grubbs catalyst was due to the overall popularity and the favourable catalytic properties of this class of ruthenium complexes (see section 6.1). It was further assumed that the catalyst had already undergone the initial dissociation step of the phosphine ligand and activation cycle, leading to the active methylidene species.

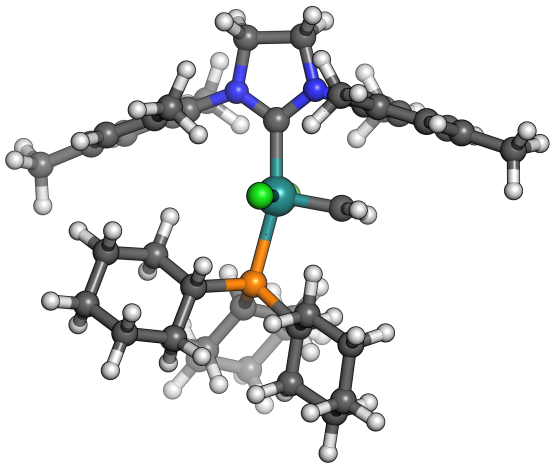


Figure 6.1: Second generation Grubbs catalyst in its methylidene form.

6.1.1 *The Catalytic Cycle*

Based on the reaction mechanism highlighted in section 3.2, it was possible to construct a scheme of the RCM reaction and the species encountered along the reaction path.

Previous computational studies on similar systems show that the encountered π-complexes are stable intermediates.[49,51] As depicted in Figure 3.3, these π-complexes can be separated into two groups, those anteceding the metallacycle and those succeeding the metallacycle. For the first class, there is proof, that a first transition state leading to the formation of the π-complex is barrierfree or possesses a negligible barrier when compared to the second transition state describing the ruthenacycle formation, which typically is one of the rate determining steps of RCM.[49–51] Since this second transition state contributes the most to the overall energetics of the RCM reaction, the π-complex and the first transition state associated with its formation can be neglected. The same holds true for the second class, where the reaction profile is dominated by contributions of the third transition state describing the ring opening of the metallacyclobutane, whereas the energetic influences of the π-complex and the fourth transition state mediating olefin dissociation are negligible in comparison.

During a whole catalytic cycle of RCM leading up to the restoration of the initial active catalyst, four π-complexes, two of each class, are encountered, resulting in a total of eight species (four transition states describing olefin coordination or dissociation and four π-complexes) which play only a minor role in the energetics of the reaction.

Based on these observations, the π-complexes and the transition states describing olefin coordination and dissociation were omitted in the analysis of the RCM reaction. This simplification helped to limit the computational expense of evaluating the energy profile and was especially helpful in the high throughput screening process. The resulting scheme is depicted in Figure 6.2.

Throughout the course of this chapter, the nomenclature listed in Table 6.1 and partly shown in Figure 6.2 will be adopted for the different transition states and stationary points.

Table 6.1: Labels used to refer to the different stages of the RCM reaction.

Label	Description
C1E	Active methylidene complex and diallylether reactant
C1T1	Transition state leading to the first metallacycle
C1M	First metallacyclobutane intermediate
C1T2	Cleavage of the first metallacycle
C1P	Intermediate products of the first metathesis reaction
C2T1	Formation of the second metallacyclobutane
C2M	Second metallacycle intermediate
C2T2	Second metallacycle cleavage
C2P	Final dihydrofurane product and restored catalyst

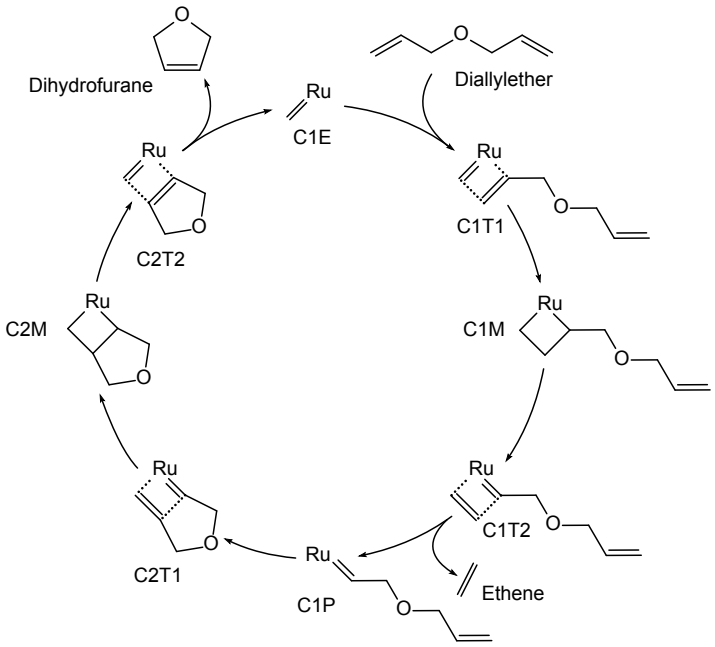


Figure 6.2: The catalytic cycle of one full turnover of the substrate undergoing RCM. Due to the minor influence of π-complexes and transition states mediating olefin coordination and dissociation, these species were neglected in the treatment of the reaction. For sake of clarity the main ligand and both chlorines were omitted.

6.1.2 *Geometries of Stationary Points and Transition States*

Geometries of stationary points (**C1E**, **C1M**, **C1P**, **C2M**, **C2P**) were optimised with B3LYP. Connecting transition states (**C1T1**, **C1T2**, **C2T1**, **C2T2**) were located using the GAUSSIAN QST2 protocol.[132] Minima as well as saddle points were characterized via frequency analysis. For a detailed description of the employed methods refer to section 5.1.

The optimal geometry of the initial active complex without the substrate (Figure 6.3) shows the hydrogen atoms of the methylidene moiety to be perpendicular to the plane spanned by the chlorine atoms and the methylidene carbon atom. This rearrangement compared to the planar conformation in the undissociated complex (Figure 6.1), is due to the absence of the steric pressure exerted by the bulky phosphine-ligand. The presence of the mesityl groups keeps the N-heterocyclic-carbene-ligand in-plane with the Ru-methylidene bond.

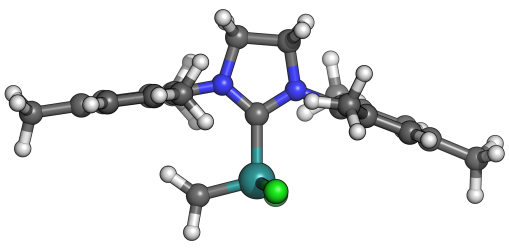


Figure 6.3: Active Grubbs-catalyst **C1E** in its methylidene form after dissociation of the phosphine ligand.

Figure 6.4 depicts the transition state **C1T1** leading to the formation of the first metallacycle. The complex adopts a distorted square-pyramidal geometry after substrate coordination. The methylidene bond and the ether double bond are arranged parallel to each other, thus maximising interactions. Catalyst and diallylether adopt an almost planar arrangement, already strongly resembling the metallacycle.

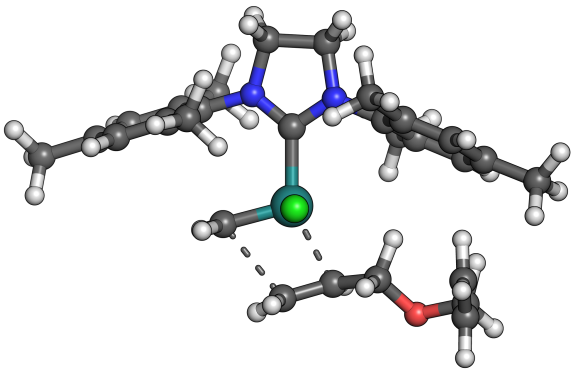


Figure 6.4: Transition state **C1T1** mediating the formation of the first ruthenacycle through insertion of one terminal ether double bond into the ruthenium-methylidene bond.

The resulting metallacyclobutane intermediate **C1M** retains this planar arrangement (Figure 6.5). Compared to the angle of 148° between both chlorine ligands in the initial catalyst, the chlorines adopt an angle of approximately 180° in this case, bisecting the ruthenacycle plane.

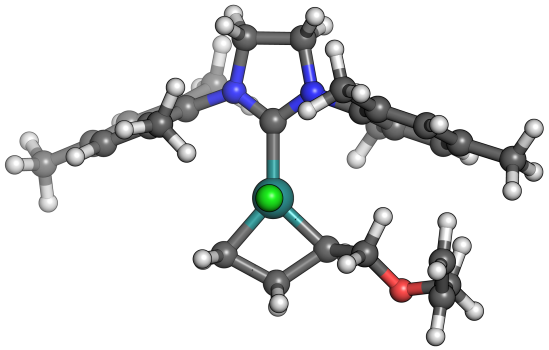


Figure 6.5: First metallacyclobutane intermediate **C1M** encountered during the RCM reaction.

Cleavage of the metallacycle proceeds via the transition state **C1T2** (Figure 6.6), exhibiting a geometry similar to **C1T1** (Figure 6.4).

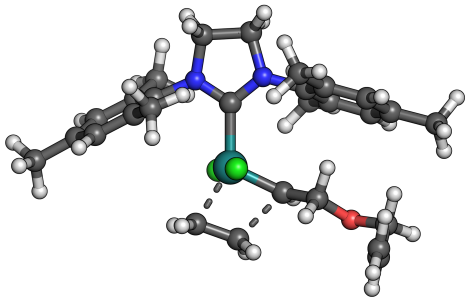


Figure 6.6: Transition state **C1T2** leading to the opening of the metallacyclobutane-ring.

The products of the first metathesis cycle are ethene and the complex **C1P**, resembling the initial catalyst, albeit with the alkylidene functionalities exchanged (Figure 6.7). The Ru-C bond once again adapts a conformation coplanar to the carbene.

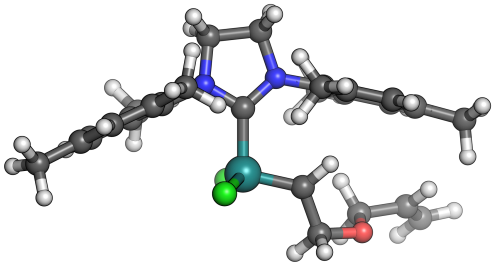


Figure 6.7: Product **C1P** of the first metathesis turnover carrying the ether-alkylidene analogon.

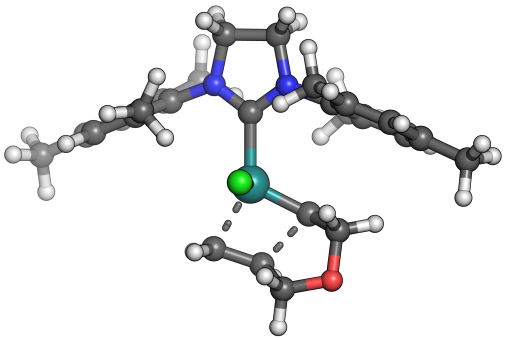


Figure 6.8: Transition state **C2T1** mediating the ring-closure of the second metallacyclobutane species. The transition state geometry already anticipates the forming 5-membered ring in envelope conformation.

Formation of the second ruthenacycle is facilitated by the transition state **C2T1**, as shown in Figure 6.8. As before, the arrangement of the bonds involved in the reaction is highly planar, but the plane is slightly twisted relative to the N-heterocyclic-carbene-ligand. This is due to the transition already anticipating the envelope conformation of the forming 5-membered ring.

The resulting metallacycle **C2M** is very similar to **C2T1** in its overall appearance (Figure6.9). The 5-membered ring has now fully adopted its envelope conformation.

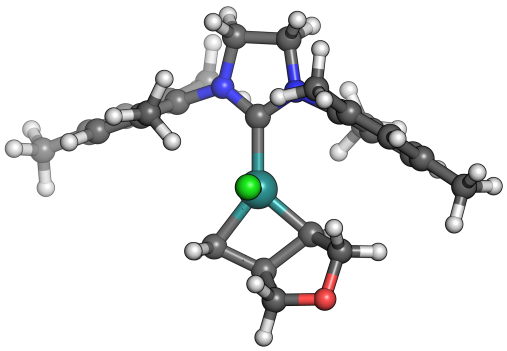


Figure 6.9: Second metallacyclobutane intermediate **C2M** already showing the 5-membered ring.

During the opening of the metallacycle facilitated by the transition state **C2T2**, the future dihydrofurane can be seen adopting its preferred planar geometry (Figure 6.10).

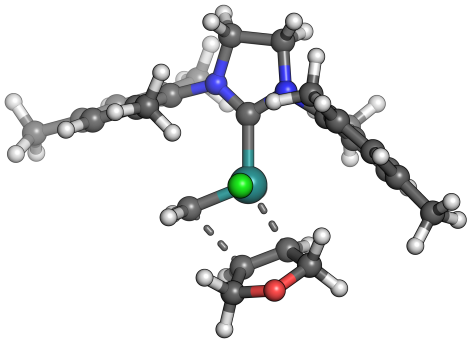


Figure 6.10: Transition state **C2T2** mediating the cleavage of the second metallacycle.

In the last step, dihydrofurane dissociates from the catalytic complex, which is restored to its initial form **C1E**, thus closing the RCM cycle.

The found geometries strongly resemble the ones reported by previous experimental and theoretical studies on similar systems.[49,50,52,121]

6.1.3 *Free Energy Curve of the 2nd Generation Grubbs Catalyst*

An energy profile of the reaction was compiled based on the obtained structures. In order to account for entropic and thermochemical effects important for the RCM reaction, the free energy computed with B3LYP was used to describe the reaction profile.

Figure 6.11 shows the free energy barriers of the different species encountered along the reaction coordinate relative to the energy of system consisting of the active complex and the reactant.

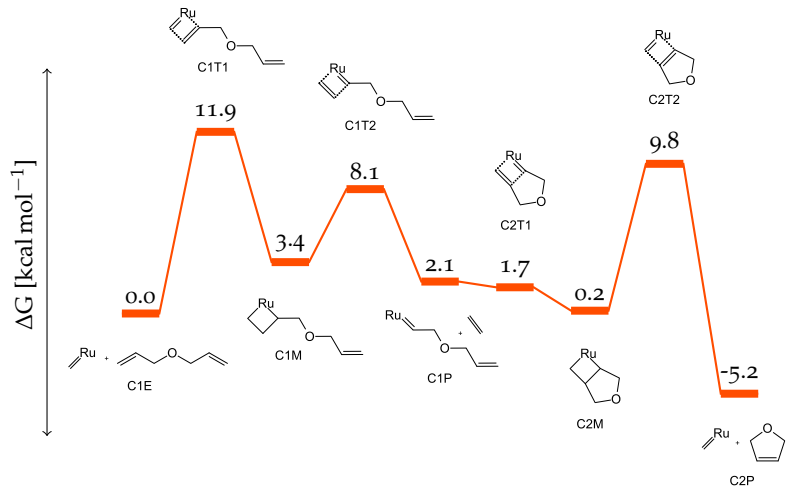


Figure 6.11: Free energy curve of one RCM catalysis cycle.

Several observations can be made regarding the free energy landscape of the RCM reaction:

- The metallacycles **C1M** and **C2M** appear as intermediates rather than transition states, which is in agreement with recent computational results[49,50] and experimental findings.[51,147]

- The transition states either leading to or from the metallacyclobutane are usually accepted to be the rate limiting steps of the metathesis reaction. The relatively high energy barriers computed for ring cleavage and formation in the model system are in good agreement with this observation. A notable exception is the **C2T1** transition state, which lies significantly lower in energy than its analogon **C1T1**. This dramatic difference can be explained by considering entropic effects. The transition state **C1T1** on the one hand describes the coordination of the ether to the complex, a step which leads to a significant decrease in the system's entropy. **C2T1** on the other hand constitutes an intramolecular reaction and the disadvantageous entropic influence is diminished.

- Thermodynamic reasoning also serves to elucidate the difference in energy between **C1T1** and **C1T2**, since the opening of the ruthenacycle associated with **C1T2** is favoured because of an increase in entropy. However, this argument fails when applied to the second cycle transition states, as **C2T1** lies lower in energy than **C2T2**. The relative height of the barrier associated with

C2T2 is the consequence of the developing ring strain during the formation of the 5-membered dihydrofurane.

According to the computational results, the rate determining step of the RCM reaction with diallylether as substrate is either **C1T1** or **C2T2**, *i.e.* the formation of the initial metallacycle or of the 5-membered product.[49,51] Although **C1T1** exhibits a higher activation barrier compared to **C2T2**, the difference of 2.3 kcal mol^{-1} is of the same magnitude as potential errors introduced by intrinsic limitations to the quantum chemical methods employed. The free energy curve alone therefore provides no reliable criterium to discern which one of the two steps truly is rate-limiting.

A comparison with calculations done on the all-carbon analogue of the substrate shows a significantly lower insertion barrier, but only small deviations regarding the final ring-opening step.[52] Experimental studies report good reactivity and conversion rates for the pure carbon substrate under standard metathesis conditions, whereas diallylether requires increased temperatures and prolonged reaction time and only incomplete conversion is achieved.[148] The logical consequence of combining these two findings suggests, that the initial metallacycle formation **C1T1** is indeed the rate-limiting step of the overall RCM reaction. A possible reason for the increased barrier height is a destabilizing electronic influence exerted by the ether oxygen atom.

The free energy barrier of **C1T1** was thus chosen to be the main target of the stochastic optimisation procedure of the genetic algorithm and the fitness function was modelled accordingly (recall section 5.2.5).

6.2 AMINO ACID ALTERNATIVES TO THE CARBENE LIGAND

One of the issues remaining to be addressed before starting the automated theozyme design process is how to attach the ruthenium moiety to a protein. The previous computations were performed with the active variant of the second-generation Grubbs catalyst bearing a heterocyclic carbene. Albeit possible in principle, it is challenging to incorporate carbene-moieties into a protein scaffold. A more natural approach would be to exchange the ligand by an amino acid residue if possible, which would also greatly simplify future synthesis attempts.

6.2.1 *Candidates for Alternative Ligands*

To search for possible alternatives to a carbene ligand, free energy calculations of the reaction curve were performed for several amino-acid-substituted versions of the complex. To allow coordination to the ruthenium atom, the potential candidates were required to contain a heteroatom. Positively charged residues were dropped early during the screening process, as they tended to coordinate to the chlorine atoms instead of the metal center. In a similar manner, aromatic amino acids forming a possible η-6 complex were neglected due to observed instabilities.

These criteria reduced the list of amino acid complexes to be examined to the six compounds shown in Figure 6.12. They can be grouped into three main classes: coordination via a nitrogen atom (histidine),

coordination via sulphur (methionine, cysteine) and coordination via oxygen (threonine, tyrosine and aspartic acid).

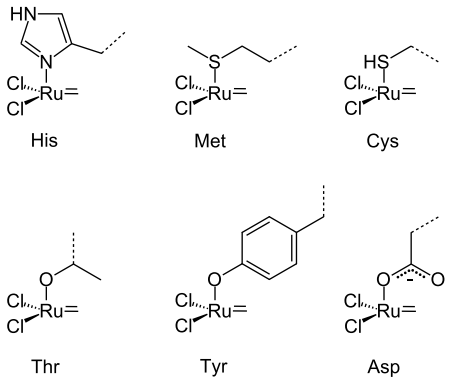


Figure 6.12: Amino acid complexes studied as potential alternatives to the N-heterocyclic-carbene-catalyst.

6.2.2 *Free Energy Profiles of the Amino Acid Catalysts*

Geometry optimisations and free energy computations were carried out according to the standard B3LYP protocol defined in section 5.1. The resulting energy curves can be found in Figure 6.13 and the associated values are given in Table 6.2.

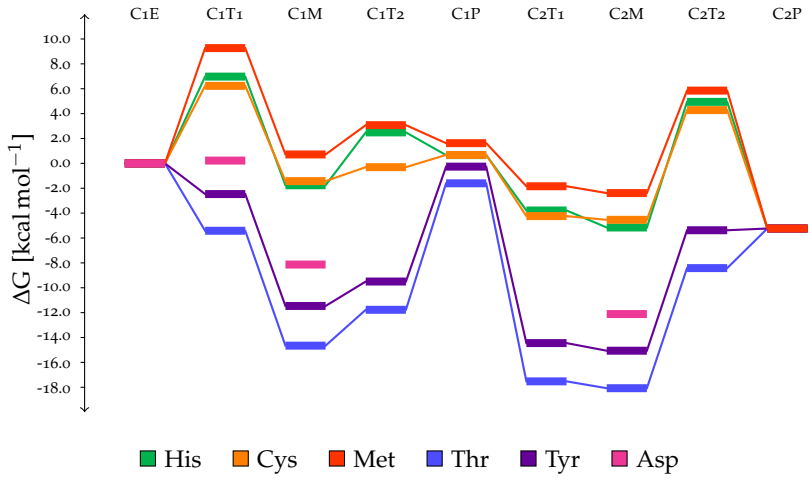


Figure 6.13: Free energy profiles obtained for the amino acid carrying catalysts.

Two different trends are noticeable. Fist of all, the oxygen coordinated ligands – threonine, tyrosine and aspartic acid – seem to overstabilize the metallacycle intermediates. This property could lead to the reaction becoming trapped at the stage of the ruthenacycle species, as the depth of the associated wells on the free energy surface makes further chemical transformations unlikely, rendering this group of amino acid ligands unsuitable for catalysis.

The second trend is that the other ligands – histidine, methionine and cysteine – compare very favourable to the N-heterocyclic carbene ligand of the original second-generation Grubbs catalyst (section 6.1), as the general properties of the reaction profile follow the trends dis-

Table 6.2: Free energies of the amino acid bearing complexes.

	ΔG [kcal mol^{-1}]					
Species	His	Met	Cys	Tyr	Thr	Asp
C1E	0.00	0.00	0.00	0.00	0.00	0.00
C1T1	6.97	9.27	6.23	-2.47	-5.42	0.22
C1M	-1.76	0.72	-1.43	-11.46	-14.65	-8.13
C1T2	2.49	3.07	-0.30	-9.50	-11.77	–
C1P	0.66	1.62	0.68	-0.26	-1.60	–
C2T1	-3.79	-1.84	-4.24	-14.45	-17.52	–
C2M	-5.15	-2.39	-4.54	-15.06	-18.07	-12.11
C2T2	4.94	5.85	4.28	-5.38	-8.42	–
C2P	-5.24	-5.24	-5.24	-5.24	-5.24	–

cussed in section 6.1.3 and the curves almost coincide with the one obtained for the original complex. Some of the barriers actually lie even lower in energy. While these observation seems to favour the amino acid ligands over the native carbene, no active metathesis catalysts of such a form have yet been reported in literature. One QSAR study assessing various compounds with respect to their feasibility as olefin metathesis catalysts explores similar compounds and attributes only subpar activity to the related complexes.[149] Unfortunately, a direct comparison of the results is not possible as the measure of productivity used in the study includes the dissociation barrier of the phosphine ligand not investigated in this work.

Taking into account these observations, it is prudent to assume that the obtained energy profiles are at least to a certain extent distorted due to the following computational effects:

- The accuracy of the functional and the basis set size are limited.
- No corrections for the basis set superposition error were employed.
- Only one low energy conformation of the amino acid ligands was sampled in each case.
- There are inherent problems when computing the free energy of compounds with low frequency modes (see section 4.4.3).

These effects could indeed shift the activation barriers to higher energies, making the associated reaction steps less favourable.

Nevertheless, despite all the caveats, a substitution of the carbene ligand by either methionine, histidine or cysteine seems viable for the approach pursued in this work. The general form of the reaction profile is promising and it can be expected to improve significantly, since the initial amino-acid-bearing complex only serves as a starting point for further stochastic optimisation in the framework of the genetic algorithm.

6.3 THEOZYME MOTIFS

Based on the insights gained in the previous sections, exploratory runs with the generational algorithm were performed to help to identify common motifs introducing beneficial interactions.

6.3.1 *Exploratory Search*

In order to perform a quick assessment of potential theozyme compositions, the genetic algorithm was used in a mode of operation resembling Monte-Carlo search strategies. Only one generation with a representative population size p_{max} of 60 individuals was computed (see section 5.2). All 21 eukaryotic proteinogenic amino acid residues (see Figure 5.2), with the exception of glycine, alanine and proline, were allowed during genome creation and the size of individual genomes was limited to between $g_{min} = 4$ and $g_{max} = 6$ amino acids. The resulting theozymes with the best fitness values were subjected to further analysis.

Three different central templates were used for the runs:

- The methionine complex.
- The histidine complex.
- A minimal complex stripped of one chlorine and the main ligand, to study the effect of chlorine substitution by negatively charged amino acid residues.

The cysteine-substituted complex was not used, as it provides less steric constraint when compared to the alternatives and could give rise to complications during later synthesis attempts.

A total of four runs was performed, one for each amino acid catalyst and two for the minimal template.

6.3.2 *Common Motifs*

Out of the 240 generated theozyme candidates, only the four most promising with regards to common beneficial interactions will be discussed here. A list of the compounds and their respective amino acid composition is given in Table 6.3. In the following sections, the beneficial catalytic motifs associated with these theozymes will be introduced.

Table 6.3: Main representatives of recurring structural motifs. The template name is composed of the three letter code of the main amino acid ligand (Figure 5.2), the generation (G*X*) and the index number of the individual's genome in the current generation (I*XX*).

Template	Theozyme	Residues
Histidine	his_G1_I25	Arg Thr Gln
Methionine	met_G1_I55	Phe
	met_G1_I57	Glu Gln Ile
Minimal	nt2_G1_I16	Met Asp

Coordination of Chlorines

A recurring structural feature observed in the resulting geometries, was the coordination of at least one of the chlorine atoms by a polar amino acid. An excellent example is his_G1_I25 (Figure 6.14), where one of the chlorines forms a hydrogen bond to glutamine, which in turn possesses a hydrogen bond to the neighboured arginine residue. The same phenomenon is exhibited by the other chlorine. Here, a threonine forms a small hydrogen bond network with the main ligand histidine and the chlorine.

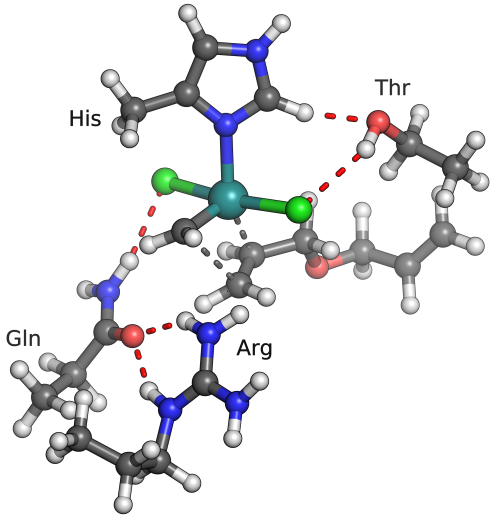


Figure 6.14: his_G1_I25 geometry showing favourable coordination of both chlorine atoms by the polar amino acids glutamine and threonine.

A similar arrangement is found in the methionine complex met_G1_I57, shown in Figure 6.15. In this case the polar glutamine coordinates one of the chlorines and the substrate simultaneously.

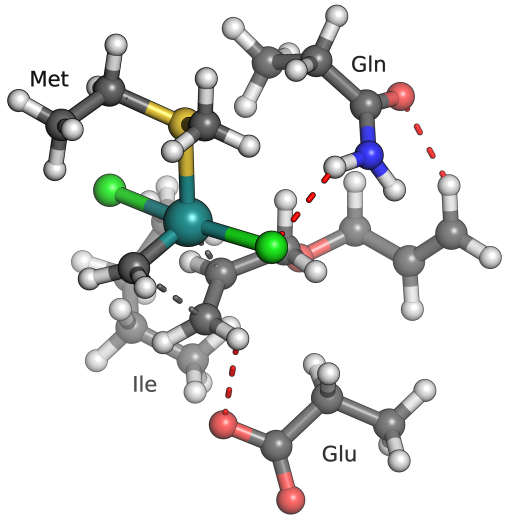


Figure 6.15: Complex met_G1_I57 exhibiting interactions between the chlorine and the glutamine residue.

With regards to energetics, both complexes show the desired stabilisation of the transition state leading to very favourable insertion barriers (Figure 6.16). Inspection of the subsequent steps also show an overstabilisation of the metallacycle resulting in an increase in the energy required for the formation of the final products. It should be noted, that no stochastic optimisation has taken place during the exploratory runs, leading to only suboptimal reaction profiles, which nevertheless exhibit the desired traits.

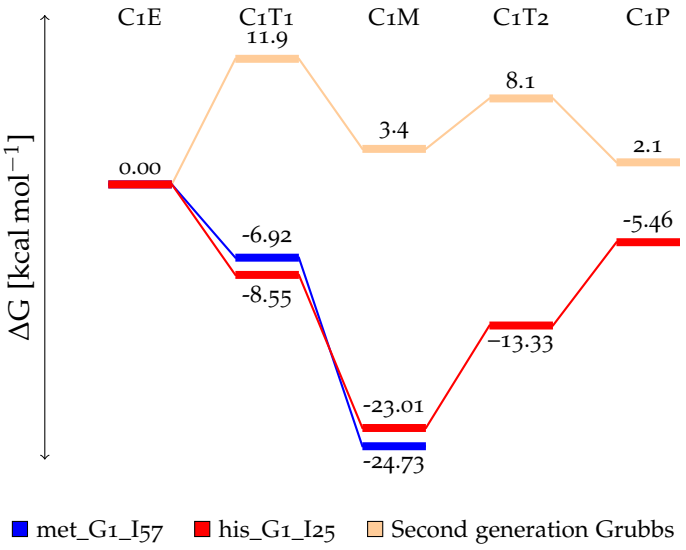


Figure 6.16: Differences in free energy for the chlorine coordinated complexes, computed for the first metathesis cycle. The energies obtained for the second-generation Grubbs catalyst were added for comparison.

Stabilisation of Double Bonds

Another regular motif found is the stabilisation of one of the substrate double bonds via an aromatic residue. A good example is the placement of phenylalanine in the methionine complex met_G1_I55 (Figure 6.17).

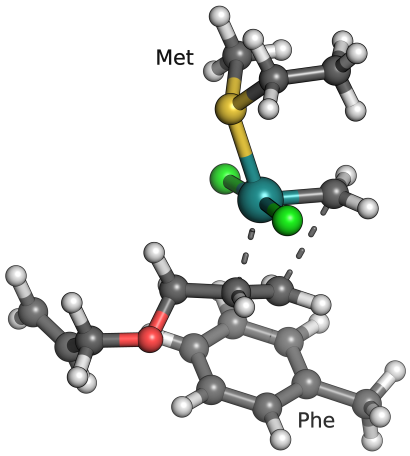


Figure 6.17: The coordination of the ether double bond before insertion is easily visible in met_G1_I55.

A potential positive side-effect of this kind of interaction is the prearrangement of the substrate for the insertion steps occurring in both metathesis cycles. This is known to be one of the main strategies employed by native enzymes (see chapter 2) and might thus exert a beneficial influence on the overall reaction profile (Figure 6.18).

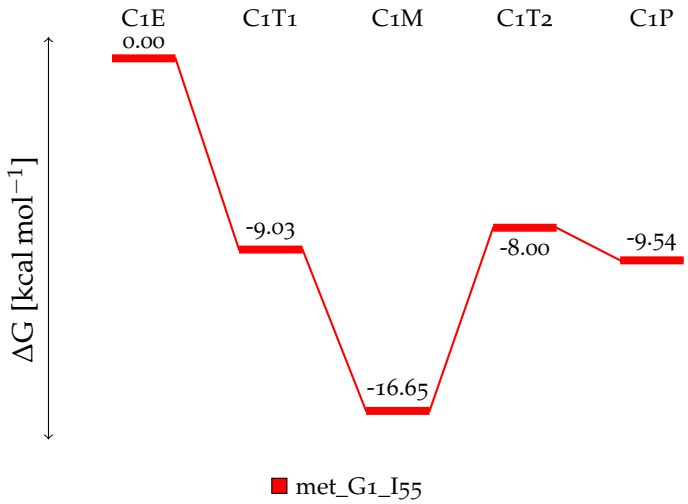


Figure 6.18: Sample of the free energy profile for the metathesis reaction promoted by met_G1_I55

Chlorine Replacement

Although possible in principle, chlorine substitution by negative residues proved to have a negative effect on the reaction cycle. A characteristic example is nt2_G1_I16 (Figure 6.19), that shows the preferred conformation of aspartic acid groups when coordinating to the central template. While this arrangement seems sensible at a first glance, it strongly inhibits the cleavage of the ruthenacycle intermediate. This effect is caused by the distortion of the favoured geometry of the forming alkylidene due to the building of considerable steric pressure.

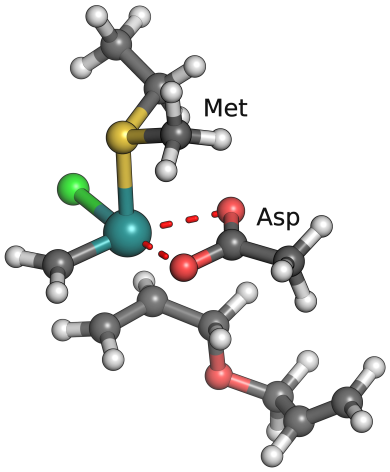


Figure 6.19: The geometry of nt2_G1_I16 shows a substitution of one chlorine by an aspartic acid residue. This arrangement strongly inhibits the formation of the ruthenacycle opening.

The associated free energy curve of nt2_G1_I16 is depicted in Figure 6.20. The steric influence of the aspartic acid group can be seen clearly for the transition step **C1T2**, for which the barrier height is greatly increased, when compared to the previous examples.

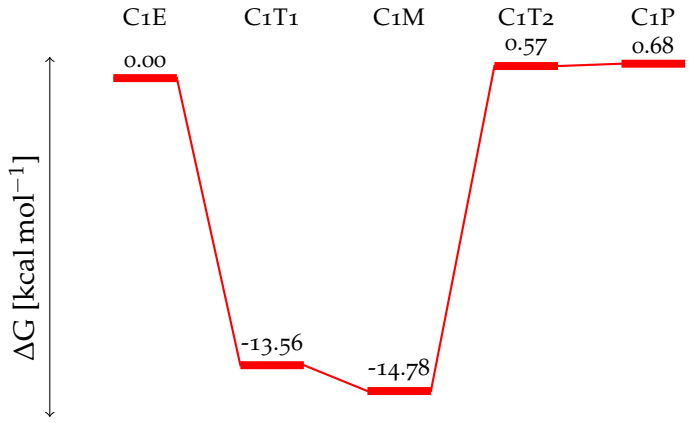


Figure 6.20: Free energy profile of nt2_G1_I16 showing the inhibition of the metallacycle cleavage by the aspartic acid residue.

The Main Ligand

Given the observations described above, the choice of a main ligand has essentially been narrowed down to histidine or methionine. Due to the stochastic nature of the method, the results of the genetic algorithm screening do not provide a decision basis in this case.

Fortunately, simple considerations regarding the pK_a of the involved amino acid side chains provide a general guideline. The imidazole moiety of histidine possesses a pK_a of approximately 6. At physiological pH-levels, as typically encountered in protein environments, this property can lead to rapid switches between the neutral and protonated state based on only small fluctuations of the pH. The presence of a positively charged main ligand would in turn negatively influence the overall reaction (see section 6.2.1).

This problem is not encountered with methionine, making it the better choice for the main ligand.

6.3.3 *The Proto-Theozyme*

With the insights gained from the exploratory runs of the genetic algorithm, a model of the theozyme was constructed. This model combined the following beneficial motifs:

- Methionine constitutes the main ligand
- Coordination of the chlorine atoms by a polar amino acid (serine, threonine, arginine, glutamine)
- Coordination of the double bonds by phenylalanine

This general arrangement of amino acids was used as a starting point for further exploratory runs and genetic optimisations.

6.4 THE FIRST THEOZYME

Based on this theozyme model, an exploratory stochastic search of the potential amino acid arrangements and combinations was performed.

Similar to the examples above, a population size of 60 genomes was used and no stochastic optimisation took place. The metathesis catalyst without main ligand but both chlorines present served as a central template. Amino acids in the vicinity of the chlorines were constrained to the polar residues. The main ligand of the ruthenium atom was set to be methionine and the placement of a phenylalanine near the coordinated olefinic double bond was required during geometry creation.

An important difference to the previous searches is the use of the Grimme D3 dispersion correction to augment the B3LYP computations performed with GAUSSIAN, instead of the D2 correction employed until now. The D3 correction was introduced in a new revision of GAUSSIAN, which became available during the course of this work. As described in section 4.3.5, D3 usually improves upon D2 and it was thus used as the standard dispersion correction for GAUSSIAN computations from this point onwards.

6.4.1 *Geometry of the First Theozyme*

Among several geometries with suboptimal activation barriers, a complex with an a insertion energy of 3.60 kcal mol^{-1} was obtained (Figure 6.21). The complex G1_I38 exhibits all of the desired traits. Both chlorines are coordinated by either threonine or serine. The methionine ligand adopts a conformation almost perpendicular to the chlorine axis, while the phenylalanine is placed in a manner strongly resembling met_G1_I55.

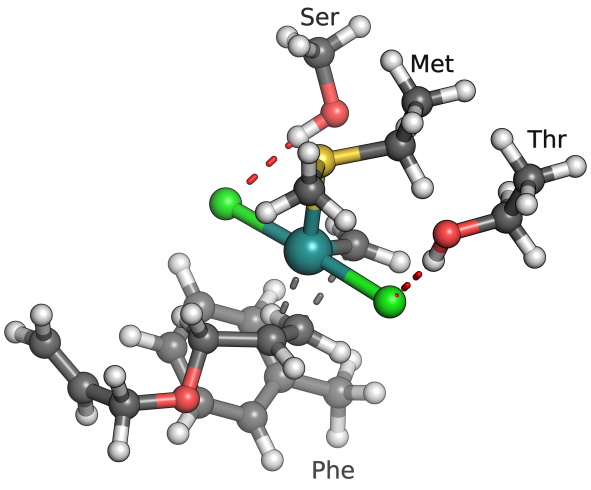


Figure 6.21: Theozyme G1_I38 found with the stochastic screening process.

The free energy profile of the RCM reaction catalysed by G1_I38 is given in Figure 6.22.

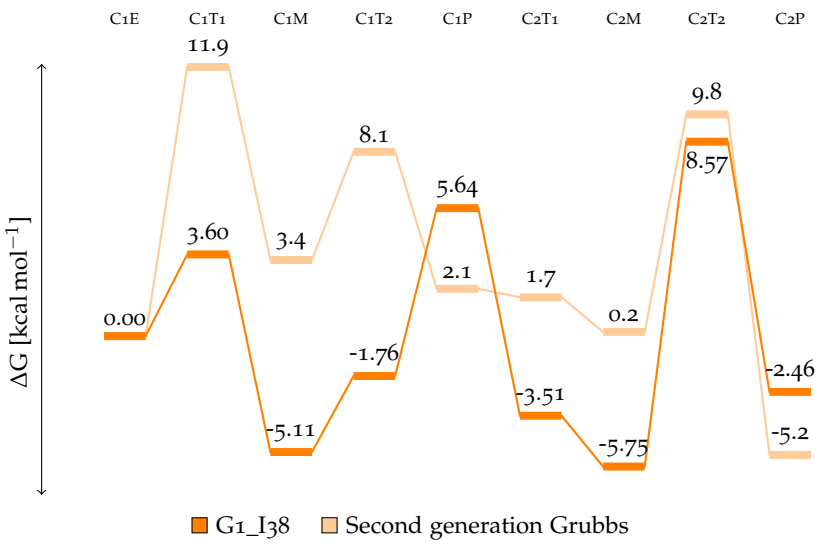


Figure 6.22: Free energy profiles of the metathesis reactions catalysed by G1_I38 and the original second-generation Grubbs methylidene catalyst. In both cases, the Grimme D3 dispersion correction is used instead of D2 in order to augment the standard B3LYP procedure.

The reaction energetics compare favourably to those of the initial second-generation Grubbs catalyst. The activation barrier for the formation of the first metallacyclobutane is significantly lowered, which is a good confirmation of the choice of catalytic motifs. Another positive aspect is the increased destabilisation of the ruthenacycle intermediate in contrast to e.g. met_G1_I55, since this reduces the risk of the reaction terminating at the ruthenacycle stage.

The transition state facilitating the metallacycle cleavage on the one hand is almost unchanged compared to the model reaction, which can be attributed to the type of fitness function employed (see section 5.2.5). The intermediate and final product stages on the other hand are slightly destabilized, changing the reaction profile in a disadvantageous manner.

It should be stressed once again, that this complex was obtained in a stochastic manner without any genetic optimisation process taking place. It is thus reasonable to assume that it only constitutes a local minimum of the search space and, while the reaction profile is already promising, room for improvements still exists.

6.4.2 *The Quest for a cheaper Fitness Function*

While suitable for exploratory search, the generational algorithm employed proved to be extremely impractical with regards to optimisation, as the periods of idle time due to the differing durations of the evaluation process led to a suboptimal use of computational resources (see discussion in section 5.2). Therefore, the switch to the steady state model improved the usage of computational resources.

Another way to increase efficiency is to introduce a cheaper fitness function. The easiest solution to save time is to replace the hybrid functional B3LYP, which requires the costly evaluation of Hartree–Fock exchange, with a GGA functional. This change in functional further-

more allows to make use of the RI approximation (see section 4.3.4), resulting in an additional speed-up of the process.

One problem associated with a change of functionals is a potential reduction in accuracy, as the performance of different functionals can vary greatly depending on the chemical system treated. However, the good results reported for computations on transition metal complexes using the BP86 functional[47,138–141] strongly support the application of this approach.

In order to assess the viability of the BP86 functional for the problem at hand, computations were carried out on the geometries of theozyme G1_I38 using this functional and the results compared to the B3LYP-D3 reaction profile discussed in the previous section. After an optimisation with the BP86 approach described in section 5.1.1, the free energies were computed. The calculations were performed with TURBOMOLE. The comparison of the results to the free energy profile of G1_I38 can be found in Figure 6.23.

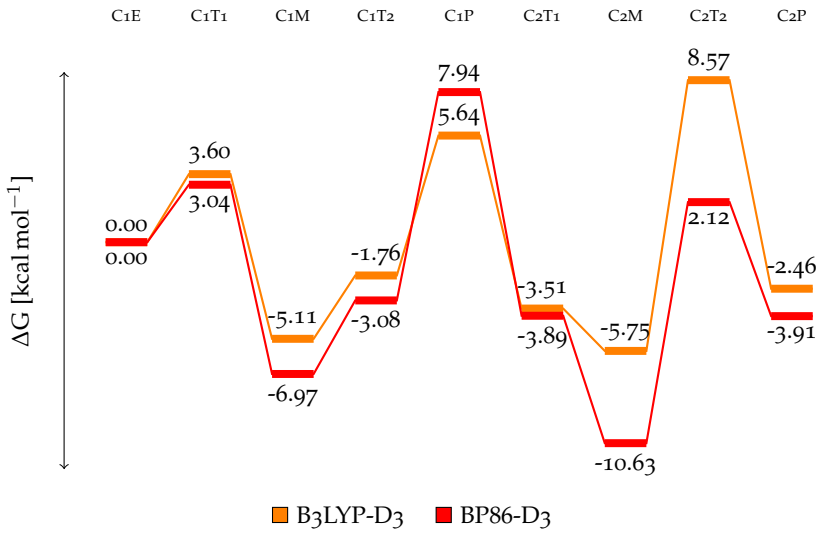


Figure 6.23: Comparison of the reaction curves obtained for the G1_I38 theozyme using the BP86 and B3LYP functionals.

For the first metathesis cycle, the results obtained for BP86 are very close to the B3LYP ones. A deviation from this trend is found for the second metallacycle and its cleavage. While this deviation from the B3LYP functional might be interpreted as an error of the BP86 GGA functional, several benchmark studies on transition metal complexes suggest otherwise. These studies report an excellent accuracy of the BP86 functional for geometries, energies and vibrational frequencies for this special class of compounds, whereas the accuracy of B3LYP is often only mediocre in comparison.[47,138–141] The switch to a cheaper fitness function employing the BP86 functional therefore not only leads to a tremendous gain in efficiency, but might additionally even improve the quality of the results.

6.5 THE THEOZYMES FROM THE STEADY-STATE ALGORITHM

To obtain a set of suitable theozymes which can be used in the subsequent matching and enzyme design procedure, the steady state genetic

algorithm was employed. The use of a steady state mode of operation together with a more efficient fitness function allows to carry out the task of global optimisation neglected in the previous search runs.

6.5.1 *Algorithm Parameters*

The previously determined theozyme motifs, like e.g. the coordination of the chorines by polar amino acids (see section 6.3.3), were used in the creation of the different individuals. Parameter settings employed during the steady state run can be found in Table 6.4, a detailed discussion of each parameter is given in section 5.2.

Table 6.4: Set of parameters employed for the steady state genetic algorithm.

Parameter	Value	Description
$p_{\max}$	60	Maximal population size
g_{size}	4	Number of genes in genome
t_{crit}	4	Genetic operations threshold
n_{eval}	300	Number of evaluations
n_{tour}	$f(p_{\text{curr}})$	Tournament size
n_{off}	2	Number of offspring
n_{new}	1	New species
r_{cx}	0.4	Crossover rate
r_{mut}	$f(p_{\text{curr}})$	Mutation rate

6.5.2 *Fitness Evolution*

The optimisation progress was monitored with the help of three variables, the average fitness F_{avg}, the minimum fitness $F_{\min}$ and the maximal fitness $F_{\max}$ of the population. Since the fitness measure essentially resembles the activation barrier of the complex, smaller fitness values represent better solutions to the problem at hand and a steady decrease of the overall fitness is desired.

This definition might seem counterintuitive, however, it allows for the direct interpretation of the fitness of every individual as the free energy in kcal mol^{-1} needed for the formation of the first ruthenacycle. To facilitate interpretation of the results, no scaling of the fitness values derived from quantum-chemical computations took place. As a result of introducing this convention, the replacement operator in Figure 5.1 now uses the minimum function instead of the maximum function and the tournament selection operator chooses the least fit genome.

The variables F_{avg}, $F_{\min}$ and $F_{\max}$, used for monitoring the stochastic optimisation process, were computed after every finished evaluation of an individual. F_{avg} is the average fitness of the current population, calculated as the sum of fitnesses over every individual currently present in the population pool, divided by the size of the population p_{curr}. This value helps to monitor the overall progress of the stochastic optimisation procedure. The variable $F_{\min}$ essentially measures how efficiently good solutions are generated, as it describes the fittest

solution, whereas F_{max} represents the worst individual and provides insights with regards to the selection procedure and selection pressure, as it allows to discern how fast the worst results are filtered out.

The evolution of F_{avg}, F_{min} and F_{max} during the algorithm run is shown in Figure 6.24. The number of evaluations n_{eval} serves as a measure for the elapsed time.

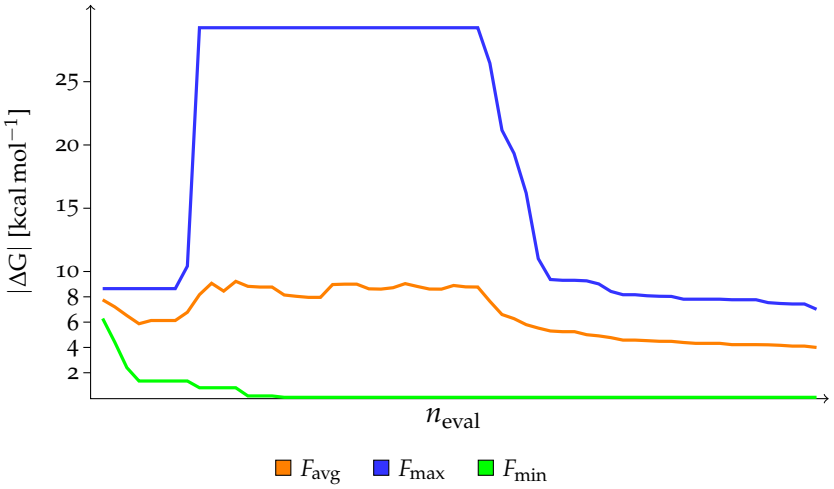


Figure 6.24: Change of the average fitness F_{avg}, the minimum fitness F_{min} and the maximal fitness F_{max} over the course of the genetic algorithm run. The number of evaluations done n_{eval} serves as a time measure. The fitness values can be interpreted as the absolute of the activation barrier ΔG in kcal mol^{-1}.

The curves F_{avg} and F_{max} show an initial saturation period of the population until the maximum population size p_{max} is reached. The introduction of individuals with suboptimal fitness into the growing population pool during this phase, leads to an increase in both variables. Once a particularly bad individual is introduced into the population, F_{avg} and F_{max} remain approximately constant, as no extinctive selection takes place yet. This phase is important, as the relatively low selection pressure allows for the exploration of the fitness landscape, thus counteracting the premature convergence to a local optimum.

As soon as p_{max} is reached, the extinctive replacement procedure as well as the growing tournament size adjust the selection pressure and the system begins to improve towards solutions of better quality.

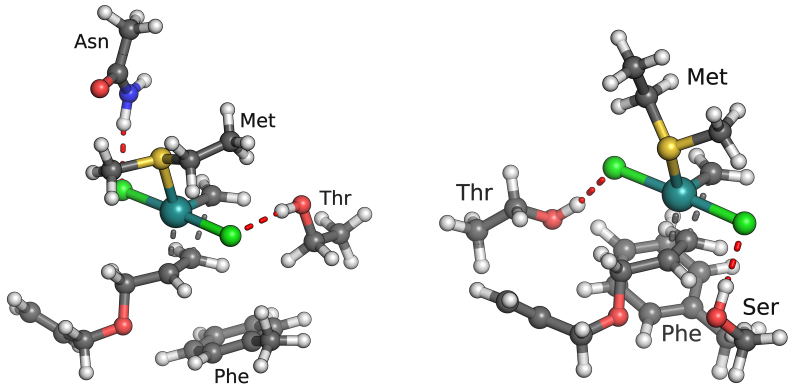
The curve F_{min} shows, that very fit individuals were already introduced in the early stages of the genetic algorithm. This effect can be expected to exert a positive influence on crossover operations, as these fit individuals provide good genetic material for the offspring generated.

These observations attest the validity of the stochastic optimisation approach adopted in this work to screen for feasible theozymes.

6.5.3 *Theozyme Geometries*

The detailed discussion of all 300 theozymes yielded by the steady state search will be omitted at this point, as the essential features of the solutions can be showcased with the help of only two examples.

The sample geometries I015 and I279 are depicted in Figure 6.25. The numbers indicate the evaluation cycle during which the structures were produced, meaning that I015 was created at the beginning and I279 almost at the end of the steady state genetic algorithm run.



(a) (b)

Figure 6.25: Theozymes I015 (a) and I279 (b) generated by the genetic algorithm.

The BP86 free energies of activation are 1.6 kcal mol^{-1} for the complex I015 and 2.1 kcal mol^{-1} for I279. These barrier heights have improved dramatically compared to the native catalyst (11.9 kcal mol^{-1}) and moderately compared to the theozyme obtained through stochastic sampling (3.0 kcal mol^{-1}), demonstrating the power of the stochastic optimisation approach taken in this work. Although care should be taken when comparing directly to the values obtained for the model reaction, because different methods were used, the overall trend is readily noticeable.

While both theozymes exhibit the desired combination of motifs defined in section 6.3, there are nevertheless several marked differences. Regarding the amino acid composition, asparagine and threonine form hydrogen bonds to the chlorines in I015, while in I279 the chlorines are coordinated by threonine and serine. The greatest deviation lies in the orientation of the catalytic residues. The hydrogen bond donors in I015 point in the direction of the methylidene bond in case of threonine and of the sulphur-ruthenium bond in case of the arginine. In I279 however, both donors are placed on the other side of the complex, oriented in the direction of the diallylether. A similar observation is made for the phenylalanines. The arrangement of the aromatic ring is almost mirrored through the plane of the forming ruthenacycle when comparing the two theozymes. The methionine ligand is coplanar to the methylidene bond in I015, while in I279 it is perpendicular to it, with the C_β atom facing upwards.

The difference in the observed geometries suggests that the stochastic optimisation procedure has not converged to a global optimum and the theozymes obtained probably only represent local solutions, albeit good ones. This conclusion is supported by the fitness progress shown in Figure 6.24, where the graphs still exhibit a perceivable slope at the last timestep. Longer run times would be required for the algorithm

to converge in accordance with the fact that the free energy fitness landscape is very rugged.

The diversity of the different theozyme geometries however is not necessarily a problem, on the contrary: it increases the likelihood of finding a suitable scaffold to accommodate the active sites in the following matching procedure, as will be seen in the following section.

6.6 THE SEARCH FOR PROTEIN SCAFFOLDS

With the optimized theozymes at hand, the next step of the enzyme design procedure is the screening for protein scaffolds onto which the catalytic residues of the theozymes can be crafted.

Out of the theozymes obtained with the steady state genetic algorithm, the most promising ones with respect to geometry and fitness were chosen for the matching process. The theozyme G1_I38 yielded by the stochastic search was also included. The set of theozymes, their activation barriers and their composition are summarized in Table 6.5. Only the hydrogen-bond donating residues are listed, as the others remain the same for every catalyst (see also Figure 6.25).

Table 6.5: Theozymes used in the matching procedure, with the associated free energy of activation in kcal mol^{-1} and a list of the hydrogen bond donors.

Theozyme	ΔG	Residues	
G1_I38	3.0	Thr	Ser
I004	1.2	Thr	Asn
I015	1.6	Thr	Asn
I156	1.8	Thr	Asn
I159	1.2	Thr	Ser
I192	0.7	Thr	Thr
I279	2.1	Thr	Ser

6.6.1 *Primary Matching*

Initial searches for proteins with suitable backbone conformations were undertaken using the primary matching algorithm of ROSETTA and the scaffold library introduced in section 5.3.

As the theozymes contain four catalytic residues, four constraints need to be specified in the cst-file required for the matching process. In case of the primary matching algorithm, every residue is hashed independently. An example for an entry in the geometric constraint file is given in Figure 6.26.

The first two blocks bearing the `TEMPLATE` keyword specify the substrate and residue involved in the catalytic contact, as well as three atoms each. The atoms define the internal coordinates determining the relative positioning of amino acid and substrate to each other.

These are specified in the `CONSTRAINT` block. The first column holds a description of the coordinate (see Figure 6.27) followed by the respective value. The third column is the tolerance within which the

```
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_name: FE1 C2 C3
TEMPLATE:: ATOM_MAP: 1 residue3: THZ

TEMPLATE:: ATOM_MAP: 2 atom_name: SD CG CB
TEMPLATE:: ATOM_MAP: 2 residue3: MET

CONSTRAINT:: distanceAB:     2.34   0.1  100.0     1   0
CONSTRAINT::    angle_A:   178.06   2.0   50.0  360.   1
CONSTRAINT::    angle_B:   111.47   2.0   50.0  360.   1
CONSTRAINT::  torsion_A:  -100.71   5.0   60.0  360.   2
CONSTRAINT:: torsion_AB:   -27.50   8.0    0.0  360.   2
CONSTRAINT::  torsion_B:   100.41   5.0   60.0  360.   2
CST::END
```

Figure 6.26: Entry of the geometric constraint file showing the block specifying the positioning of the main methionine ligand.

coordinate will be sampled. The fourth column holds a penalty for later force field evaluations, while the fifth entry gives information on the nature of the bond in case of `distanceAB` or on the periodicity of the constraint for the rest. The last field indicates how often the coordinate is to be sampled.

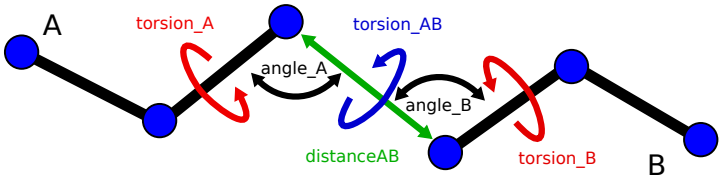


Figure 6.27: Internal coordinate notation used in the ROSETTA constraint file.

Use of the primary algorithm is usually advantageous in situations for which the geometry of the active site is well defined, such as in this case.

Unfortunately, no matches were obtained using this search mode. This result was to be expected, since a shortcoming of the primary matching protocol is a decrease in the likelihood of finding a match for every additional constraint given.

6.6.2 *Secondary Matching*

A secondary matching algorithm exists as an alternative to the primary one. In this case, the residue rotamer is not sampled from all accessible backbone positions and the associated theozyme placement checked for matches with previous placements. Instead, the residue rotamer is "grown" relative to a previously obtained theozyme placement and its C_α atom is checked for possible matches within the accessible backbone positions.[150] This approach allows for a more tolerant sampling procedure, as not all six internal coordinates have to be specified explicitly and is usually used for interactions not clearly defined.

Due to the catalytic contact formed by the phenylalanine residue being the least rigid one, secondary matching was used to sample this interaction. The remaining catalytic contacts were still treated

with the standard algorithm. The problem of resulting suboptimal orientations of phenylalanine could then be addressed during the following sequence design stage.

This change in strategy produced several matches for the different theozymes, distributed over a range of protein scaffolds (Table 6.6).

Table 6.6: Matches found during screening. The PDB-codes of the scaffold proteins are listed for each theozyme.

Theozyme	Scaffolds
G1_I38	-
I004	2R9P 2CZD
I015	3IXB 2WIW 1U00 3C9U 1PJ5 1JQ5 3E3P
I156	1R7A 2WIW 1U00 1N1S 2J8G 3GBX 3CB9 2P5Y 2AGS 1N1Y 1GBJ 1GPQ 1X1R 3KKQ 3IXF
I159	1AB0 1LIC 1LID 1LQA 2Q91 3C1V
I192	3CNV
I279	1O8V

6.6.3 *Distribution of Matches*

Although different theozymes can sometimes be accommodated by the same scaffold, as seen in Table 6.6, these "overlaps" are sparse. This supports the conclusion reached at the end of section 6.5: A diverse set of theozyme geometries is indeed beneficial for the matching process.

The distribution of matches, given in Table 6.7, suggests an affinity of theozymes for certain CATH[143]-architectures (see section 5.3.1 and Table 5.1 for a more detailed explanation). I015, for example, produces three matches in the 2- and 3-layer sandwiches (3.30, 3.40) and zero to one for the remaining scaffolds. Whether this preferences are due to recurring geometric motifs of the proteins capable of hosting the novel active site or just random effects cannot be discerned clearly with the amount of data at hand.

Nevertheless, a correlation between theozyme and scaffold geometry is undeniable. Some of the theozymes exhibit residue arrangements more readily accommodated by peptide scaffolds, like I156, which produces 17 matches in different proteins. Others possess a structure that proves disadvantageous in the screening process. In case of G1_I38, the failure to find suitable scaffolds can be attributed to the spatial proximity of the non-aromatic residues' C_β atoms (Figure 6.21), requiring a very peculiar backbone conformation of the scaffold.

6.7 ENZYME DESIGN

The final step in the creation of a novel enzyme is a fine tuning of the matches via sequence design. This procedure is done in order to provide optimal geometries and environments for the active site

Table 6.7: Matches grouped by theozyme (columns) and CATH architecture identifiers of the scaffolds (rows), introduced in Table 5.1. The number in a field indicates, how many matches were found by ROSETTA-match for this particular scaffold/theozyme combination. A dash means no matches were found.

	G1_I38	I004	I015	I156	I159	I192	I279
1.10	-	-	1	1	2	-	-
2.40	-	1	1	1	3	-	1
2.60	-	-	1	4	-	-	-
3.10	-	-	-	-	-	-	-
3.20	-	1	-	2	1	-	-
3.30	-	-	3	-	-	-	-
3.40	-	-	3	6	-	1	-
3.90	-	-	1	3	-	-	-
Sum	0	2	10	17	6	1	1

and to increase the likelihood of the protein scaffold folding into the desired three-dimensional arrangement.

6.7.1 *Design Parameters*

The enzyme design utility provided by ROSETTA requires the specification of several parameters. These parameters are the four cutoff radii, the number of sequence design cycles, the ligand packing weight and whether to use the soft force field or not.

The cutoff radii are used to identify the residues allowed to mutate during the sequence design procedure (see section 5.3.2). The number of sequence design cycles specifies, how many iterations of alternating sequence design and energy minimisation are performed. The ligand packing weight is a factor for scaling the relative importance of substrate-protein interactions versus protein-protein interactions. The soft force field uses an alternative repulsive potential compared to the standard ROSETTA force field and is discussed in section 5.3.2. For a more detailed explanation of the different parameters, see appendix A.1.

As the inside-out protocol and the ROSETTA suite of programs are relatively young, no general rationale for setting these parameters exists yet. The influence of the cutoff radii on the design procedure is readily apparent: The more generous the values used, the more mutations will be present in the protein and vice versa. Since the presence of too many mutations may exert a disadvantageous influence on the folding behaviour of the scaffold and in turn its tertiary structure, it is normally better to choose conservative cutoff radii. Switching on the soft repulsive force field is usually beneficial, as it allows certain limitations of the traditional energy function to be overcome and may in turn lead to a better packing of the active site residues. Unfortunately, no guidelines exist for the ligand packer weight and the number of design cycles and optimal values are commonly found by trial and error. Standard values recommended by the Baker group are 6.0, 8.0,

10.0 and 12.0 Å for the cutoff radii, 4 cycles of sequence design and a ligand packer weight of 1.6.[150]

Different settings for the cutoff radii were tried in this study and the best results were obtained for cutoff radii almost identical to the ones suggested above, but with the first two radii (c_1 and c_2) reduced to slightly smaller values. The designs 3E3P and 3C9U (see below) were found using the radii $c_1 = 4.0$ Å and $c_2 = 6.0$ Å, while the other two designs 1O8V and 1JQ5 (see below) were obtained with $c_1 = 5.0$ Å and $c_2 = 6.0$ Å. This choice was made with regard to the experimental expression of the final designs, as it reduces the number of mutations which have to be introduced into the existing proteins. The soft repulsive force field was used in all cases as it provides better sidechain packing (see section 5.3.2). Since the last unconstrained repack step automatically uses the default ROSETTA force field instead of the soft repulsive one, the number of cycles was increased by one in order perform four full cycles of sequence design with the alternative force field.

6.7.2 *Design Runs*

The Monte-Carlo algorithm involved in the enzyme design stage is stochastic in its nature. Random choices regarding residue mutations and conformation sampling will be made during every individual run of the enzyme design program, yielding different results even if exactly the same initial conditions are used. It is therefore possible, that a certain theozyme-scaffold combination, obtained in the matching process, can result in a promising design in one case and only yield a suboptimal geometry in the other. In order to increase the chance of obtaining good enzyme designs, a large number of runs has to be performed. Here, 20 enzyme design runs were performed for every single one of the 32 theozyme-scaffold combinations listed in Table 6.6, resulting in a total of 640 different potential designs. The best of these enzyme designs, were then identified according to several criteria.

The scoring function provided by ROSETTA served as a general guideline, with the caveat that an iron atom had to be used instead of the central ruthenium atom for force field energies, as the ROSETTA force field is not parametrized for ruthenium. The distance between the chlorines and hydrogen bond donors was utilized as a criterion to filter out suboptimal designs. Designs with hydrogen-bond lengths significantly longer than the optimal length obtained through quantum-chemical calculations ($\sim$ 2.3 Å) were omitted during the later evaluation stages, as these designs are unlikely to exhibit the required coordination of the chlorine atoms. In a final visual inspection of the protein, the overall quality of the active site was assessed, using geometric reasoning and chemical intuition. The overall placement and accessibility of the active site was investigated, as well as the arrangement and orientation of the substrate and catalytic residues. Additional interactions between the substrate and adjacent residues were also considered. This procedure lead to the identification of four potential designs.

6.8 DESIGNED ENZYMES

The following section introduces the four enzymes yielded by the design procedure. The relative placement of the theozymes in the scaffolds, as well as the nature of the mutations will be discussed. A short overview will be given regarding the native function and active site configurations of the wild-type proteins. Throughout this section the naming scheme of the PDB database will be adopted, where every protein is identified by a four letter ID consisting of a digit, followed by three alphanumeric characters.

6.8.1 *Enzyme based on the Scaffold 1JQ5*

The following enzyme design was obtained by the introduction of the theozyme Io15 into the scaffold of 1JQ5 (Figure 6.28). The native protein is the glycerol dehydrogenase of *Bacillus stearothermophilus*, a thermophile bacterium. Its role is the oxidation of glycerol to dihydroacetone with the help of the coenzyme nicotinamide adenine dinucleotide (NAD+), serving as an alternative source of metabolic energy in anaerobic environments.[151]

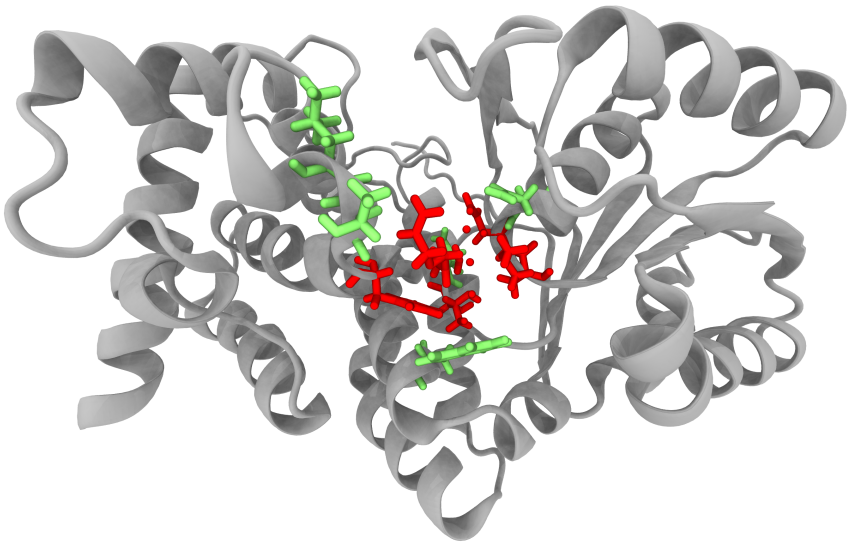


Figure 6.28: The designed enzyme combining the theozyme Io15 and the protein scaffold 1JQ5. The catalytic residues and the metal-substrate complex are shown in red, auxiliary mutations introduced by the sequence-design procedure are shown in green. The new active site is located in the cleft formed by the α-helical domain on the left and the β-sheet domain on the right.

The enzymatic structure can be divided into two domains separated by a cleft, which is at the same time the location of the active site and NAD+ binding. The first domain consists of a β-sheet framed by four α-helices and a single β-strand. The second domain is a helical bundle of 14 α-helices. Buried in the cleft lies the active site constituted of a zinc atom coordinated by the residues Asp173, His256 and His274. The

protein is currently believed to adopt a homooctameric conformation in solution.[152]

To accommodate the novel theozyme, a total of 9 mutations was introduced during the design procedure (Table 6.8).

Table 6.8: Mutations of the scaffold 1JQ5 introduced by ROSETTA.

Pos	Nat	→	Mut
95	Gly	→	Met
96	Gly	→	Asn
97	Lys	→	Asn
121	Ser	→	Thr
122	Thr	→	Asp
166	Leu	→	Trp
270	His	→	Leu
271	His	→	Thr
273	Thr	→	Phe

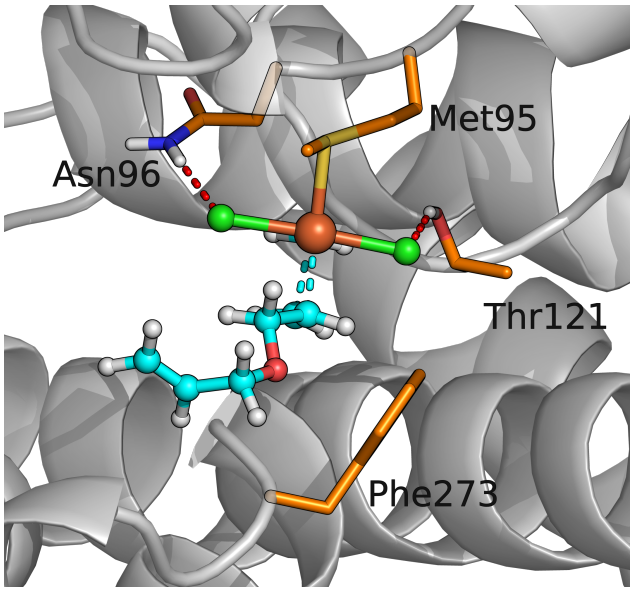


Figure 6.29: Close-up of theozyme I015 as it was crafted onto scaffold 1JQ5. Thr121 is located on a turn of medium length (10 residues), introducing uncertainty in the final arrangement of this amino acid. Phe273 adopts a conformation which may lower its beneficial influence on the RCM reaction.

The active site of the designed enzyme (Figure 6.29) lies in the NAD+ binding pocket of the wild-type, which undergoes major changes, whereas the zinc binding site (Asp173, His256 and His274) remains unmodified. The residues Gly96 and Lys97 originally bind the NAD+ via their backbone nitrogens and are both replaced by asparagines, which can potentially coordinate one of the chlorine atoms in the novel enzyme. The other hydrogen bond donor is introduced via a mutation of Ser121 to threonine. The serine sidechain coordinates to the NAD+ pyrophosphate oxygen atom in the native binding site. The amino acid Gly95, part of a glycine-rich turn (residues 94-96), gives way to the main methionine ligand of the ruthenium atom.

Thr273 is modified to provide the phenylalanine sidechain Phe273 for ether coordination. Unfortunately, Phe273 adopts a disadvantageous orientation when compared to the original theozyme I015 (Figure 6.25). Due to the backbone orientation of the phenylalanine residue, the suboptimal conformation cannot be amended through rearrangement. The change of Leu166 to tryptophane might additionally stabilize Phe273 through steric interactions. While the suboptimal orientation of Phe273 might have an negative effect on catalytic activity, this design was nevertheless considered, as the combination of all mutations results in an excellent positioning of the novel active site in the cleft between the two protein domains (Figure 6.30). Since this is also the location of the NAD+ binding pocket in the wild-type, it can be expected to provide a good balance between accessibility and shielding at the same time, compensating for some of the shortcomings of the design. The remaining mutations seem to play only a minor role.

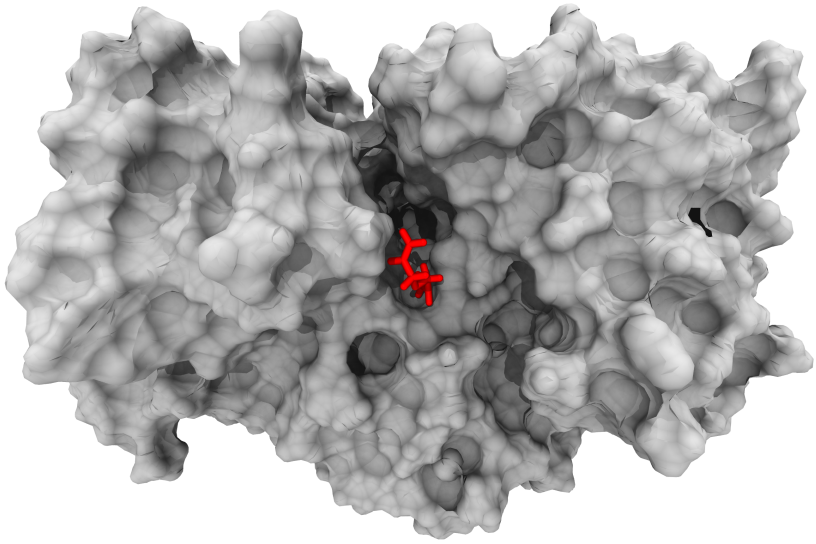


Figure 6.30: Solvent-accessible surface (SAS) of the modified protein scaffold 1JQ5. The diallylether substrate (red) is easily accommodated by the cleft, which provides substantial shielding from environmental influences.

The bulk of the important catalytic contacts of the theozyme is situated on well-preserved structural motifs, which are also described well by the ROSETTA algorithm.[153] The only exception is Thr121. The turn bearing this residue connects two β-strands and is of considerable length (residues 118-128). It may thus be subject to major conformational changes during the protein folding process, weakening or even destroying the hydrogen bond to the chlorine atom. However, this rearrangement does not necessarily have to take place and the potential risk is easily offset by the excellent location of the active site.

6.8.2 *Enzyme based on the Scaffold 3E3P*

This enzyme is a combination of the scaffold 3E3P and the theozyme I015 (Figure 6.31). The native protein is the glycogen synthase kinase-3

of *Leishmania major* and uses the coenzyme adenosine-triphosphate (ATP).[154] This *protozoa* is one of the pathogens associated with the parasitic disease leishmaniasis, making the protein a primary target for anti-parasitic drugs.

The 353 residues of the wild-type protein form two domains, a motif encountered on a regular basis in kinase enzymes.[155] A β-sheet framed by two α-helices constitute one domain, the other one is formed by a bundle of 17 α-helices. The native active site and the ATP-binding site are located in the fold formed by the two domains. A glycine rich loop serves as a kind of lid.

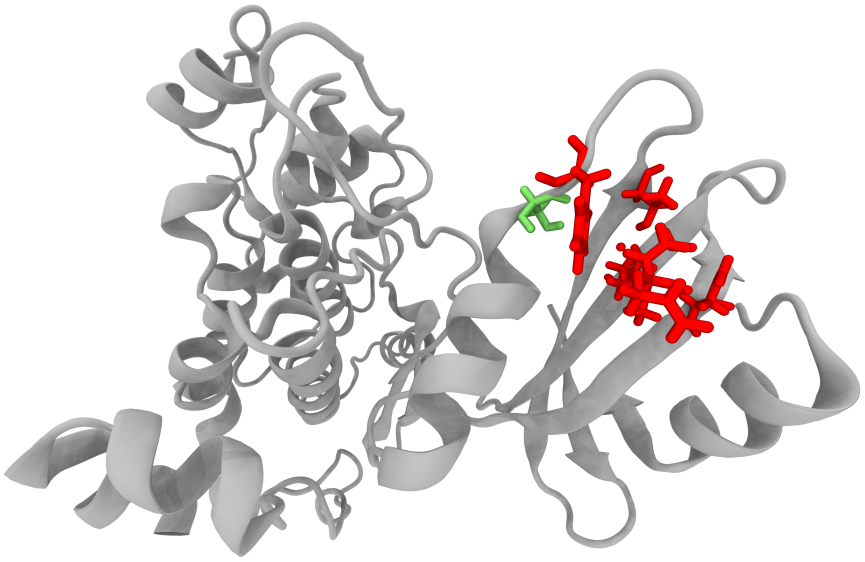


Figure 6.31: Protein scaffold 3E3P bearing theozyme I015. The novel active site (red) and the mutations introduced by ROSETTA (green) are all situated on or near the β-sheet of the right domain. The α-helical domain is shown on the left.

The mutations introduced by ROSETTA in order to accommodate the theozyme are listed in Table 6.9.

Table 6.9: Mutations of the 3E3P-scaffold in the designed enzyme.

Pos	Nat	→	Mut
53	Gln	→	Thr
58	Arg	→	Phe
59	Asn	→	Ser
84	Tyr	→	Asn
96	Leu	→	Met

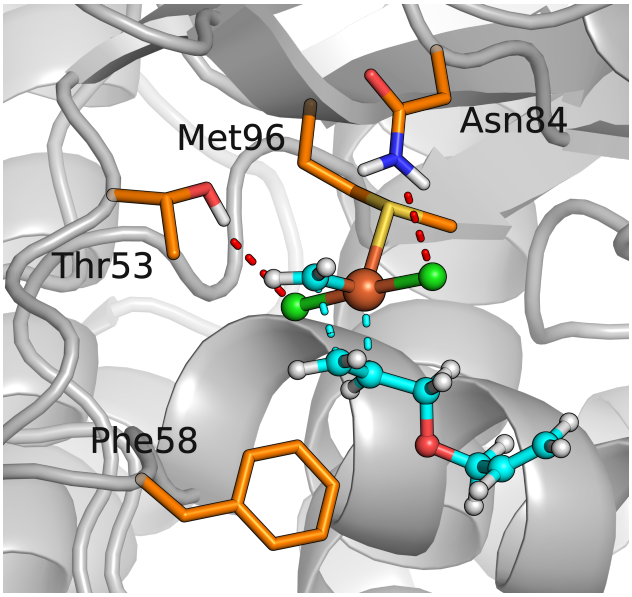


Figure 6.32: Novel active site of the 3E3P mutant based on theozyme I015. The catalytic sidechain Met96 is located directly on an α-helix, while the remaining residues are situated in close vicinity of highly preserved motifs. The suboptimal conformation of Phe58 relative to the coordinated ether double bond can be amended through orientation.

Unlike 1JQ5, the theozyme is crafted onto the β-sheet of the first domain (Figure 6.32), far off the native active site and almost opposite to the glycine rich loop, leaving the native active site completely intact. While the conservation of the wild-type catalytic center should exert no negative influence on the catalytic activity of the design, the increased accessibility accompanying the change in location might be a problem. Out of the mutated residues, three are located directly on the β-sheet, one on each separate strand. These residues are Met96, the main ruthenium ligand, and Thr53 and Asn84, the two residues coordinating the chlorines. The phenylalanine Phe58 facilitating the ether arrangement is introduced at the position of Arg58. This phenylalanine sidechain can be seen adopting a suboptimal arrangement when compared to the original theozyme, but the spatial orientation of its C_α-C_β bond allows switching to the optimal conformation at least in principle. It is situated on the turn connecting the β-sheet to one of the framing α-helices, along with the last mutated residue Ser58. This location is close to the end of the α-helix and should therefore be subject to only minor conformational changes.

The placement of the catalytic mutations on highly preserved protein motifs – such as the β-sheets – in general, should prove beneficial, as they are more likely to keep their native tertiary structure.

As stated above, the overall location of the novel active site (Figure 6.33) has both advantageous and disadvantageous. On the one hand, it lies completely open, which would facilitate the introduction of the ruthenium-alkylidene, as well as the association of the diallylether. On the other hand, this accessibility might also introduce disadvantageous interactions with environmental influences, such as solvent molecules.

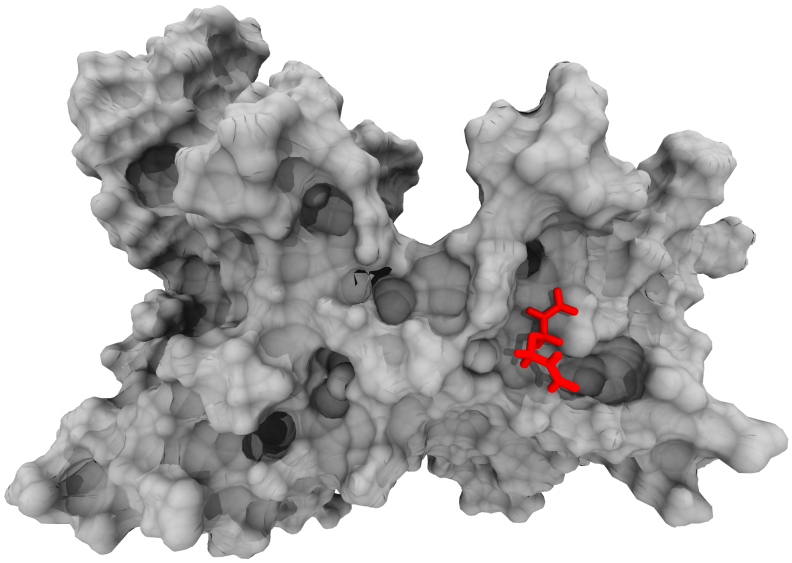


Figure 6.33: SAS of the scaffold 3E3P with the substrate shown in red. The relative exposure of the novel active site is easily noticeable.

Moreover, only a relatively small part of the protein is required for catalytic action, the other domain remains completely unused (Figure 6.31), as opposed to the native enzyme, where residues from both domains contribute to the active site situated in the inter-domain cleft. While it would be possible to sever the linker (residues 102 to 106) between the domains and only utilize the theozyme bearing part of the protein, it is difficult to predict to what extent the missing of the helical domain would influence the three dimensional structure of the fragment. Removing this domain could for example lead to distortions in the novel active site, destroying its catalytic geometry. Molecular mechanics and dynamics studies would be required to address this problem.

6.8.3 *Enzyme based on the Scaffold 1O8V*

The combination of theozyme I279 and the protein 1O8V yielded another promising candidate. The scaffold is the fatty-acid-binding protein 1 of *Echinococcus granulosus*, the dog tapeworm and cause of the echinococcal disease.[156] It differs from the other proteins insofar, as it possesses no inherent catalytic activity. Research points to a role as a transport-protein for lipids, as the parasitic organism is incapable of synthesising most of its required lipids on its own.

This peculiar function is reflected in the form of the protein. Two β-sheets form a barrel-like structure, consisting of 10 strands in total. The first strand exhibits a slight kink, allowing it to participate in both sheets. Strands 4 and 5 are separated by an inter-strand gap filled by highly ordered water molecules. The barrel is topped at one end by two α-helices, a region that is commonly accepted to be the entry point for the fatty-acids. The opposite end of the barrel, which is closed by a short 3_{10}-helix (a secondary structure motif similar to an

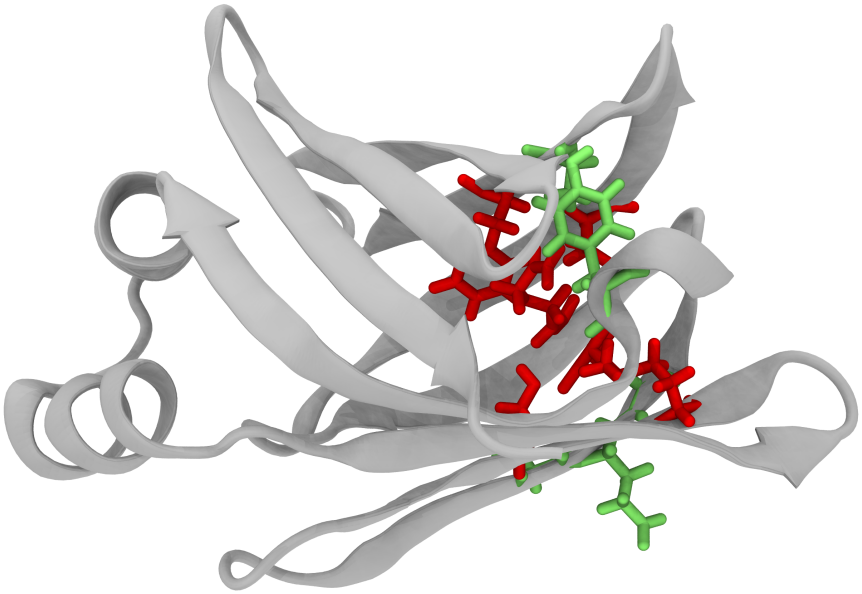


Figure 6.34: The protein scaffold 1O8V bearing the novel active site I279 (red). Additional mutations are coloured green. The barrel-like form of the scaffold is readily noticeable. On the left side of the protein, the two α-helices forming the entry portal to the native fatty-acid binding site can be recognized. The new active site is introduced on the other end of the barrel.

Table 6.10: Mutations required for the scaffold 1O8V to bear the theozyme functionality.

Pos	Nat	→	Mut
4	Phe	→	Tyr
50	Tyr	→	Met
52	Met	→	Ser
63	Cys	→	Ser
64	Ser	→	Glu
65	Phe	→	Gly
85	Ile	→	Thr
92	Met	→	Ala
94	His	→	Phe

α-helix but with three residues per turn instead of four), possesses no known purpose in the wild type. Buried in the cavity near the two α-helices, lies the Arg107-Tyr129-Arg127 triad, responsible for binding the carboxylate-group of the fatty acid. One other important motif is the presence of a "structural" water molecule, forming hydrogen bonds to the carbonyl oxygens of Lys66 and Glu69, as well as the amide nitrogen of Ile85. A final point of note is the oxidation state of Cys63, which is oxidised to its sulfenic acid analogon S-hydroxy cysteine. It is not clear whether this motif is inherent to the peptide scaffold or just an artefact due to the expression of the protein in *Escherichia coli*. The oxidation has no effect on the binding if the fatty-acids in the wild-type.

In this case, 9 mutations are performed by ROSETTA to accommodate the theozyme I279 into the scaffold (Table 6.10).

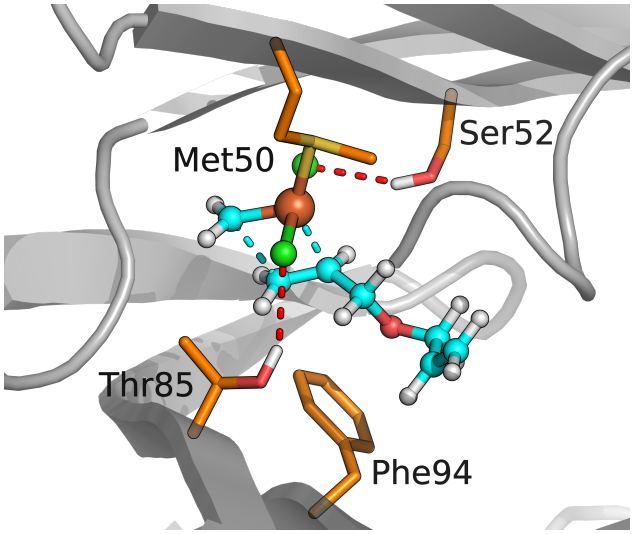


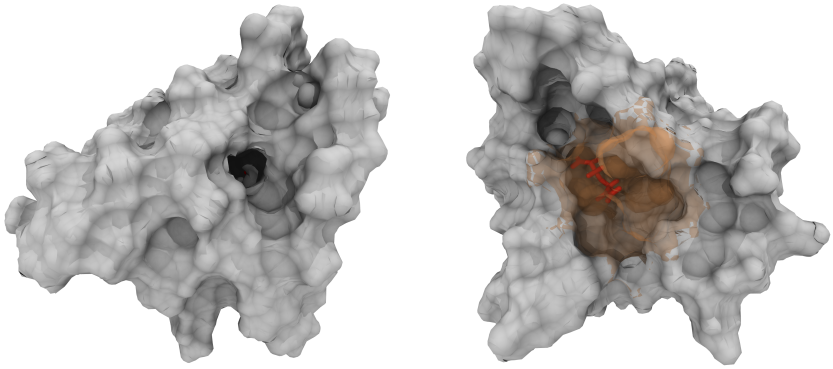
Figure 6.35: The novel active site based on theozyme I279 as introduced into scaffold 1O8V. The geometric arrangement of the catalytic sidechains should be subject to only minor conformational changes, as all residues are located on highly preserved β-sheets. Phe94 can be seen adopting a suboptimal arrangement, but the orientation of its C_α-C_β axis allows for rotation into the right conformation.

The theozyme is introduced inside the barrel region, opposite to the native portal site (Figure 6.35). Tyr50 is mutated to methionine, in order to serve as the main ligand for the ruthenium atom. To satisfy coordination of the ether double bond, phenylalanine replaces histidine at residue 50. Although it does not adopt the optimal orientation, this is only a minor drawback for the same reasons as in the case of design 3E3P. Ser52 acts as one of the chlorine ligands, the other is threonine at position of Ile85. As mentioned above, this residue is originally involved in coordination of the "structural" water. However, since the interaction partner of the water is the backbone amide nitrogen of Ile85 and not the sidechain, a change in the sidechain should have negligible influence on the structural integrity of this motif. The remaining mutations are of a more cosmetic nature. Substitution of Phe65 for a glycine creates space to accommodate the ruthenium atom. Mutation of Met92 to alanine minimises steric clashes between protein and substrate in a similar manner. Introduction of a tyrosine residue in place of Phe4 on the 3_{10}-helix may serve to stabilize the overall geometry of the barrel end, as it is perfectly positioned to form a hydrogen bond to the backbone carbonyl group of Val91. Orientation and placement of the mutated Ser63 suggests, that it might serve as an additional hydrogen-bond donor for the chlorine coordinated by Ser52. Residue Ser64 changes to a glutamic acid group facing outwards, a common motif in the barrel surface and beneficial for the water solubility of the protein. Finally the S-hydroxy cysteine at position 63 is exchanged for serine. The native residue does not seem to have a major influence on the tertiary structure of the scaffold and the same is to be expected from the mutation.

It should also be noted that the fatty-acid binding site remains almost unchanged, apart from a repacking of Arg107 to accommodate the theozyme, as the novel active site is introduced at the opposite end of the barrel.

As almost all mutations, especially the ones constituting the new catalytic site, are located on β-sheets, the structure of the designed enzyme is well defined and the correct three-dimensional geometry should be adopted during the folding process.

A problem with this design might be the overall form of the protein and the location of the active site, which are quite unusual for enzymes. This property could, however, also work in favour of the designed enzyme, as the scaffold interior provides a hydrophobic environment, optimal for metathesis reactions.



(a) (b)

Figure 6.36: SAS plots of the 1O8V-mutant. Image (a) shows the native fatty-acid portal site, while (b) shows the other end of the barrel, where the novel active site is situated. This active center is shielded completely by the protein, which might lead to issues regarding substrate association. Here the 3_{10}-helix (orange) is shown transparent in order to provide a view of the coordinated substrate.

There is also the issue of accessibility, as the active site is buried within one end of the barrel and the wild-type portal region is located on the opposite side (Figure 6.36). Whether the substrate can enter the enzyme on this alternative entry site remains to be seen.

6.8.4 *Enzyme based on the Scaffold 3C9U*

This designed enzyme is composed of the I015 theozyme and the protein scaffold of 3C9U (Figure 6.37). The wild type protein is the thiamin monophosphate kinase of *Aquifex aeolicus*, a thermophilic bacterium.[157] It originally catalyses the ATP-dependent phosphorylation of thiamin monophosphate.

The enzyme forms a homodimer, with two β-sheets as the interface, shown in Figure 6.38. This interface region is mainly stabilized by hydrophobic interactions of the involved residues.

The structure of the 309 residue monomer can be divided into two α/β-folds, separated by a cleft bearing the binding site of the substrates as well as the catalytic center. An important motif of

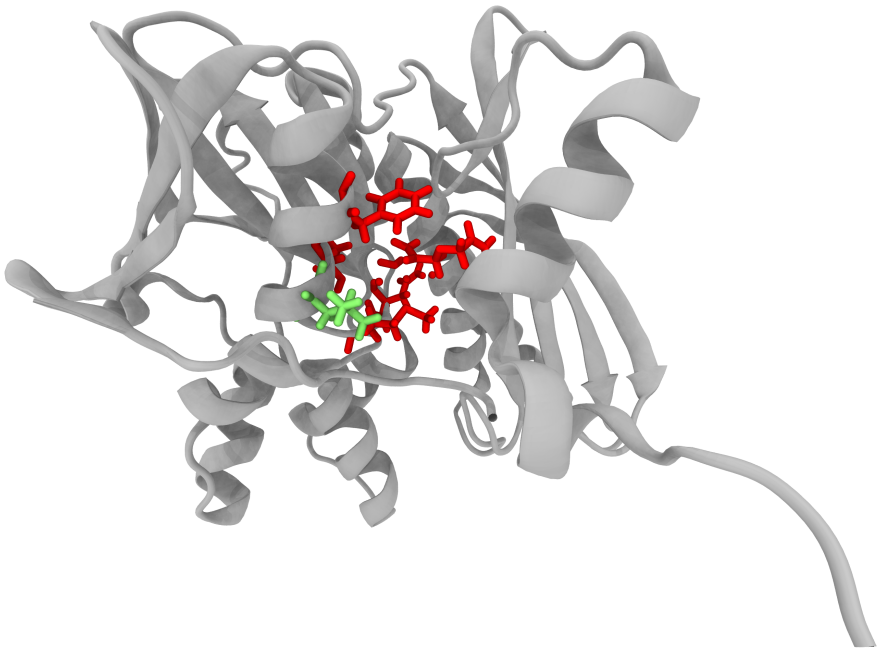


Figure 6.37: Mutant of scaffold 3C9U with theozyme I015 shown in red. Stabilizing mutations introduced by ROSETTA are shown in green. The novel active site is located in the inter-domain gap.

the original enzyme are five magnesium atoms situated in this cleft, typically coordinated by aspartate residues. Three of the metal atoms form a cluster and play a role in substrate coordination, while the other two are involved in the phosphorylation reaction.

A total of five mutations during the sequence design procedure lead to the mutant variant capable of hosting the theozyme, presented here. The list of mutations is given in Table 6.11.

Table 6.11: Mutations required to accommodate theozyme I015 into the protein scaffold 3C9U.

Pos	Nat	→	Mut
209	Ser	→	Asn
210	Asp	→	Met
214	Ala	→	Leu
215	Asp	→	Thr
218	His	→	Phe

The theozyme is introduced at the location of the active site in the wild-type (Figure 6.39). Out of the mutated residues, Ser209 and Asp210 are of immediate importance for the native active site. Ser209 forms a hydrogen bond to the β-phosphate group of the reactant thiamin pyrophosphate, while Asp210 coordinates one of the magnesium atoms involved in substrate binding. In the designed enzyme Ser209 was replaced by asparagine, which coordinates one of the chlorines of the central complex. Asp210 was mutated to a methionine residue, the main ligand of the ruthenium atom. This leads to the ruthenium

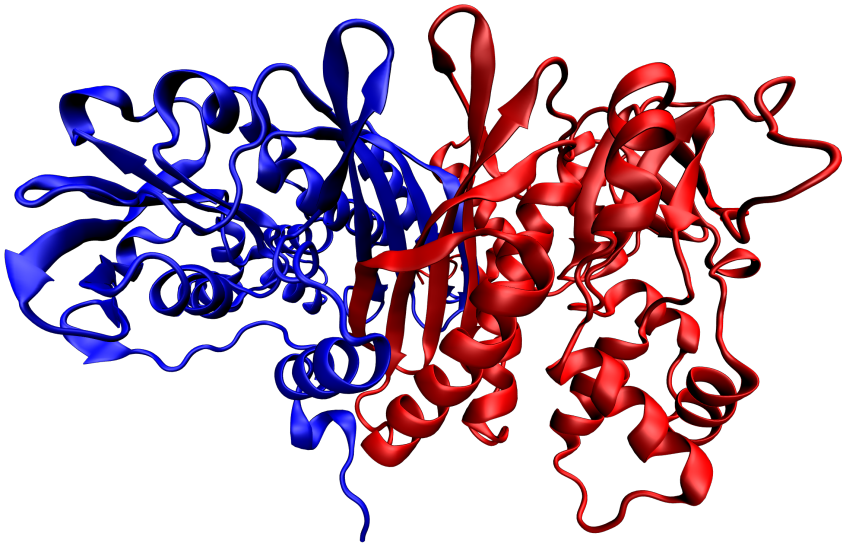


Figure 6.38: Homodimer formed by two monomers of 3C9U (red and blue). The interface region consisting of two β-sheets can be seen in the center of the dimer.

atom adopting a position approximately 4 Å from the magnesium cluster of the wild type, effectively replacing one of the magnesium atoms involved in the phosphorylation reaction. This placement is likely to work in favour of the design, as the local environment, which is still similar to the native magnesium binding site, should promote coordination of the ruthenium moiety. The geometry of the remaining four metal binding sites remains essentially unchanged by the design procedure, but the binding of magnesium atoms should be inhibited, due to steric and electronic effects exerted by the sidechain Ser209, introduced in close vicinity to the native magnesium coordination sites. Both mutations are located on the turn connecting β-strand $\beta 7$ to the α-helix $\alpha 8$ (for the nomenclature adopted here, see reference 157). These motifs are well-preserved and should therefore not be subject to significant conformational changes during the folding process, stabilizing the theozyme geometry. The helix $\alpha 8$ is also the region where the rest of the mutations are situated. Substitution of Asp215 to threonine allows coordination of the second chlorine. The phenylalanine required for double bond stabilisation is introduced at the location of His218. The mutation of residue 214 from alanine to leucine helps to improve the packing of the amino acid residues in its vicinity. Of interest is also the positioning of the unmutated residue Phe9, which would allow for the additional coordination of one of the terminal ether-double bonds.

The placement of the theozyme in the original active site should prove beneficial for the binding of the diallylether-substrate, as the good accessibility of the native scaffold is preserved (Figure 6.40).

Moreover, all mutations take place in regions usually well described by the ROSETTA-algorithm.[153] The backbone of the $\alpha 8$-helix can be expected to retain its conformation even with the designed residues present. The same holds for the turn bearing the two remaining

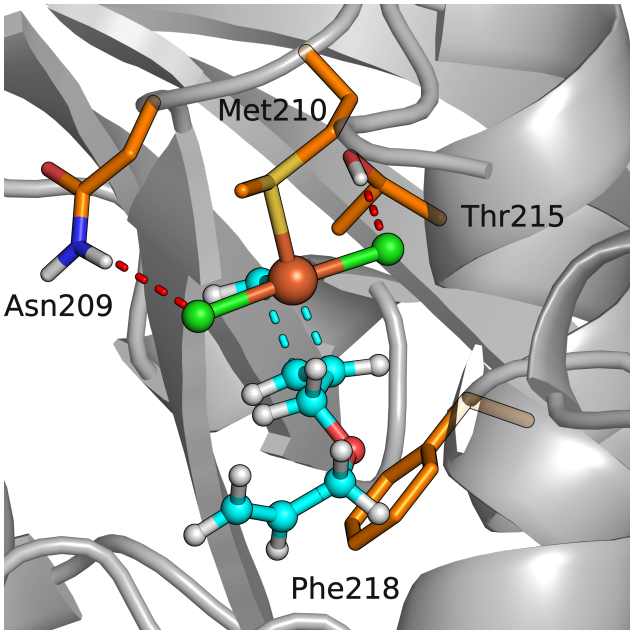


Figure 6.39: Close up of the novel active site based on I015 in the 3C9U scaffold. The chlorines are coordinated by Thr215 and Asn209, while Met210 provides the main ruthenium ligand. Both Phe218 and Thr215 are situated on the same α-helix, a motif that should help to stabilize their steric arrangement. The same is the case for Asn209 and Met210, which are located on a short turn connecting a β-sheet to a helix.

catalytic amino acids, as it is composed of only three residues and thus possesses only a very limited degree of conformational freedom.

One last note regarding the formation of the dimer: Since the dimerisation of two monomers is usually facilitated by hydrophobic interactions, mutations in the interface region could lead to a destabilisation of the protein-complex. This is fortunately not the case for this particular design and the monomers can thus be expected to dimerise.

6.9 COMPARISON OF THE DESIGNS

Out of the four designs, 1JQ5 seems to be the most unfavourable. This is because the unfortunate positioning of the hydrogen bond donor threonine on a turn introduces great uncertainty in the final theozyme geometry. There is also the issue of the phenylalanine orientation. The main advantage of the protein is the accessibility of the active site, which is probably unparalleled by the other *de-novo* enzymes.

The active site in 3E3P is situated far away from the one of the native enzyme. While this placement might not be inauspicious by itself, it leads to exposure of the central ruthenium complex, which might be a disadvantage. Another problem is the twisted phenylalanine ligand. However, as noted before, the increased accessibility is not necessarily only a negative trait and the orientation of Phe58 can easily change.

The greatest drawback of design 1O8V is probably the missing enzymatic activity of the wild-type. This makes it hard to discern whether the protein is even capable of bearing a catalytic site. The accessibility is only a minor potential problem, as fatty acids seem

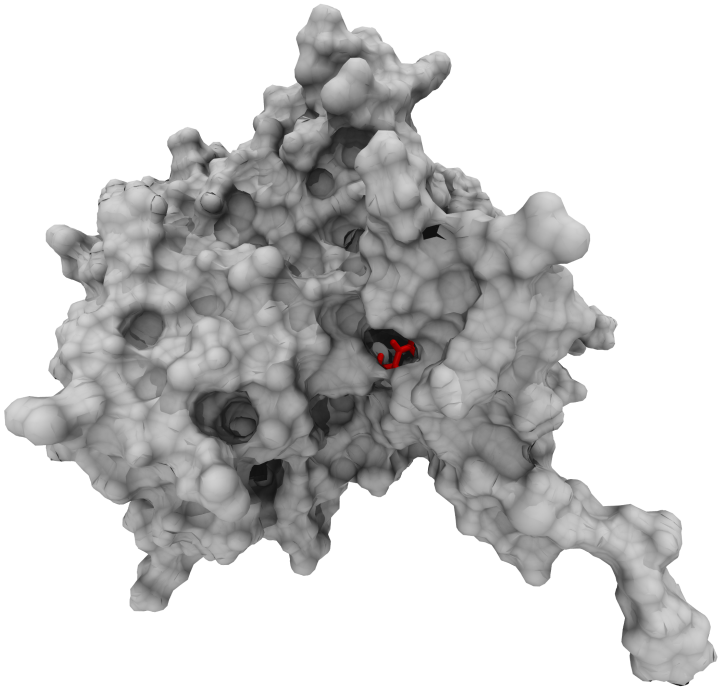


Figure 6.40: SAS of the 3C9U mutant. The scaffold geometry provides good shielding from environmental influences, while still allowing access of the substrate (red).

to be capable of penetrating deep into the protein. The case of the suboptimally arranged phenylalanine is similar to the one in 3E3P. The great advantage of the mutant 1O8V is its geometry. Every catalytic residue is positioned on a highly conserved structural motif. Moreover, the general makeup of the whole protein suggests, that it will very likely fold into the required tertiary structure. The form of a barrel additionally provides a very unique environment for the enzymatic reaction, a trait not found in the other designs.

The design 3C9U seems to combine several good traits of the other *de-novo* enzymes. The theozyme geometry is very close to optimal and all catalytic residues reside on the backbones of highly preserved structural motifs, as is the case for 1O8V. The active site location in the inter-domain cleft, similar to 1JQ5, is also highly desirable.

The combination of all these traits makes 3C9U probably the most promising one of all the obtained enzymes.

While the created enzymes show potential, it is prudent to remember that all results presented here are ultimately theoretical in nature. This means approximations and simplifications were involved in their derivation and certain aspects were simply neglected. The reason is the extreme complexity of enzymatic systems, which makes a detailed modelling of all the interactions and effects an impossible task. Whether the designs are capable of catalysing olefin metathesis in reality can only be ascertained through experiment.

7

SUMMARY

The aim of this thesis was the computational *de-novo* design of a metalloenzyme capable of catalysing the olefin RCM reaction of diallylether. To this end the inside-out design protocol[22] was applied.

A genetic algorithm was newly developed for the creation of different theozymes, combining high-throughput DFT computations for fitness evaluation with a stochastic optimisation procedure. This approach was able to generate active site templates with significantly lowered activation energies of the RCM reaction, even when compared to highly active conventional metathesis catalysts, making the algorithm an extremely valuable tool in the creation of novel active site geometries. As an additional feature, the algorithm exhibits the typical robustness of this class of evolutionary algorithms, requiring almost no information on the reaction to be modelled, as opposed to the classical approach to theozyme creation, which makes excessive use of chemical intuition. This property allows to augment the standard inside-out protocol with the genetic algorithm, thus closing in on an universal black-box procedure for *de-novo* enzyme design.

In a subsequent step, the matching procedure of the Rosetta program package was used to search a protein scaffold library for sites capable of accommodating the theozymes obtained with the genetic algorithm. The resulting "raw" matches were then subjected to several cycles of sequence design in order to optimise the catalytic interactions and substrate binding. Out of the 640 potential designs obtained in this way, the best were chosen according to geometric and chemical criteria. This procedure culminated in the *de-novo* design of 4 promising metalloenzymes on the basis of the protein scaffolds 1JQ5, 3E3P, 1O8V and 3C9U. All of these designs exhibit the traits associated with highly active metathesis catalysts, as derived earlier in this work by quantum chemistry. The most likely candidate for an active enzyme is the design based on 3C9U, due to the excellent geometry and positioning of its active site. While the created enzymes seem promising, it is prudent to remember that all results presented here are ultimately theoretical in nature. This means approximations and simplifications were involved in their derivation. The reason is the extreme complexity of enzymatic systems, which makes a detailed modelling of all the interactions and effects a very challenging task. Whether the designs are capable of catalysing olefin metathesis in reality can only be ascertained through experiments, which will be carried out by the group of Univ.-Prof. Dr. C. Becker.

If metathesis activity of these enzyme designs were to be confirmed by experiment, exciting new possibilities would open up in industrial applications, green chemistry and organic synthesis. With these advances at hand, goals such as the catalysis of reactions under mild conditions, the cheap mass synthesis of drugs and the disclosure of novel reaction pathways and materials, will be easier to reach in the future.

8

OUTLOOK

As stated in the introduction, the computational *de-novo* design of enzymes is a relatively young field of research. Due to the intrinsic complexity of enzymatic systems and reactions, current approaches to enzyme design involve a multitude of approximations in order to render the task possible at all, leaving room for many improvements, targeting either the designed enzymes themselves or the protocol and methods employed in their creation.

The information gained through the experimental expression and characterisation of the four designed enzymes (1JQ5, 3E3P, 1O8V and 3C9U) with regards to their metathesis activities and crystal structures can be utilized to enhance even only minor metathesis activity possibly present in any of the designs, leading to more reactive enzymes. One potential – purely experimental – method to this end is directed evolution[6], where steps of mutation and screening for enzymes with increased activity are alternated *in vitro*. This approach is a powerful tool to enhance enzyme reactivity, as demonstrated in the design of the Kemp eliminase enzymes.[25] From a theoretical side, molecular dynamics (MD) simulations, as well as quantum mechanics/molecular mechanics (QM/MM) computations on the crystal structures of the four designs could provide detailed insights into the geometry of the newly introduced active sites and the dynamic behaviour of the protein scaffolds, which could in turn be used to tweak the designs. It is also possible to combine theoretical and experimental approaches in order to enhance the designed enzymes by applying iterative cycles of experimental expression and theoretical analysis, utilizing molecular dynamics MD studies in a manner similar to the protocol described in reference 158, where X-ray crystallography and MD data are used to continuously improve upon previous designs.

In addition to these refinements based on experimental results, several improvements can be made to each of the three different stages of the inside-out design protocol.

The genetic algorithm facilitating theozyme creation, for example, can be enhanced by modifying the fitness evaluation procedure, as currently just one reaction barrier of the RCM reaction is considered. The algorithm is therefore only capable of optimising a small part of the total reaction landscape. Introducing a composite fitness function incorporating several reaction barriers or switching to a multi-objective optimisation procedure[159] would allow a more fine-tuned engineering of the reaction profile. However, even with a refined modelling of the fitness landscape, the performance of the stochastic optimisation still depends on the quality of the quantum chemical computations. While the DFT method used in this work allows for an efficient evaluation of the RCM energy barriers, there are several limitations to the accuracy of this method (see section 4.3.6). Potential alternatives are recently developed computational approaches based on *ab initio* principles, such as the domain-based local pair natural orbital-coupled-cluster

method[160] and the density matrix renormalisation group method.[161] These electronic structure methods would also profit from a procedure often used for genetic algorithms involving expensive fitness functions, the so-called fitness approximation.[162] In this approach, a cheap fitness function, e.g. based on QSAR studies[149] or artificial neural networks[163], is used to screen the potential solutions for the most promising candidates, which are then evaluated using the more expensive but accurate fitness evaluation protocol. To further improve the quality of fitness computations, QM/MM energy calculations could be performed during this stage, thus recovering the electrostatic influences exerted by the protein environment and second shell amino acids, which are currently ignored. The coupling of fitness evaluation and the matching process required to obtain the scaffolds necessary for such a QM/MM treatment could at the same time provide measures, how well the geometry of an energetically feasible theozyme can be accommodated by a protein scaffold – e.g. the number of matches found for a particular theozyme –, which can then be incorporated in the computation of its fitness.

Regarding the matching procedure, one potential improvement would be the introduction of backbone flexibility during the matching process, since protein scaffolds are no rigid entities.[164,165]

In the case of the sequence design stage of the inside-out protocol, one important issue to address is, whether the metal moiety and the substrate will coordinate to the newly designed active site or if there are concurring binding sites. This potential shortcoming of designed enzymes could be amended by actively applying a negative design procedure, aimed to systematically destabilize these alternative binding sites, thus increasing the likelihood of the cofactor and the substrate binding in the desired way.[166] This negative design approach, along with the general sequence design procedure could both profit tremendously by switching from the stochastic Monte-Carlo approach to deterministic dead end elimination[21], leading to enhanced reliability and reproducibility of the sequence/structure optimisation process. Unfortunately, the presence of a minimum energy sequence provides no measure for the reactivity of the design. Therefore, the introduction of reliable protocols to discern between potentially active and inactive designs would also be of advantage. MD studies have been shown to be of great help for assessing the quality of the designed active sites, as they allow to observe the integrity of the theozyme geometry with respect to the motions of the protein scaffold.[38]

All the improvements suggested above are only a few examples of potential upgrades to the inside-out design protocol. Unfortunately, this approach itself is essentially limited in the novel chemistry it can produce, as it relies on existing protein scaffolds as building blocks. And while an incredible amount of diverse proteins exists, it is unlikely that a matching scaffold will be found for every potential theozyme. The ultimate goal of computational *de-novo* enzyme is the complete design of a protein scaffold with novel catalytic activity. This task is indeed formidable, as the inverse folding problem is one of the great unresolved issues of structural biology and although promising results are reported occasionally[153], the research on this topic is still in its infancy.

APPENDIX

A.1 ENZYME DESIGN PARAMETERS AND SETTING

In this section the different parameters and settings used by ROSETTA during the final sequence design procedure (see section 5.3) will be discussed. A sample input file is given in Figure A.1. For a detailed explanation of the different parameters on the basis of an example design procedure, see reference 150.

```
-extra_res_fa THZ.params
-enzdes::cstfile I004.cst
-enzdes::detect_design_interface
-enzdes:cut1  6.0
-enzdes:cut2  8.0
-enzdes:cut3 10.0
-enzdes:cut4 12.0
-enzdes:cst_opt
-enzdes:bb_min
-enzdes:chi_min
-enzdes:cst_design
-enzdes:design_min_cycles 4
-enzdes:lig_packer_weight 1.60
-packing:soft_rep_design
-enzdes:cst_min
-enzdes:bb_min
-enzdes:chi_min
-packing:ex1
-packing:ex2
-packing:use_input_sc
-packing:linmem_ig 10
-enzdes:cst_min
-enzdes:bb_min
-enzdes:chi_min
-packing:ex1
-packing:ex2
-packing:use_input_sc
-out:file:o ranking.log
```

Figure A.1: Example input file used for ROSETTA during the final enzyme design stage.

The first two entries serve to specify the parameter file of the central substrate (`THZ.params`) and the theozymatic constraint file, in this case for geometry Ioo4 (`I004.cst`).

In order to specify which residues are to be designed or optimised by the algorithm, the amino acids in vicinity of the active site have to be grouped into the categories catalytic, designable and repackable. This is facilitated by the `detect_design_interface` command using the four cutoff radii `cut1-4` given in the subsequent lines. Residues with their C_α within `cut_1` or within `cut_2` and their C_β closer to

the substrate are set to be designable. In a same way, the repackable residues are chosen according to `cut3` and `cut_4`. The catalytic residues are read in from the geometric constraint file.

The initial optimisation of the minimal active site is invoked with `cst_opt`. The keywords `bb_min` and `chi_min` allow for backbone flexibility and small changes in the dihedral angles of the catalytic residues during energy minimisation.

`cst_design` starts the iterative sequence design procedure. The number of cycles is given by `design_min_cycles`. The relative importance of substrate-protein interactions versus protein-protein interactions can be scaled via `lig_packer_weight`. The keyword `soft_rep_design` activates the use of an alternative force field making use of a shorter van-der-Waals repulsive term, which allows for a more fine-grained sampling of conformation space. The optimisation following each sequence design step is invoked by `cst_min`.

The use of `ex1`, `ex2` and `use_input_sc` is optional, but it improves the behaviour of the algorithm when sampling the amino acid sidechain rotamers. The option `linmem_ig` helps to improve memory requirements and performance during the design stage.

The final unconstrained optimisation is invoked automatically once the sequence design cycles have finished. It makes use of several keywords already encountered in the previous steps. The last line of the input specifies the file to which the different ranking criteria are written.

A.2 SOURCE CODE OF THE STEADY-STATE GENETIC ALGORITHM

This section contains the source code for the steady-state genetic algorithm used in the creation of the RCM theozymes. In order to increase the efficiency of arithmetic and array operations, the Numerical Python (numpy) package[167] was used. The source code is given in Listing 1.

Listing 1: Source code of the steady-state genetic algorithm used for the creation and optimisation of theozymes.

```
#!/usr/bin/python -uvd

import numpy as np
from numpy import *
import datetime
from random import choice
import subprocess
import fileinput
import multiprocessing
from multiprocessing import Process, Manager, Queue
import pickle
import copy
import os
import sys

##################################################
# Alternate take on the genetic algorithm using
# a steady state implementation, since this helps
# to keep idle time a minimum
##################################################

#**************************************************
```

```
# GLOBAL VARIABLES
#************************************************

#----------Processors--------------
nproc = 4                           # use nproc cores for
    population creation

#------------Paths-----------------
# Get current working directory:
WDIR = os.getcwd()
# Build necessary paths:
scr_path = WDIR + '/scripts/'       # scripts
lib_path = WDIR + '/res_lib/'       # molecule residues and
    templates
# Build output paths:
out_path = WDIR + '/tt_geometries/'  # initial geometries
qm_path = WDIR + '/tt_qm_comp/'      # repository for QM
    computations
log_path = WDIR + '/tt_log_files/'   # holds logs


#-------Geometry Creation---------
coll_limit = 4                      # maximum of sqrt 4
    Angstrom between residues
lim_orient = 0.5*pi                 # allowed backbone
    orientation
# Cylinder from which coordinates are to be sampled (center
     atom lies as origin):
h_cyl_up = 6.0
h_cyl_lo = 7.0
r_min = 3.0
r_max = 7.0

#---------Genome Creation----------
# length of alleles:
al_min = 1
al_max = 2
# core-templates used:
template_a = 'nolig_template_a'          # geometry of first
     point
template_b = 'nolig_template_b'          # second point
    geometry
# list of residues to be chosen from:
#l_resid = [ 'arg', 'asn', 'asp', 'cys', 'gln', 'glu', 'his
    ', 'ile', 'leu', 'lys', 'met', 'phe', 'ser', 'thr', '
    trp', 'tyr', 'val', 'sec', 'hsp', 'hse', 'cyt' ]
l_resid = [ 'gln', 'asn', 'met', 'phe', 'ser', 'thr' ]

#------Population Parameters-------
no_eval = 500                       # maximum number of
    fitness evaluations to be performed
species_id = 300                    # index to keep track of
    each individual
popsize = 60                        # population size
p_threshold = 4                     # minimal population size
    , before genetic operators are applied
CR = 0.4                            # crossover-rate
n_sel = 2                           # number of parents used
    for crossover
n_offspring =  2                    # offspring created by
    mating
tournament_size = 4                 # size of tournament for
    selection, larger values bias towards the fittest
    individual
```

```
new_species = 1                    # species generated anew
    each cycle

#--------Restart Parameters--------
is_restart = 1                     # logical operator if
    restart from specified folder is to be done
restart_log = '2013-07-07-23:12.bin' # binary log-file used
     for restart

#----------QM Parameters-----------
qm_queue = 'lead.q'                # Queue used for
    computations
qm_max = popsize                   # maximum of jobs in
    queue at the same time, for convenience set to popsize
free_min = 4                       # minimum amount of free
    slots in queue before genetic operations are applied

# Unfreeze last n atoms of the template:
unfreeze_a = 12                    # first template
unfreeze_b = 5                     # second template

#---------To Be Initialized--------
charge = {}                        # will hold charges
res_size = {}                      # will hold size of
    residues
is_comp = {}                       # check if individual was
     already evaluated
to_comp = {}                       # individuals to be
    evaluated
pivots = {}                        # dictionary holding
    pivot carbon positions based on first geometry
elegible = {}                      # individuals elegible
    for selection
comp_time = {}                     # time required for
    evaluation of individual
population = []                    # will hold the members
    currently in the population
fitness = {}                       # dictionary containing
    the computed fitness values
dead = []                          # list of all individuals
     which died during evaluation
restart_points = {}                # dictionary holding the
    current progress of every computation


#*************************************************
# FUNCTIONS
#*************************************************

#=========================================
# Initialisation
#=========================================

#-----------------------------------
# Initialize charge and residue size;
# for this the second line of the xyz
# files needs to be of the format:
# "Charge= <CHARGE>"
#-----------------------------------
def init_charge_size():
    global charge
    global res_size
    files = [ f for f in os.listdir( lib_path ) if os.path.
        isfile( lib_path + f ) ]
```

```
    for f in files:
        name = f[:-4]                                # remove
            .xyz
        lines = open( lib_path + f, 'r' ).readlines()
        try:
            r_size = int( lines[0] )                 # get
                size
        except:
            print "File␣%s␣is␣no␣xyz-file." % f
            exit()
        try:
            r_charge = int( lines[1].split()[1] ) # get
                charge
        except:
            print '''In␣order␣to␣read␣in␣the␣charges␣
                automatically,␣the␣second␣line␣of
the␣xyz-file␣%s␣needs␣to␣be␣of␣the␣format:
Charge=␣<CHARGE>.''' % f
            exit()
        # update the dictionaries:
        res_size[name] = r_size
        charge[name] = r_charge


#=========================================
# Input and Output
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Read coordinates from xyz-file and
# return the number of atoms, a vector
# holding the atom types and an 3*n
# array holding the coordinates in
# angstrom.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def read_coords( filename ):
    try:
        file = open( filename, 'r' ).readlines()
    except:
        print "File␣%s␣opened␣for␣reading␣does␣not␣exist."
            %filename
        exit()
    natoms = int( file.pop(0) )                  # read
        number of atoms
    scrap = file.pop( 0 )                        # skip
        line holding the charge
    # initialize the arrays:
    atype = empty( ( natoms ), dtype='a2' )
    geom = empty( ( natoms, 3 ), dtype='f8' )
    # read in atom types and coordinates:
    for i in range( natoms ):
        columns = file[i].split()
        atype[i] =  columns[0]; geom[i]=[ float( columns[1]
            ), float( columns[2] ), float( columns[3] ) ]
    return natoms, atype, geom

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create and write xyz-file from atom-
# types and coordinates
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def write_xyz( natoms, atype, geom, filename ):
    outfile = open( filename, 'w' )
    outfile.write( "%s\n" % natoms )
    outfile.write( "\n" )
```

```
    for i in range( natoms ):
        outfile.write( '%s\t%10.7f\t%10.7f\t%10.7f\n' %(
            atype[i], geom[i,0], geom[i,1], geom[i,2] ) )
    outfile.close()

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create xyz-file directly from genome
# for a single species
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def write_species( species, population ):
    filename = out_path + species + '.xyz'
    natoms, atype, as_geom = cat_geom( population[species
        ][0], population[species][1], population[species
        ][2] )
    write_xyz( natoms, atype, as_geom, filename )

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Print a list of individuals (l_indi)
# and their residue sequences
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def print_genome( l_indi, canditates ):
    print '    Name\t\t Residues'
    print '  ---------------------------------------'
    for individual in l_indi:
        print '    %s\t\t%s' %( individual, ' '.join(
            canditates[individual][0] ) )
    print '  ---------------------------------------'
    print ''

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Print fitness of a list of
# individuals
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def print_fitness( l_indi ):
    print ' Current fitness:'
    print '    Name\t\t Fitness'
    print '  ---------------------------------------'
    for individual in l_indi:
        print '    %s\t\t%.3f' %( individual, fitness[
            individual] )
    print '  ---------------------------------------'
    print ''

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Test directories for existence and
# create if neccessary; print used
# directories to stdtout
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def print_folders():
    print ''
    if not os.path.exists( scr_path ):
        print ' Directory holding scripts could not be
            found.'
        exit()
    else:
        print ''' Scripts and define template will be taken
            from:
    %s''' % scr_path
    if not os.path.exists( lib_path ):
        print ' Directory containing residue geometries was
            not found.'
        exit()
    else:
```

```
        print ''' Residue geometries will be read in from:
    %s''' % lib_path
    if not os.path.exists( qm_path ):
        os.makedirs( qm_path )
    print ''' QM results will be written to:
    %s''' % qm_path
    if not os.path.exists( out_path ):
        os.makedirs( out_path )
    print ''' Generated geometries will be written to:
    %s''' % out_path
    if not os.path.exists( log_path ):
        os.makedirs( log_path )
    print ''' Log files will be written to:
    %s''' % log_path
    print ''

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Print those individuals to be
# evaluated in this cycle
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def print_evaluate( l_evaluate ):
    print '\n Structures to be evaluated:'
    counter = 1
    for key in l_evaluate:
        print ' ' + key + '\t',
        if counter%5 == 0:
            print '\n',
        counter += 1
    print ''

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Print list in a nice way
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def print_list( list ):
    counter = 1
    for key in list:
        print ' ' + key + '\t',
        if counter%4 == 0:
            print '\n',
        counter += 1
    print ''


#=========================================
# GEOMETRY CREATION
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Rotate residue geometry array by
# quaternions q passed as vector
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def rotate_res_rev( q, res_geom ):
    q0 = q[0]; q1 = q[1]; q2 = q[2]; q3 = q[3]
    roma = empty( (3,3) )                          #
        initialize rotation matrix
    # Create rotation matrix from quaternions:
    roma[0] = [ 1-2*q2*q2-2*q3*q3, 2*q1*q2-2*q0*q3, 2*q1*q3
        +2*q0*q2 ]
    roma[1] = [ 2*q2*q1+2*q0*q3, 1-2*q3*q3-2*q1*q1, 2*q2*q3
        -2*q0*q1 ]
    roma[2] = [ 2*q3*q1-2*q0*q2, 2*q3*q2+2*q0*q1, 1-2*q1*q1
        -2*q2*q2 ]
    roma = np.transpose( roma )                    #
        transpose because of array structure
```

```
    res_geom =  dot( res_geom, roma )            # perform
        rotation
    return res_geom

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Translate residue geometry by a
# displacement vector dr
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def translate_res( dr, res_geom ):
    dx = dr[0]; dy = dr[1]; dz = dr[2]
    res_geom[:,0] += dx
    res_geom[:,1] += dy
    res_geom[:,2] += dz
    return res_geom

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Perform brute force collision check
# on growing temporary structure. The
# minimum allowed distance is specified
# by coll_limit, 1 is returned if
# collision is detected; since this is
# done residue-wise it is more
# efficient than a collision check for
# a finished geometry
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def res_coll_brute_force( res_geom, tmp_geom ):
    range_i = range( len( res_geom ) )
    range_j = range( len( tmp_geom ) )
    for i in range_i:
        for j in range_j:
            diff = res_geom[i] - tmp_geom[j]     # compute
                connecting vector
            diff_sq = dot( diff, diff )          # compute
                distance
            if diff_sq < coll_limit:             # perform
                check
                return 1, tmp_geom
    return 0, tmp_geom

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# An alternative collision check is
# needed for offspring and mutations,
# where the geometries are not created
# from scratch
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def off_coll_brute_force( geom_list ):
    tmp_geom = coll_geom
    for geom in geom_list:
        range_i = range( len( geom ) )
        range_j = range( len( tmp_geom ) )
        for i in range_i:
           for j in range_j:
               diff = geom[i] - tmp_geom[j]
               diff_sq = dot( diff, diff )
               if diff_sq < coll_limit:
                   return 1
        tmp_geom = row_stack( ( tmp_geom, geom ) )
    return 0

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Yet another collision check where one
# can vary the minimum distance between
# atoms; needed for offspring, since
# the normal collision check can get
```

```
# (and will get) stuck because of the
# compact geometries
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def child_coll_brute_force( geom_list, coll_thres ):
    tmp_geom = coll_geom
    for geom in geom_list:
        range_i = range( len( geom ) )
        range_j = range( len( tmp_geom ) )
        for i in range_i:
           for j in range_j:
               diff = geom[i] - tmp_geom[j]
               diff_sq = dot( diff, diff )
               if diff_sq < coll_thres:
                   return 1
        tmp_geom = row_stack( ( tmp_geom, geom ) )
    return 0

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Check if the side-chain is oriented
# away from the central atom, since
# this would not result in a sensible
# geometry; for this the angle between
# the translation vector and the vecor
# from the pivot carbon to the origin
# is computed; if it is smaller than
# lim_orient 1 is returned.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def res_orientation( res, coords, res_geom ):
    v_trans = coords                                # coords
        holds the translation vector
    v_orient = res_geom[0] - v_trans                # compute
        other vector, first atom in
                                                    # geometry
                                                        is
                                                        assumed
                                                         to be
                                                        the
                                                        pivot
                                                        carbon
    # compute the angle
    theta = np.arccos( dot( v_trans, v_orient )  / ( linalg
        .norm( v_trans ) * linalg.norm( v_orient ) ) )
    if theta > lim_orient:                          # if
        threshold is reached, 1 is returned
        return 1
    return 0

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Sample random coordinates in a
# cylinder around the central atom
# specified by r_min, r_max, h_cyl_lo
# and h_cyl_up.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def create_coords():
    phi=random.uniform( -pi , pi )
    r = random.uniform( r_min, r_max )
    z = random.uniform( -1*h_cyl_lo, h_cyl_up )
    y = r*math.sin(phi)
    x = r*math.cos(phi)
    return [ x, y, z ]

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create a biased set of coordinates,
# where the first residue is positioned
```

```
# above the central ruthenium, the
# second and third amino acids are
# placed (at least in theory) near the
# chlorine atoms, and the fourth
# residue is placed beneath the complex
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def create_coords_bias( index ):
    # if it is the first amino acid, put it on top of
        complex
    if index == 0:
        phi=random.uniform( -pi , pi )
        r = random.uniform( 0, 3 )
        z = random.uniform( 3, h_cyl_up*0.7 )
    # if it is the second, place it near the missing
        chlorine
    elif index == 1:
        phi=random.uniform( -pi*0.2 , pi*0.2 )
        r = random.uniform( r_min, r_max )
        z = random.uniform( -0.3*h_cyl_lo, h_cyl_up*0.3 )
    # if it is the second, place it opposite
    elif index == 2:
        phi=random.uniform( -pi*0.2 + pi , pi*0.2 + pi )
        r = random.uniform( r_min, r_max )
        z = random.uniform( -0.3*h_cyl_lo, h_cyl_up*0.3 )
    # phe below
    elif index == 3:
        phi=random.uniform( -pi , pi )
        r = random.uniform( 0, 3 )
        z = random.uniform( 3, h_cyl_lo*-1.2 )
    # else place it at random
    else:
        phi=random.uniform( -pi , pi )
        r = random.uniform( r_min, r_max )
        z = random.uniform( -1*h_cyl_lo, h_cyl_up )
    y = r*math.sin(phi)
    x = r*math.cos(phi)
    return [ x, y, z ]

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Sample unit quaternions according to
# the scheme by Marsaglia.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def create_quat():
    while True:
        x1 = random.uniform( -1, 1 )
        y1 = random.uniform( -1, 1 )
        s1 = x1*x1 + y1*y1
        if s1 < 1:
            break
    while True:
        x2 = random.uniform( -1, 1 )
        y2 = random.uniform( -1, 1 )
        s2 = x2*x2 + y2*y2
        if s2 < 1:
            break
    q0 = x1; q1 = y1; q2 = x2*math.sqrt( ( 1 - s1 )/s2 );
        q3 = y2*math.sqrt( ( 1 - s1 )/s2 )
    norm = math.sqrt( q0*q0 + q1*q1 + q2*q2 + q3*q3 )
    q0 = q0/norm; q1 = q1/norm; q2 = q2/norm; q3 = q3/norm
         # normalize the quaternions
    return [ q0, q1, q2, q3 ]

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Read the residue coordinates from
```

```
# library and perform translation
# and rotation.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def build_res_geom( resname, dr, q ):
    res_lib = lib_path + resname + '.xyz'
    natoms, atype, res_geom, = read_coords( res_lib )
              # read in coordinates from file
    res_geom = rotate_res_rev( q, res_geom )
                      # rotate by q
    res_geom = translate_res( dr, res_geom )
                      # translate by dr
    return natoms, atype, res_geom

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Concatenate the geometries of the
# residues and the templates to a
# valid active site; sequence is a
# list of the residues, l_atype and
# l_geom hold the different atom
# types and geometries.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def cat_geom( sequence, l_atype, l_geom ):
    natoms, atype, as_geom = read_coords( lib_path +
        template_a + '.xyz' )
    rescount = 1
    for res, ratype, res_geom in zip( sequence, l_atype,
        l_geom ):
        # combine everything:
        atype = hstack( ( atype, ratype ) )
        as_geom = row_stack( ( as_geom, res_geom ) )
        natoms = natoms + res_size[res]
        rescount += 1
    return natoms, atype, as_geom


#=========================================
# POPULATION CREATION
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create a canditate set of <individuals>
# individuals using nproc cores;
# the indexing is done using a global
# variable species_id
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def ppopulate( individuals ):
    global species_id
    species = []
    # initialize population:
    for i in xrange( individuals ):
        species_id += 1                                  #
             keep track of individual IDs
        species.append(  "I%03d" % species_id )          #
             generate the name
    # scheme to split processor load equally:
    b = 0; chunks = []                                   #
         chunks holds the pieces for very processor
    for i in range( nproc ):
        quotient, remainder = divmod( individuals - i,
            nproc)
        a = b; b = b  + quotient + ( remainder != 0 )    #
             expression in brackets 1 if true and zero if
            false
```

```
        chunks.append( species[a:b] )                          #
             split population accordingly
    # start parallel computation:
    canditates = {}                                            #
         will hold canditates
    subprocs = []                                              #
         repository to keep track of subprocesses
    queue = Queue()                                            #
         queue object used for computation
    # start genome generation using the create_genome
       function:
    for i in range( nproc ):
        r_seed = i * ( individuals + 1 )
        subproc = multiprocessing.Process( target=
            create_genom, args=( queue, chunks[i], r_seed )
             )
        subprocs.append( subproc )
        subproc.start()
    for i in range( nproc ):                                   #
         retrieve results from the queue
        canditates.update( queue.get() )
    for indi in canditates:
        is_comp[indi] = '0'                                    #
             tag structure as not validated yet
    for subproc in subprocs:                                   #
         wrap all the processes up
        subproc.join()
    print "␣Geometry␣creation␣finished..."
    return canditates

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Use a queue object and a list of
# individuals to create a valid active
# site geometry; rand_seed holds seed
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def create_genom( queue, individuums, rand_seed ):
    np.random.seed( rand_seed )                                #
         seed rng for additional diversity
    rstate = np.random.get_state()
    cand_chunk = {}
        # will hold part of canditates computed by the
        function
    for indi in individuums:
        np.random.set_state( rstate )
        no_alleles = np.random.random_integers( al_min,
            al_max )             # generate genome size
#        sequence = [ choice( l_resid ) for i in range(
    no_alleles ) ]        # generate sequence from residues
     in l_resid
        sequence = [ 'met', choice( [ 'asn', 'gln', 'thr',
            'ser' ] ), choice( [ 'asn', 'gln', 'thr', 'ser'
             ] ), 'phe' ] # <= BIASED SEQUENCE
        tmp_geom = coll_geom

            # geomertry to be checked against (central
            template)
        l_atype = []; l_geom = []; l_coords = []
                                        # list of atom
            types, geometries and displacements
        for res in sequence:
            i_res = sequence.index( res )
                                                   # for
                coords (BIASED)
            while True:
```

```
                coords = create_coords()
                                                    #
                    use zylider with z oriented template
                coords = create_coords_bias( i_res )
                                          # <= BIASED
                    COORDINATES
                quaternions = create_quat()
                                                 # get
                    rotation
                natoms, atype, res_geom = build_res_geom(
                    res, coords, quaternions )        #
                    build residue geometry from
                    displacement and rotation
                # check if generated geometries are valid:
                if res_orientation( res, coords, res_geom )
                     == 0:                            # 1)
                    check for orientation
                    coll, tmp_geom = res_coll_brute_force(
                        res_geom, tmp_geom )             #
                        retrieve results of collision check
                    if coll == 0:

                        # 2) check for collisions
                        tmp_geom = row_stack( ( tmp_geom,
                            res_geom ) )
                                                # updat
                            list of coordinates to be
                            checked against
                        l_atype.append( atype ); l_geom.
                            append( res_geom ); l_coords.
                            append( coords ) # update
                            genome
                        break
            cand_chunk.update( { indi : [ sequence, l_atype,
                l_geom ] } )          # update canditates
            rstate = np.random.get_state()
                                                  # reseed
                 rng ( maybe counterproductive )
        for species in cand_chunk:
            write_species( species, cand_chunk )
                                               # write
                geometry to output-folder
        queue.put( cand_chunk )
                                                          #
             put results in queue


#=========================================
# RESTART
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Initialize a restart from binary log;
# important containers (e.g. list of
# canditates, fitnesses etc.) are
# imported and the status of the
# previously running computations is
# checked and saved in the dictionary
# 'restart_points'
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def restart( restart_log ):
    global fitness
    global is_comp
    global population
```

```
    global restart_points
    global species_id
    global evals
    try:
        log_file = open( log_path + restart_log, 'r+b' )
                                            # open log-file
    except:
        print "␣Log-file␣%s␣for␣restart␣could␣not␣be␣found.
            " % restart_log
        exit()
    try:
        canditates, population, fitness, evals, species_id
            = pickle.load( log_file )   # extract pickled
            objects
    except:
        print "␣Binary␣file␣%s␣corrupted,␣data␣could␣not␣be
            ␣loaded." % restart_log
        exit()
    log_file.close()
    for individual in canditates.keys():
        restart_point = 0
        if os.path.exists( qm_path + individual + '/tm_1/
            control' ):                 # check if define
             was run
             restart_point += 1
        if os.path.exists( qm_path + individual + '/tm_1/
            GEO_OPT_CONVERGED' ):       # check if first
            optimisation was run
             restart_point += 1
        if os.path.exists( qm_path + individual + '/tm_2/
            control' ):                 # check for first
             frequency job
             restart_point += 1
        if os.path.exists( qm_path + individual + '/tm_2/
            GEO_OPT_CONVERGED' ):       # check for
            second optimisation
             restart_point += 1
        if individual in population:

            # if individual is in the population,
             is_comp[individual] = '1'

                # all computations have been performed
                already
             pass
        else:
             restart_points[individual] = restart_point
                                                # create
                dictionary of computation progress
             is_comp[individual] = '0'
    print "␣Initialization␣from␣%s␣finished." % restart_log
    return canditates


#=========================================
# STEADY STATE
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Main steady state function; after
# initialisation, jobs are checked for
# completion continuously and if a
# certain threshold is reached, genetic
# operations are performed and the
```

```
# obtained individuals sent to the
# queue for fitness evaluation until
# a certail amount ( no_eval ) of
# evaluations was performed
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def run_steady_state( individuals, no_eval ):
    global fitness
    global is_comp
    global population
    global to_comp
    global pivots
    global comp_processes                          # for easy
        termination
    global free_min
    global new_species
    global restart_points
    global evals
    #-------------------------------------
    # Initialize everything
    #-------------------------------------
    print "␣Initializing␣queue␣and␣creating␣geometries."
    evals = 0                                      # set
        evaluations to zero
    manager = multiprocessing.Manager()            # create
        manager object used to govern the shared list
    dead = manager.list()                          # create
        shared list
    comp_processes = {}                            #
        dictionary which will hold all currently running
        processes
    comp_queue = multiprocessing.Queue()           #
        initialize the queue which will handle the parallel
         data processing
    #-------------------------------------
    # Start normal initialisation:
    #-------------------------------------
    if is_restart == 0:
        canditates = ppopulate( individuals )      #
            initialize the population
        print_genome( canditates, canditates )     # print
            genome of canditates
        print_evaluate( canditates )               # print
            structures to be evaluated
        print "␣Starting␣steady-state␣evaluation."
        for individual in canditates:
            to_comp[individual], pivots[individual] =
                get_charge_pivot( individual, canditates )
                        # get the charge and the pivot
                carbons
            comp_proc = multiprocessing.Process( target =
                comp_tm, args = ( comp_queue, individual,
                dead ) )   # create the evaluation process
            comp_processes[individual] = comp_proc

                # register the process
            comp_proc.start()

                # and start it
    #-------------------------------------
    # Restart form binary log-file:
    #-------------------------------------
    elif is_restart == 1:
        print "␣Restarting␣computations␣from␣%s." %
            restart_log
```

```
        canditates = restart( restart_log )       # build
            data structures from log and check for
            computation progress
        print_evaluate( restart_points )          # print
            structures to be reevaluated
        print "␣Starting␣steady-state␣reevaluation."
        for individual in restart_points:         # restart
             the computations, using special master
            function 'comp_restart'
            to_comp[individual], pivots[individual] =
                get_charge_pivot( individual, canditates )
            comp_proc = multiprocessing.Process( target =
                comp_restart, args = ( comp_queue,
                individual, dead ) )
            comp_processes[individual] = comp_proc
            comp_proc.start()
    running = len( comp_processes )               # get the
        number of currently running evaluatiuons
    #-------------------------------------
    # Keep continous flow running
    #-------------------------------------
    while evals < no_eval:                                #
         will finish after no_eval computations
        finished = []                                     #
             will processes finished this turn
        for i in xrange( qm_max ):                        #
             check if processes finished since last turn
            try:
                individual = comp_queue.get( True, 5 )    #
                     try to retrieve values from queue, if
                    nothing is to be gained, stop after one
                     second
                finished.append( individual )             #
                     collect finished processes
                comp_processes[individual].join()         #
                     wrap them up
                del comp_processes[individual]            #
                     and unregister them
            except:
                pass
        running = len( comp_processes )                   #
             get number of processes still running
        print "\n␣~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
        print "␣␣␣␣CHECK␣%s␣␣␣␣" %( datetime.datetime.now()
            .strftime("%Y-%m-%d␣%H:%M") )
        print "␣~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
        print "␣Evaluations␣done␣so␣far:␣%d" % evals
        print "␣Computations␣running:␣␣␣␣%d" % running
        if len( dead ) > 0:
            print "\n␣++++++++␣In␣memoriam␣+++++++"
            print_list( dead )
            print "␣++++++++++++++++++++++++++++"
        for individual in finished:
            evals += 1                                    #
                 update number of evaluations
            if individual in dead:
                del canditates[individual]                #
                     remove failed evaluations from the
                    candidates
                del is_comp[individual]                   #
                     and from the computation list
            elif individual not in dead:                  #
                 check if individual finished computations
                successfully
```

```
            population.append( individual )           #
                 add to population
            canditates = update_genome( canditates,
                individual ) # update the genome with
                optimized coordinates
            update_fitness( individual, 1, 2 )
            fit_penalty( individual, canditates )     #
                 incur penalty if oxygen coordinates to
                 ruthenium
            if individual in dead:                    #
                 check if fitness computation succeeded
                del canditates[individual]            #
                     remove failed evaluations from the
                     candidates
                del is_comp[individual]               #
                     and from the computation list
            else:
                is_comp[individual] = '1'             #
                     mark individual as evaluated
    if len( finished ) > 0:
        write_log( canditates )                       #
             write the log files
    #-------------------------------------
    # If threshold is reached, process
    # population
    #-------------------------------------
    if len( finished ) > 0 and len( population ) >=
        p_threshold: # wait until a job is completed
        and the population has a certain size
        print_genome( population, canditates )       #
            print genomes of current population
        print_fitness( population )                  #
            print fitness of population
        running = len( comp_processes )              #
            get running processes
        free_slots = qm_max - running                #
            get free queue slots (SGE queue not python)
        #-------------------------------------
        # If there are enough slots, do
        # genetic operations
        #-------------------------------------
        if free_slots >= free_min:
            print "␣*****************************"
            print "␣*␣␣␣␣␣THRESHOLD␣REACHED␣␣␣␣␣␣*"
            print "␣*****************************"
            print "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"
            print "␣Starting␣genetic␣operations...\n"
            print "␣Unleashing␣romantic␣music..."
            tournament_size = int( rint( len(
                population ) * 0.14 + 1.4 ) )       #
                compute tournament size based on
                population size; linear fit is used
            print "␣Using␣tournaments␣of␣size␣%d␣for␣
                parent␣selection." % tournament_size
            canditates, children = mate( n_offspring,
                canditates )      # create children
                through crossover
            print "\n␣Found␣a␣bar␣of␣Californium."
            canditates, mutant, mutated = mutate(
                canditates )          # create a mutant
                , return 0 if function failed
            #-------------------------------------
            # if no mutant could be created,
            # generate additional new species
```

```
#-------------------------------------
print "\n Launching an expedition..."
if mutated == 0:
    mutants = []

        # needs to be emptied to avoid key
        clashes
    new_cand = ppopulate( new_species + 1 )
                          # create
        completely new species
else:
    mutants = [ mutant ]

        # create list of mutants
    new_cand = ppopulate( new_species )
                            # create
        completely new species
print " %d new species discovered." %len(
    new_cand )
canditates.update( new_cand )
                                    # update
    canditates with new geometry
#-------------------------------------
# Send new geometries to queue for
# evaluation and print information
#-------------------------------------
to_evaluate = children + mutants + [ cand
    for cand in new_cand ] # build a list
    of those to be evaluated
print_evaluate( to_evaluate )
                                     # print
    them out
print " Sending jobs to queue.\n"
for individual in to_evaluate:
                                    # register
     them for computation
    to_comp[individual], pivots[individual]
         = get_charge_pivot( individual,
        canditates )        # get charge,
        pivot carbons
    comp_proc = multiprocessing.Process(
        target = comp_tm, args = (
        comp_queue, individual, dead ) )  #
         create process
    comp_processes[individual] = comp_proc

        # register it
    comp_proc.start()

        # and start
#-------------------------------------
# Reduce the population size by
# removing the most unfit individuals
# if it grows too large
#-------------------------------------
while len( population ) > individuals:
    ssentif = {}
    for indi in population:
         ssentif[indi] = fitness[indi]
    to_cull = max( ssentif, key = ssentif.
        get )
    population.remove( to_cull )
    print " Population too large, removing
        suspicous subject %s." % to_cull
```

```
                ssentif = {}
                for indi in population:
                    ssentif[indi] = fitness[indi]
                print "␣Average␣fitness:␣␣%f" %( sum(
                    ssentif.values() ) / len( ssentif )  )
                                    # give information
                    about average fitness
                print "␣Minimum␣fitness:␣␣%f" %( min(
                    ssentif.values() ) )
                                                        #
                    give information about minimum fitness
                print "␣Maximum␣fitness:␣␣%f" %( max(
                    ssentif.values() ) )
                                                        #
                    give information about maximum fitness
    print_genome( population, canditates )
    print_fitness( population )
    for indi in comp_processes:
        comp_processes[indi].terminate()


#=========================================
# POPULATION OPERATORS
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Select n_sel parents for reproduction
# and produce n_offspring children by
# crossover
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def mate( n_offspring, canditates ):
    global species_id
    children = []
    offsp = 0
    while offsp < n_offspring:
        mates = tournament_select( n_sel )
                                                    #
            use tournament selection to choose parents
#         childa, childb = crossover( canditates[mates[0]],
    canditates[mates[1]] )      # do crossover
        childa, childb = crossover_single_point( canditates
            [mates[0]], canditates[mates[1]] )      # do
            alternative crossover routine
        if child_coll_brute_force( childa[2], 0.04 ) == 0:
                                    # check if children
            are geometrically feasible
            species_id += 1

                # keep track of individual IDs
            name = "I%03d" % species_id

                # generate the name
            canditates[name] = childa
            children.append( name  ), childa, name
            is_comp[name] = '0'

                # needs to be evaluated
            print len( childa )
            write_species( name, canditates )
            offsp += 1
        # same process for other child, for odd numbers of
            children see if number of offspring is already
            reached.
```

```
            if offsp < n_offspring and child_coll_brute_force(
                childb[2], 1.0 ) == 0:
                species_id += 1

                    # keep track of individual IDs
                name = "I%03d" % species_id

                    # generate the name
                canditates[name] = childb
                print len( childb ), childb, name
                children.append( name )
                is_comp[name] = '0'

                    # needs to be evaluated
                write_species( name, canditates )
                offsp +=1
        print "␣%s␣followed␣the␣call␣of␣nature␣and␣gave␣birth␣
            to␣%s." %( "␣and␣".join( mates ), "␣and␣".join(
            children ) )
        return canditates, children

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Take the genomes of two parents and
    # perform a crossover routine designed
    # for genomes of different size;
    # CR is the crossover rate
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def crossover( genome_1, genome_2 ):
        genome_1 = copy.deepcopy( genome_1 )
        genome_2 = copy.deepcopy( genome_2 )
        size_1 = len( genome_1[0] ); size_2 = len( genome_2[0]
            )
        indices_1 = range( size_1 ); indices_2 = range( size_2
            )
        child_1 = [[],[],[]]
        child_2 = [[],[],[]]
        iter = min( size_1, size_2 )
        for i in range( iter ):
            allele_1 = choice( indices_1 ); allele_2 = choice(
                indices_2 )
            if np.random.sample() >= CR:
                # no crossover happens:
                child_1[0].append( genome_1[0][allele_1] )
                child_1[1].append( genome_1[1][allele_1] )
                child_1[2].append( genome_1[2][allele_1] )
                child_2[0].append( genome_2[0][allele_2] )
                child_2[1].append( genome_2[1][allele_2] )
                child_2[2].append( genome_2[2][allele_2] )
            else:
                # crossover happens:
                child_1[0].append( genome_2[0][allele_2] )
                child_1[1].append( genome_2[1][allele_2] )
                child_1[2].append( genome_2[2][allele_2] )
                child_2[0].append( genome_1[0][allele_1] )
                child_2[1].append( genome_1[1][allele_1] )
                child_2[2].append( genome_1[2][allele_1] )
            indices_1.remove( allele_1 ); indices_2.remove(
                allele_2 )
        # distribute remaining alleles for genomes of different
             size, original distribution is favored if
            crossoverrate CR is low
        if indices_1 > indices_2:
            if np.random.sample() >= CR:
                for index in indices_1:
```

```
                child_1[0].append( genome_1[0][index] )
                child_1[1].append( genome_1[1][index] )
                child_1[2].append( genome_1[2][index] )
        else:
            for index in indices_1:
                child_2[0].append( genome_1[0][index] )
                child_2[1].append( genome_1[1][index] )
                child_2[2].append( genome_1[2][index] )
    else:
        if np.random.sample() >= CR:
            for index in indices_2:
                child_2[0].append( genome_2[0][index] )
                child_2[1].append( genome_2[1][index] )
                child_2[2].append( genome_2[2][index] )
        else:
            for index in indices_2:
                child_1[0].append( genome_2[0][index] )
                child_1[1].append( genome_2[1][index] )
                child_1[2].append( genome_2[2][index] )
    return child_1, child_2

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Alternative crossover routine for
# heavily biased genomes, crossover
# preserves order of residues
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def crossover_single_point( genome_1, genome_2 ):
    size_1 = len( genome_1[0] ); size_2 = len( genome_2[0]
        )
    child_1 = [[],[],[]]
    child_2 = [[],[],[]]
    iter = min( size_1, size_2 )
    for i in xrange( iter ):
        if np.random.sample() <= CR:
            # crossover happens:
            child_1[0].append( genome_2[0][i] )
            child_1[1].append( genome_2[1][i] )
            child_1[2].append( genome_2[2][i] )
            child_2[0].append( genome_1[0][i] )
            child_2[1].append( genome_1[1][i] )
            child_2[2].append( genome_1[2][i] )
        else:
            # no crossover happens:
            child_1[0].append( genome_1[0][i] )
            child_1[1].append( genome_1[1][i] )
            child_1[2].append( genome_1[2][i] )
            child_2[0].append( genome_2[0][i] )
            child_2[1].append( genome_2[1][i] )
            child_2[2].append( genome_2[2][i] )
    return child_1, child_2

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Tournament selection to select mates;
# selection pressure can be adjusted
# via tournament_size; clones of the
# parents are possible
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def tournament_select( n_sel ):
    mates = []
    while len( mates ) < n_sel:
        canditates = {}
        while len( canditates ) < tournament_size:
            canditate = choice( population )
            if canditate not in canditates:
```

```
                canditates[canditate] = fitness[canditate]
        winner = min( canditates, key = canditates.get )
        mates.append( winner )
    return mates

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Mutation routine, using a population
# size dependant mutation rate;
# mutation occurs through residue
# exchange, where the position and
# orientation in the original genome
# is kept
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def mutate( canditates ):
    global population
    global species_id
    mutation_rate = 1.0 / ( len( population ) + 1 )
                    # since population size changes with
        time, deterministic mutation can be implemented in
        this way
    print "␣Current␣mutation␣rate:␣␣␣%f" % mutation_rate
    mutant = choice( population )
                                        # choose
        individual used as template for mutant creation
    species_id += 1
                                                        #
        increment id for naming
    name = "I%03d" % species_id
    canditates[name] = copy.deepcopy( canditates[mutant] )
            # generate new entry in canditates for
        future mutant
    counter = 0
    while True:
        mutated = 0

            # set unmutated flag
        for i in range( len( canditates[name][0]) ):
            if np.random.rand() < mutation_rate:
                                # check if mutation
                occured
                # if it did, exchange residues
                canditates[name][0][i], canditates[name
                    ][1][i], canditates[name][2][i] =
                    res_exchange( canditates[mutant][0][i],
                     canditates[mutant][1][i], canditates[
                    mutant][2][i] )
                mutated = 1
        if mutated == 1:
                                                # do
            collision check to see if new geometry is
            feasible
            collc = off_coll_brute_force( canditates[name
                ][2] )
            if collc == 0:
                print '␣%s␣produces␣a␣mutant␣%s.' %( mutant
                    , name)
                is_comp[name] = '0'
                                            # set as
                    unevaluated
                write_species( name, canditates )
                                # write geometry
                break
```

```
            # if to many iterations would be required, skip
                mutation and create additional new
               individual
            elif counter > 1000:
                species_id -= 1
                                                    # reset
                    counter for naming
                mutated = 0
                print ' %s avoided the radioactive waste.'
                    % mutant
                break
        counter += 1
    return canditates, name, mutated

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Mutation is carried out through the
# exchange of single residues, while
# trying to preserve the original
# position and orientation
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def res_exchange( res, a_list, res_geom ):
    res_geom, dr = center_geom( res_geom )
                                    # compute the
        translation
    new_res = choice( l_resid )
                                                # choose new
        residue
    natoms, na_list, new_res_geom = read_coords( lib_path +
         new_res + '.xyz' )  # read in geometry from
        library
    q = get_quat( res_geom[0,:], new_res_geom[0,:] )
                         # get quaternions for orientation
    # rotate and translate
    new_res_geom = rotate_res_rev( q, new_res_geom )
    new_res_geom = translate_res( dr, new_res_geom )
    return new_res, na_list, new_res_geom


#=========================================
# GEOMETRY ANALYSIS OF OUTPUT
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Compute the quaternions neccessary to
# transform vector a to vector b; used
# in the res_exchange function
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def get_quat( v_a, v_b ):
    v_a = v_a / linalg.norm( v_a );  v_b = v_b / linalg.
        norm( v_b )   # compute unit vectors
    n_ab = np.cross( v_b, v_a )
                                                # get the
        normal vector
    n_ab = n_ab / linalg.norm( n_ab )
    theta = np.arccos( dot( v_a, v_b ) )
                                    # compute angle
        between vectors
    theta = theta * 0.5
    q0 = np.cos( theta ); qr = np.sin( theta ) * n_ab
                         # get the unit quaternions
    q = hstack( ( q0, qr ) )
    q = q / linalg.norm( q )
    return q
```

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Read in optimized geometries and
# split them according to the amino
# acid residues used during creation
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def read_split( filename, genome ):
    natoms, atype, geom = read_coords( filename )
                          # read in data from geometry
        file
    geom = geom[res_size[template_a]:]
    atype = atype[res_size[template_a]:]
    gen_geoms = []
                                                        #
         create list of arrays with residue geometries
    atom_types = []
    # split accordingly
    for res in genome[0]:
        res_geom = geom[:res_size[ res ]]
        atypes = atype[:res_size[ res ]]
        geom = geom[res_size[ res ]:]
        atype = atype[res_size[ res ]:]
        atom_types.append( atypes )
        gen_geoms.append( res_geom )
    return atom_types, gen_geoms

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Compute the translation of the
# geometric center from the origin;
# used in the res_exchange function
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def center_geom( geom ):
    x_sum, y_sum, z_sum = geom.sum(axis=0)
    res_len = len( geom )
    dx = x_sum / res_len
    dy = y_sum / res_len
    dz = z_sum / res_len
    geom[:,0] -= dx
    geom[:,1] -= dy
    geom[:,2] -= dz
    return geom, [ dx, dy, dz ]


#============================================
# LOG-FILES
#============================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create a textual logfile holding the
# python constructs, as well as a
# pickle binary object for easier
# recovery
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def write_log( canditates ):
    filename = datetime.datetime.now().strftime("%Y-%m-%d-%
        H:%M")    # use current time and date as unique
        filename
    log_file = open( log_path + filename + '.log', 'w' )
    log_file.write( 'canditates' + '\n' )            #
        write canditates, fitness, is_comp and population
        to text file
    log_file.write( str( canditates ) + '\n' )
    log_file.write( 'fitness:\n' )
    log_file.write( str( fitness ) + '\n' )
    log_file.write( 'is_comp:\n' )
```

```
    log_file.write( str( is_comp ) + '\n' )
    log_file.write( 'population:\n' )
    log_file.write( str( population ) + '\n' )
    log_file.close()
    pop_file = open( log_path + filename + '.bin', 'wb' )
                # do the same in the form of a binary
        pickle file
    pickle.dump( ( canditates, population, fitness, evals,
        species_id ), pop_file )
    pop_file.close()
    print ' Log and binary file %s written.\n' %filename


#=========================================
# GENERAL QUANTUM CHEMISTRY INTERFACE
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Master function combining all steps
# of the Turbomole computations
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def comp_tm( queue, individual, dead ):
    # evaluate TS state:
    prep_dir( individual )
    prep_tm( individual, 1, template_a, dead )
    tm_opt( individual , 1, dead )
    tm_freq( individual, 1, dead )
    follow_build( individual, template_b, 2, dead )
    # evaluate educt:
    prep_tm( individual, 2, template_b, dead )
    tm_opt( individual , 2, dead )
    tm_freq( individual, 2, dead )
    queue.put( individual )

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Master function for restart from log;
# continues after last successfull step
# as specified in restart_points
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def comp_restart( queue, individual, dead ):
    if restart_points[individual] <= 1:                 #
        try to restart first optimisation
        tm_opt( individual , 1, dead )
    if restart_points[individual] <= 2:                 #
        begin with first frequency analysis
        tm_freq( individual, 1, dead )
        follow_build( individual, template_b, 2, dead )
        prep_tm( individual, 2, template_b, dead )
    if restart_points[individual] <= 3:                 #
        restart second optimisation
        tm_opt( individual , 2, dead )
    if restart_points[individual] <= 4:                 #
        second frequency analysis
        tm_freq( individual, 2, dead )
    queue.put( individual )

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Prepare folders in QM-directory and
# copy unrefined geometries
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def prep_dir( individual ):
    dir = qm_path + individual
    if not os.path.exists( dir ):
```

```
        os.makedirs( dir )

            # create directory if it does not already exist
    xyz_geom = out_path + individual + '.xyz'
    cp_geom = subprocess.Popen( [ 'cp', xyz_geom, dir + '/'
        + individual + '.xyz' ] )  # Copy geometries to
        target folder
    cp_geom.wait()

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Get charge on geometry and positions
# of pivot carbons in the coordinates
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def get_charge_pivot( individual, canditates ):
    netto_charge = charge[template_a]
                               # initialize with charge of
         central template
    cum_sum = res_size[template_a]
                                  # cumulative sum needed
        for pivot computation
    pivot = []
                                                          #
        list will hold pivot positions
    for res in canditates[individual][0]:
        netto_charge = netto_charge + charge[res]
                        # add residue charge
        pivot.append( cum_sum + 1 )
                                       # update pivot list
        cum_sum = cum_sum + res_size[res]
    return netto_charge, pivot

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Read out optimized geometry and
# update canditates
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def update_genome( canditates, individual ):
    dir = qm_path + individual
    atypes, geoms = read_split( dir + '/tm_1/topt.xyz',
        canditates[individual] )
    canditates[individual][1] = atypes
    canditates[individual][2] = geoms
    return canditates


#=========================================
# TURBOMOLE
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Prepare everything for subsequent
# Turbomole computation on point n;
# if n is 1, the whole geometry is
# optimized, else the pivot carbons
# of the amino acids will be frozen.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def prep_tm( individual, point, template, dead ):
    if individual in dead:
        # this construct is used in order
        return
            # to stop computation if error is encountered
    dir = qm_path
    target = qm_path + individual + '/' + 'tm_' + str(
        point )
    if not os.path.exists( target ):
```

```
        os.makedirs( target )
    os.chdir( target )
    coord_file = open( 'coords.tmp', 'w' )

        # open temporary coord file to write to
    if point == 1:
        proc = subprocess.Popen( [ 'x2t', '../' +
            individual + '.xyz' ], stdout = coord_file ) #
            transform coordinates and write them to file,
            don't forget, we are in subfolder
    else:
        geom_file = 'tm4tm'
        proc = subprocess.Popen( [ 'x2t', geom_file ],
            stdout = coord_file )                    #
            transform coordinates and write them to file,
            don't forget, we are in subfolder
    proc.wait()
                                                    # wait
        until finished
    coord_file = open( 'coords.tmp', 'r' )
                        # open for reading
    tm_freeze( coord_file, individual, point, template )
           # freeze the appropriate coordinates
    coord_file.close()
                                              # close file
    proc = subprocess.Popen( [ scr_path + 'prep_define',
        str( to_comp[individual] ), scr_path ], stdout =
        subprocess.PIPE, stderr = subprocess.STDOUT ) #
        script uses sed to prepare define input
    proc.wait()
    error = 0
    def_log = open( target + '/define.log' ).readlines()
    for line in def_log:
        if 'abnormally' in line:
            error = 1
    if error == 1:
        print ' define for %s finished unsuccessfully' %
            individual
        sys.stdout.flush()
        dead.append( individual )
    else:
        print ' define for %s finished successfully' %
            individual
        sys.stdout.flush()
    # additional control modifications
    for line in fileinput.FileInput( target + '/control',
        inplace = 1 ):
        if line.startswith( '$end' ):
            print '$maxcor 2000'
                                             # increase
               memeory for frequency analysis
            print line.rstrip()
        elif line.startswith( '$scfdump' ):
            pass
        else:
            print line.rstrip()
    os.chdir( dir )
                                                # change to
         parent directory

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Freeze coordinates depending on point
# in evaluation procedure using
# coord_file as template; first point:
```

```
# only central template is frozen;
# else pivot carbons are frozen too
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def tm_freeze( coord_file, individual, point, template ):
    if point == 1:
        unfreeze = unfreeze_a
                                              # unfreeze
            certain parts of template
    else:
        unfreeze = unfreeze_b
    counter = 1
                                                        #
        counter to keep track of frozen atoms
    temp = open( 'coords', 'w' )
                                    # open final coordinate
        file for writing
    diff = res_size[template_a] - res_size[template]
    for line in coord_file:
        line = line.strip()
                                               # strip to get
            rid of newlines
        if counter > 1 and counter <= ( res_size[template]
            + 1 - unfreeze ): # add f to freeze template (
            beware, file starts with $coord )
             line = line + ' f'
        # If computation but the first, freeze amino acid
            pivot points
        elif point != 1 and counter + diff - 1  in pivots[
            individual]:
             line = line + ' f'
        counter += 1
        temp.write( line + '\n' )
    temp.close()

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Submit Turbomole jobs to specified
# queue; in order to speed up
# conversion, an analytical hessian is
# computed before the optimisation
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def tm_opt( individual , point, dead ):
    if individual in dead:
        return
    dir = qm_path + individual
    target = dir + '/tm_' + str( point )
    os.chdir( target )
    # generate TURBOMOLE script for job submission:
    script_t = '''#!/bin/bash
#
#$ -cwd
#$ -q %s
#$ -S /bin/bash
#$ -N %s
#$ -sync y
#$ -r n
#
export INPDIR=$SGE_O_WORKDIR
export TURBOTMPDIR=$TMPDIR
export QSUB_WORKDIR=$TMPDIR
cp $INPDIR/* $TMPDIR
cd $TMPDIR
jobex -ri -c 50 >& jobex.log
grep -v "hssupdate" control > control.tmp
mv control.tmp control
```

```
aoforce␣>␣aoforce.out
jobex␣-ri␣-c␣3000␣>␣jobex.log
cp␣-f␣$TMPDIR/*␣$INPDIR
rm␣-rf␣$TMPDIR''' % ( qm_queue, 'tm_o' + str( point ) + '_'
    + individual ) # Create script for qsub
    #-------------------------------------
    # The sync keyword is used to keep
    # track of queue operations; the
    # optimisation starts with 50 steps
    # of steepest descent, then computes
    # an analytical hessian and proceeds
    # with the default optimisation
    # routine
    #-------------------------------------
    script = open( 'tm_submit.sh', 'w' )
                                # write script to file
    script.write( script_t )
    script.close()
    # execute script and submit job
    comp = subprocess.Popen( [ 'qsub', './tm_submit.sh' ],
        stderr = subprocess.STDOUT, stdout = subprocess.
        PIPE ) # submit job to queue
    print '␣TURBOMOLE␣job␣%s␣(point␣%d)␣submitted␣to␣%s' %(
         individual, point, qm_queue )
    sys.stdout.flush()
    comp.wait()
                                                        #
         wait for its completion
    print '␣TURBOMOLE␣optimization␣of␣%s␣(point␣%d)␣
        finished.' %( individual, point )
    sys.stdout.flush()
    os.chdir( dir )

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Check if computation finished
# successfully and follow up with
# frequency analysis for fitness
# evaluation
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def tm_freq( individual, point, dead ):
    if individual in dead:
        return
    dir = qm_path + individual
    target = dir + '/tm_' + str( point )
    os.chdir( target )
    # Check if tm computations ended successfully ( assumes
         'not.converged' as indicator )
    if os.path.exists( 'diis_errvec' ):
        print '␣SCF␣convergence␣problem␣in␣calculation␣of␣%
            s␣(point␣%d)' %( individual, point )
        sys.stdout.flush()
        dead.append( individual )
    # Check if optimisation converged:
    elif os.path.exists( 'GEO_OPT_FAILED' ):
        print '␣Geometry␣optimisation␣of␣%s␣(point␣%d)␣not␣
            converged' %( individual, point )
        sys.stdout.flush()
        dead.append( individual )
    else:
        convert = subprocess.Popen( [scr_path + '/
            tm_extract.sh'] ) # external script uses head,
            tail and awk to extract geometry
        convert.wait()
        # generate turbomole script for frequency analysis
```

```
        script_t = '''#!/bin/bash
#
#$ -cwd
#$ -q %s
#$ -S /bin/bash
#$ -N %s
#$ -sync y
#$ -r n
#
export INPDIR=$SGE_O_WORKDIR
export TURBOTMPDIR=$TMPDIR
export QSUB_WORKDIR=$TMPDIR
cp $INPDIR/* $TMPDIR
cd $TMPDIR
aoforce > aoforce.out
cp -f $TMPDIR/* $INPDIR
rm -rf $TMPDIR''' % ( qm_queue, 'tm_f' + str( point ) + '_'
    + individual ) # Create script for qsub
        script = open( 'tm_freq_submit.sh', 'w' )
                          # write script to file
        script.write( script_t )
        script.close()
        # execute script and submitt job
        comp = subprocess.Popen( [ 'qsub', './
            tm_freq_submit.sh' ], stderr = subprocess.
            STDOUT, stdout = subprocess.PIPE ) # submit job
             to queue
        print ' TURBOMOLE frequency analysis of  %s (point
            %d) submitted to %s' %( individual, point,
            qm_queue )
        sys.stdout.flush()
        comp.wait()
                                                          #
             wait for its completion
        print ' TURBOMOLE frequency analysis of %s (point %
            d) finished.' %( individual, point )
        sys.stdout.flush()
        os.chdir( dir ) # change back

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Prepare geometry for the follow up
# computations based on optimized one
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def follow_build( individual, template, point, dead ):
    if individual in dead:
        return
    dir = qm_path + individual
    target = dir + '/tm_' + str( point )
    if not os.path.exists( target ):
        os.makedirs( target )
    os.chdir( target )
    out_file = open( 'tm4tm', 'w' )
    file=open( lib_path + template + '.xyz', 'r' ).
        readlines()  # open geometry of second teplate
    natoms = int(file.pop(0))
                                        # read #atoms
    scrap = file.pop(0)
                                            # skip
        charge
    tm_opt = open( dir + '/tm_1/topt.xyz', 'r' ).readlines
        ()
    for i in xrange( res_size[template_a] + 2 ):
                      # +2 for reading xyz file
```

```
        scrap = tm_opt.pop(0)
                                                # delete
            lines containing template a geometry
    natoms = natoms + len( tm_opt )
    # write new xyz file
    out_file.write( str( natoms ) + '\n' + '\n' )
    for line in file:
        out_file.write( line )
                                               # write
            geometry to file
    for line in tm_opt:
        out_file.write( line )
    out_file.close()
    print "␣Created␣follow␣up␣geometry␣for␣%s." %
        individual
    os.chdir( dir )


#=========================================
# FITNESS EVALUATION
#=========================================

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Calculate the fitness (deltaG in
# kJ/mol) and update global fitness
# dictionary; the absolute value of
# the barrier is used, to avoid
# falling into sinks
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def update_fitness( individual, point_1, point_2 ):
    global fitness
    if individual in dead:
        return 1
    pg_1 = get_energy_point_n( individual, point_1 )
    pg_2 = get_energy_point_n( individual, point_2 )
    if pg_1 == "FAILED" or pg_2 == "FAILED":
        dead.append( individual )
        print "␣Fitness␣evaluation␣of␣%s␣failed." %
            individual
    else:
        fitness[individual] = abs( get_energy_point_n(
            individual, point_1 ) - ( get_energy_point_n(
            individual, point_2 ) - 812533.708620335 ) ) #
            last term is energy of ether

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# If oxygen coordinates to ruthenium
# a penalty is introduced; assumes Ru
# is at the origin
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def fit_penalty( individual, canditates ):
    global fitness
    is_penal = 0
                                                    #
        logical operator if penalty is to be applied
    pos_o = []
    for i in xrange( len( canditates[individual][1] ) ):
        for j in xrange( len( canditates[individual][1][i]
            ) ):
            if canditates[individual][1][i][j] == "O":
                pos_o.append( (i,j) )
                                               # get
                    positions of all oxygen atoms
    for ox in pos_o:
```

```
            i, j = ox
            vector = canditates[individual][2][i][j]
            diff_sq = dot( vector, vector )
            if diff_sq <= 6.25:
                                                    # allow a
                minimum distance of 2.5 Angstrom
                 is_penal = 1
        if is_penal == 1:
            fitness[individual] += 42
                                              # add penalty

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Compute the free energy on one point;
    # units are kJ/mol
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def get_energy_point_n( individual, point ):
        dir = qm_path + individual
        target = dir + '/tm_' + str( point )
        os.chdir( target )
        os.system( "freeh < %s > freeh.out" % ( scr_path + '
            freeh.in' ) )
        try:
            freeh_p = open( target + '/freeh.out' ).readlines()
            ener_f = open( target + '/energy' ).readlines()
        except:
            return "FAILED"
        for i in xrange( len( freeh_p ) ):
            if "ln(qtrans) ln(qrot) ln(qvib)" in freeh_p[i]:
                free_en = float( freeh_p[i+3].split()[5] )
        try:
            ener = float( ener_f[-2].split()[1] )
            energy_point = ener * 2625.49962 + free_en
        except:
            return "FAILED"
        os.chdir( dir )
        return energy_point


    ##################################################
    # MAIN PROGRAM
    ##################################################

    if __name__ == "__main__":

        cctoms, ccatype, coll_geom = read_coords( lib_path +
            template_a + '.xyz' )
        print_folders()
        init_charge_size()
        try:
            run_steady_state( popsize, no_eval )
        except (KeyboardInterrupt, SystemExit):
            for individual in comp_processes:
                comp_processes[individual].terminate()
            raise
            print "Goodbye"
```

BIBLIOGRAPHY

[1] J. M. Berg, J. L. Tymoczko, L. Stryer: *Biochemistry*. New York: W. H. Freeman, 5th edition (2002).

[2] S. J. Benkovic, S. Hammes-Schiffer: A Perspective on Enzyme Catalysis. *Science*, **301**(5637), 1196–1202 (2003).

[3] R. Wolfenden, M. J. Snider: The Depth of Chemical Time and the Power of Enzymes as Catalysts. *Acc. Chem. Res.*, **34**(12), 938–945 (2001).

[4] L. Hedstrom: *Enzyme Specificity and Selectivity*. John Wiley & Sons, Ltd (2001).

[5] J. B. Beilen, Z. Li: Enzyme technology: an overview. *Curr. Opin. Chem. Biol.*, **13**(4), 338–344 (2002).

[6] C. Jäckel, P. Kast, D. Hilvert: Protein Design by Directed Evolution. *Annu Rev Biophys*, **37**(1), 153–173 (2008).

[7] D. Hilvert: Critical analysis of antibody catalysis. *Annu. Rev. Biochem.*, **69**, 751–793 (2000).

[8] K. N. Houk, A. G. Leach, S. P. Kim, X. Zhang: Bindungsaffinitäten von Wirt-Gast-, Protein-Ligand- und Protein-Übergangszustands-Komplexen. *Angew. Chem. Int. Ed.*, **115**(40), 5020–5046 (2003).

[9] V. Nanda, R. L. Koder: Designing artificial enzymes by intuition and computation. *Nat Chem*, **2**(1), 15–24 (2010).

[10] G. Kiss, N. Çelebi Ölçüm, R. Moretti, D. Baker, K. N. Houk: Computational Enzyme Design. *Angew. Chem. Int. Ed.*, **52**(22), 5700–5725 (2013).

[11] T. Sasaki, E. T. Kaiser: Helichrome: synthesis and enzymic activity of a designed hemeprotein. *J. Am. Chem. Soc.*, **111**(1), 380–381 (1989).

[12] R. L. Koder, J. L. R. Anderson, L. A. Solomon, K. S. Reddy, C. C. Moser, P. L. Dutton: Design and engineering of an O_2 transport protein. *Nature*, **458**(7236), 305–309 (2009).

[13] A. Lombardi, C. M. Summa, S. Geremia, L. Randaccio, V. Pavone, W. F. DeGrado: Retrostructural analysis of metalloproteins: Application to the design of a minimal model for diiron proteins. *PNAS*, **97**(12), 6298–6305 (2000).

[14] C. M. Summa, A. Lombardi, M. Lewis, W. F. DeGrado: Tertiary templates for the design of diiron proteins. *Curr. Opin. Chem. Biol.*, **9**(4), 500–508 (1999).

[15] L. Di Costanzo, H. Wade, S. Geremia, L. Randaccio, V. Pavone, W. F. DeGrado, A. Lombardi: Toward the de Novo Design of a Catalytically Active Helix Bundle: A Substrate-Accessible Carboxylate-Bridged Dinuclear Metal Center. *J. Am. Chem. Soc.*, **123**(51), 12749–12757 (2001).

[16] G. A. Papoian, W. F. DeGrado, M. L. Klein: Probing the Configurational Space of a Metalloprotein Core: An ab Initio Molecular Dynamics Study of Duo Ferro 1 Binuclear Zn Cofactor. *J. Am. Chem. Soc.*, **125**(2), 560–569 (2003).

[17] M. Klemba, K. H. Gardner, S. Marino, N. D. Clarke, L. Regan: Novel metal-binding proteins by design. *Nat Struct Mol Biol*, **2**(5), 368–373 (1995).

[18] L. Regan, N. D. Clarke: A tetrahedral zinc(II)-binding site introduced into a designed protein. *Biochemistry*, **29**(49), 10878–10883 (1990).

[19] H. W. Hellinga, F. M. Richards: Construction of new ligand binding sites in proteins of known structure: I. computer-aided modeling of sites with pre-defined geometry. *Journal of Molecular Biology*, **222**(3), 763–785 (1991).

[20] D. N. Bolon, S. L. Mayo: Enzyme-like proteins by computational design. *PNAS*, **98**(25), 14274–14279 (2001).

[21] J. Desmet, M. D. Maeyer, B. Hazes, I. Lasters: The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, **356**(6369), 539–542 (1992).

[22] A. Zanghellini, L. Jiang, A. M. Wollacott, G. Cheng, J. Meiler, E. A. Althoff, D. Röthlisberger, D. Baker: New algorithms and an in silico benchmark for computational enzyme design. *Protein Sci.*, **15**(12), 2785–2794 (2006).

[23] A. Leaver-Fay, M. Tyka, S. M. Lewis, O. F. Lange, J. Thompson, R. Jacak, K. Kaufman, P. D. Renfrew, C. A. Smith, W. Sheffler, I. W. Davis, S. Cooper, A. Treuille, D. J. Mandell, F. Richter, Y.-E. A. Ban, S. J. Fleishman, J. E. Corn, D. E. Kim, S. Lyskov, M. Berrondo, S. Mentzer, Z. Popović, J. J. Havranek, J. Karanicolas, R. Das, J. Meiler, T. Kortemme, J. J. Gray, B. Kuhlman, D. Baker, P. Bradley: ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Meth. Enzymol.*, **487**, 545–574 (2011).

[24] L. Jiang, E. A. Althoff, F. R. Clemente, L. Doyle, D. Röthlisberger, A. Zanghellini, J. L. Gallaher, J. L. Betker, F. Tanaka, C. F. Barbas, D. Hilvert, K. N. Houk, B. L. Stoddard, D. Baker: De Novo Computational Design of Retro-Aldol Enzymes. *Science*, **319**(5868), 1387–1391 (2008).

[25] D. Röthlisberger, O. Khersonsky, A. M. Wollacott, L. Jiang, J. DeChancie, J. Betker, J. L. Gallaher, E. A. Althoff, A. Zanghellini, O. Dym, S. Albeck, K. N. Houk, D. S. Tawfik, D. Baker: Kemp elimination catalysts by computational enzyme design. *Nature*, **453**(7192), 190–195 (2008).

[26] J. B. Siegel, A. Zanghellini, H. M. Lovick, G. Kiss, A. R. Lambert, J. L. St Clair, J. L. Gallaher, D. Hilvert, M. H. Gelb, B. L. Stoddard, K. N. Houk, F. E. Michael, D. Baker: Computational design of an enzyme catalyst for a stereoselective bimolecular Diels-Alder reaction. *Science*, **329**(5989), 309–313 (2010).

[27] S. D. Khare, Y. Kipnis, P. J. Greisen, R. Takeuchi, Y. Ashani, M. Goldsmith, Y. Song, J. L. Gallaher, I. Silman, H. Leader, J. L. Sussman, B. L. Stoddard, D. S. Tawfik, D. Baker: Computational redesign of a mononuclear zinc metalloenzyme for organophosphate hydrolysis. *Nat Chem Biol*, **8**(3), 294–300 (2012).

[28] D. J. Tantillo, C. Jiangang, K. N. Houk: Theozymes and compuzymes: theoretical models for biological catalysis. *Curr. Opin. Chem. Biol.*, **2**(6), 743–750 (1998).

[29] L. Pauling: Molecular Architecture and Biological Reactions. *Chem. Eng. News*, **24**(10), 1375–1377 (1946).

[30] J. H. Holland: *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press (1975).

[31] R. H. Grubbs: Olefin metathesis. *Tetrahedron*, **60**(34), 7117–7140 (2004).

[32] J. Mol: Industrial applications of olefin metathesis. *J. Mol. Catal.*, **213**(1), 39–45 (2004).

[33] K. C. Nicolaou, P. G. Bulger, D. Sarlah: Metathesis Reactions in Total Synthesis. *Angew. Chem. Int. Ed.*, **44**(29), 4490–4527 (2005).

[34] D. E. Koshland: Application of a Theory of Enzyme Specificity to Protein Synthesis. *Proc Natl Acad Sci USA*, **44**(2), 98–104 (1958).

[35] A. Warshel: Electrostatic Origin of the Catalytic Power of Enzymes and the Role of Preorganized Active Sites. *J. Biol. Chem.*, **273**(42), 27035–27038 (1998).

[36] X. Zhang, K. N. Houk: Why Enzymes Are Proficient Catalysts: Beyond the Pauling Paradigm. *Acc. Chem. Res.*, **38**(5), 379–385 (2005).

[37] T. Drepper, T. Eggert, W. Hummel, C. Leggewie, M. Pohl, F. Rosenau, S. Wilhelm, K.-E. Jaeger: Novel biocatalysts for white biotechnology. *Biotechnol. J.*, **1**(7-8), 777–786 (2006).

[38] G. Kiss, D. Röthlisberger, D. Baker, K. N. Houk: Evaluation and ranking of enzyme designs. *Protein Sci.*, **19**(9), 1760–1773 (2010).

[39] X. Zhang, J. DeChancie, H. Gunaydin, A. B. Chowdry, F. R. Clemente, Smith, T. M. Handel, K. N. Houk: Quantum Mechanical Design of Enzyme Active Sites. *J. Org. Chem.*, **73**(3), 889–899 (2008).

[40] A. H. Hoveyda, R. R. Schrock: Catalytic Asymmetric Olefin Metathesis. *Chem. Eur. J.*, **7**(5), 945–950 (2001).

[41] P. Schwab, R. H. Grubbs, J. W. Ziller: Synthesis and Applications of RuCl2(CHR')(PR3)2: The Influence of the Alkylidene Moiety on Metathesis Activity. *J. Am. Chem. Soc.*, **118**(1), 100–110 (1996).

[42] J. P. Morgan, R. H. Grubbs: In Situ Preparation of a Highly Active N-Heterocyclic Carbene-Coordinated Olefin Metathesis Catalyst. *Org. Lett.*, **2**(20), 3153–3155 (2000).

[43] T. M. Trnka, R. H. Grubbs: The Development of L_2X_2RuCHR Olefin Metathesis Catalysts: An Organometallic Success Story. *Acc. Chem. Res.*, **34**(1), 18–29 (2001).

[44] C. Adlhart, P. Chen: Comparing Intrinsic Reactivities of the First- and Second-Generation Ruthenium Metathesis Catalysts in the Gas Phase. *Helv. Cim. Acta*, **86**(4), 941–949 (2003).

[45] A. Correa, L. Cavallo: The Elusive Mechanism of Olefin Metathesis Promoted by (NHC)Ru-Based Catalysts: A Trade between Steric, Electronic, and Solvent Effects. *J. Am. Chem. Soc.*, **128**(41), 13352–13353 (2006).

[46] J. L. Hérisson, Y. Chauvin: Catalyse de transformation des oléfines par les complexes du tungstène. II. Télomérisation des oléfines cycliques en présence d'oléfines acycliques. *Makromol. Chem.*, **141**(1), 161–176 (1971).

[47] S. F. Vyboishchikov, M. Bühl, W. Thiel: Mechanism of Olefin Metathesis with Catalysis by Ruthenium Carbene Complexes: Density Functional Studies on Model Systems. *Chem. Eur. J.*, **8**(17), 3962–3975 (2002).

[48] M. S. Sanford, J. A. Love, R. H. Grubbs: Mechanism and activity of ruthenium olefin metathesis catalysts. *J. Am. Chem. Soc.*, **123**(27), 6543–6554 (2001).

[49] C. Adlhart, P. Chen: Mechanism and Activity of Ruthenium Olefin Metathesis Catalysts: The Role of Ligands and Substrates from a Theoretical Perspective. *J. Am. Chem. Soc.*, **126**(11), 3496–3510 (2004).

[50] L. Cavallo: Mechanism of Ruthenium-Catalyzed Olefin Metathesis Reactions from a Theoretical Perspective. *J. Am. Chem. Soc.*, **124**(30), 8965–8973 (2002).

[51] S. Torker, D. Merki, P. Chen: Gas-Phase Thermochemistry of Ruthenium Carbene Metathesis Catalysts. *J. Am. Chem. Soc.*, **130**(14), 4808–4814 (2008).

[52] I. H. Hillier, S. Pandian, J. M. Percy, M. A. Vincent: Mapping the potential energy surfaces for ring-closing metathesis reactions of prototypical dienes by electronic structure calculations. *Dalton Trans.*, **40**(5), 1061–1072 (2011).

[53] S. F. Vyboishchikov, W. Thiel: Ring-Closing Olefin Metathesis on Ruthenium Carbene Complexes: Model DFT Study of Stereochemistry. *Chem. Eur. J.*, **11**(13), 3921–3935 (2005).

[54] E. Schrödinger: Quantisierung als Eigenwertproblem. *Ann. Physik*, **384**(4), 361–376 (1926).

[55] E. Schrödinger: Quantisierung als Eigenwertproblem. *Ann. Physik*, **384**(6), 489–527 (1926).

[56] E. Schrödinger: Quantisierung als Eigenwertproblem. *Ann. Physik*, **386**(18), 109–139 (1926).

[57] M. Born, R. Oppenheimer: Zur Quantentheorie der Molekeln. *Ann. Physik*, **389**(20), 457–484 (1927).

[58] A. Szabo, N. S. Ostlund: *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory (Dover Books on Chemistry)*. Dover Publications, New edition edition (1996).

[59] P. Hohenberg, W. Kohn: Inhomogeneous Electron Gas. *Phys. Rev.*, **136**(3B), B864–B871 (1964).

[60] W. Kohn, L. J. Sham: Self-Consistent Equations Including Exchange and Correlation Effects. *Phys. Rev.*, **140**(4A), A1133–A1138 (1965).

[61] J. P. Perdew, K. Schmidt: Jacob's ladder of density functional approximations for the exchange-correlation energy. *AIP Conf. Proc.*, **577**(1), 1–20 (2001).

[62] J. C. Slater: A Simplification of the Hartree-Fock Method. *Phys. Rev.*, **81**(3), 385–390 (1951).

[63] P. A. M. Dirac: Quantum Mechanics of Many-Electron Systems. *Proc. R. Soc. Lond. A*, **123**(792), 714–733 (1929).

[64] S. H. Vosko, L. Wilk, M. Nusair: Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis. *Can. J. Phys.*, **58**(8), 1200–1211 (1980).

[65] A. D. Becke: Density-functional exchange-energy approximation with correct asymptotic behavior. *Phys. Rev. A*, **38**(6), 3098–3100 (1988).

[66] J. P. Perdew, K. Burke, M. Ernzerhof: Generalized Gradient Approximation Made Simple. *Phys. Rev. Lett.*, **77**(18), 3865–3868 (1996).

[67] J. P. Perdew: Density-functional approximation for the correlation energy of the inhomogeneous electron gas. *Phys. Rev. B*, **33**(12), 8822–8824 (1986).

[68] J. P. Perdew, Y. Wang: Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, **45**(23), 13244–13249 (1992).

[69] C. Lee, W. Yang, R. G. Parr: Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density. *Phys. Rev. B*, **37**(2), 785–789 (1988).

[70] A. D. Becke: Density-functional thermochemistry. IV. A new dynamical correlation functional and implications for exact-exchange mixing. *J. Chem. Phys*, **104**(3), 1040 (1996).

[71] J. Tao, J. P. Perdew, V. N. Staroverov, G. E. Scuseria: Climbing the Density Functional Ladder: Nonempirical Meta–Generalized Gradient Approximation Designed for Molecules and Solids. *Phys. Rev. Lett.*, **91**(14), 146401 (2003).

[72] P. J. Stephens, F. J. Devlin, C. F. Chabalowski, M. J. Frisch: Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields. *J. Phys. Chem.*, **98**(45), 11623–11627 (1994).

[73] J. P. Perdew, M. Ernzerhof, K. Burke: Rationale for mixing exact exchange with density functional approximations. *J. Chem. Phys.*, **105**(22), 9982–9985 (1996).

[74] V. N. Staroverov, G. E. Scuseria, J. Tao, J. P. Perdew: Comparative assessment of a new nonempirical density functional: Molecules and hydrogen-bonded complexes. *J. Chem. Phys.*, **119**(23), 12129–12137 (2003).

[75] S. Grimme: Semiempirical hybrid density functional with perturbative second-order correlation. *J. Chem. Phys.*, **124**(3), 034108–034108–16 (2006).

[76] O. Vahtras, J. Almlöf, M. Feyereisen: Integral approximations for LCAO-SCF calculations. *Chem. Phys. Lett.*, **213**(5–6), 514–518 (1993).

[77] K. Eichkorn, O. Treutler, H. Öhm, M. Häser, R. Ahlrichs: Auxiliary basis sets to approximate Coulomb potentials. *Chem. Phys. Lett.*, **240**(4), 283–290 (1995).

[78] M. Sierka, A. Hogekamp, R. Ahlrichs: Fast evaluation of the Coulomb potential for electron densities using multipole accelerated resolution of identity approximation. *J. Phys. Chem.*, **118**(20), 9136–9148 (2003).

[79] S. Grimme: Density functional theory with London dispersion corrections. *WIREs Comput. Mol. Sci.*, **1**(2), 211–228 (2011).

[80] S. Grimme: Semiempirical GGA-type density functional constructed with a long-range dispersion correction. *J. Comput. Chem.*, **27**(15), 1787–1799 (2006).

[81] S. Grimme, J. Antony, S. Ehrlich, H. Krieg: A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. *J. Chem. Phys.*, **132**(15), 154104 (2010).

[82] H. B. G. Casimir, D. Polder: The Influence of Retardation on the London-van der Waals Forces. *Phys. Rev.*, **73**(4), 360–372 (1948).

[83] A. Tkatchenko, M. Scheffler: Accurate Molecular Van Der Waals Interactions from Ground-State Electron Density and Free-Atom Reference Data. *Phys. Rev. Lett.*, **102**(7), 073005 (2009).

[84] A. D. Becke, E. R. Johnson: Exchange-hole dipole moment and the dispersion interaction. *J. Chem. Phys.*, **122**(15), 154104–154104–5 (2005).

[85] E. R. Johnson, A. D. Becke: A post-Hartree–Fock model of intermolecular interactions. *J. Chem. Phys.*, **123**(2), 024101–024101–7 (2005).

[86] E. R. Johnson, A. D. Becke: A post-Hartree-Fock model of intermolecular interactions: Inclusion of higher-order corrections. *J. Chem. Phys.*, **124**(17), 174104–174104–9 (2006).

[87] S. N. Steinmann, C. Corminboeuf: A Density Dependent Dispersion Correction. *CHIMIA*, **65**(4), 240–244 (2011).

[88] K. T. Tang, J. P. Toennies: An improved simple model for the van der Waals potential based on universal damping functions for the dispersion coefficients. *J. Chem. Phys.*, **80**(8), 3726–3741 (1984).

[89] O. A. Vydrov, T. Van Voorhis: Nonlocal van der Waals density functional: The simpler the better. *J. Chem. Phys.*, **133**(24), 244103–244103–9 (2010).

[90] Y. Zhao, D. G. Truhlar: Density Functionals with Broad Applicability in Chemistry. *Acc. Chem. Res.*, **41**(2), 157–167 (2008).

[91] J. Klimeš, A. Michaelides: Perspective: Advances and challenges in treating van der Waals dispersion forces in density functional theory. *J. Chem. Phys.*, **137**(12), 120901–120901–12 (2012).

[92] T. Risthaus, S. Grimme: Benchmarking of London Dispersion-Accounting Density Functional Theory Methods on Very Large Molecular Complexes. *J. Chem. Theory Comput.*, **9**(3), 1580–1591 (2013).

[93] W. Hujo, S. Grimme: Performance of Non-Local and Atom-Pairwise Dispersion Corrections to DFT for Structural Parameters of Molecules with Noncovalent Interactions. *J. Chem. Theory Comput.*, **9**(1), 308–315 (2013).

[94] W. T. G. Johnson, M. B. Sullivan, C. J. Cramer: meta and para substitution effects on the electronic state energies and ring-expansion reactivities of phenylnitrenes. *Int. J. Quantum Chem.*, **85**(4-5), 492–508 (2001).

[95] G. A. Jones, M. N. Paddon-Row, M. S. Sherburn, C. I. Turner: On the Endo/Exo Stereoselectivity of Intramolecular Diels–Alder Reactions of Hexadienylacrylates: An Interesting Failure of Density Functional Theory. *Org. Lett.*, **4**(22), 3789–3792 (2002).

[96] C. J. Cramer, S. E. Barrows: Quantum Chemical Characterization of Cycloaddition Reactions between the Hydroxyallyl Cation and Dienes of Varying Nucleophilicity. *J. Org. Chem.*, **63**(16), 5523–5532 (1998).

[97] C. Ho Choi, M. Kertesz, A. Karpfen: The effects of electron correlation on the degree of bond alternation and electronic structure of oligomers of polyacetylene. *J. Chem. Phys.*, **107**(17), 6712 (1997).

[98] J. Gräfenstein, E. Kraka, D. Cremer: The impact of the self-interaction error on the density functional theory description of dissociating radical cations: Ionic and covalent dissociation limits. *J. Chem. Phys.*, **120**(2), 524 (2004).

[99] E. Runge, E. K. U. Gross: Density-Functional Theory for Time-Dependent Systems. *Phys. Rev. Lett.*, **52**(12), 997–1000 (1984).

[100] J. Schirmer, A. Dreuw: Critique of the foundations of time-dependent density-functional theory. *Phys. Rev. A*, **75**(2), 022513 (2007).

[101] C. F. Guerra, J. G. Snijders, G. t. Velde, E. J. Baerends: Towards an order-N DFT method. *Theor. Chem. Acc.*, **99**(6), 391–403 (1998).

[102] M. A. Watson, N. C. Handy, A. J. Cohen: Density functional calculations, using Slater basis sets, with exact exchange. *J. Chem. Phys.*, **119**(13), 6475 (2003).

[103] J. Paier, R. Hirschl, M. Marsman, G. Kresse: The Perdew–Burke–Ernzerhof exchange-correlation functional applied to the G2-1 test set using a plane-wave basis set. *J. Chem. Phys.*, **122**(23), 234102 (2005).

[104] A. D. Boese, J. M. L. Martin, N. C. Handy: The role of the basis set: Assessing density functional theory. *J. Chem. Phys.*, **119**(6), 3005 (2003).

[105] C. J. Cramer: *Essentials of Computational Chemistry: Theories and Models*. Wiley, Second edition (2005).

[106] McQuarrie, D. Allan: *Statistical Mechanics*. University Science Books (2000).

[107] D. E. Goldberg: *Genetic algorithms in search, optimization, and machine learning*. 2, Addison-Wesley, Reading, MA (1989).

[108] J. E. Baker: Adaptive Selection Methods for Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, 101–111, L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1985).

[109] L. B. Booker, D. B. Fogel, D. Whitley, P. J. Angeline, A. E. Eiben: Recombination. In T. Bäck, D. B. Fogel, Z. Michalewicz (editors), *Evolutionary Computation 1 Basic Algorithms and Operators*, chapter 33, 256–307, Institute of Physics Publishing, Bristol (2000).

[110] G. Syswerda: Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 2–9, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989).

[111] M. C. Durrant: The Use of Quantum Molecular Calculations to Guide a Genetic Algorithm: A Way to Search for New Chemistry. *Chem. Eur. J.*, **13**(12), 3406–3413 (2007).

[112] J. Zhang, H. Chung, W. Lo: Pseudocoevolutionary genetic algorithms for power electronic circuits optimization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, **36**(4), 590–598 (2006).

[113] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, A. J. Olson: Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *J. Comput. Chem.*, **19**(14), 1639–1662 (1998).

[114] J. E. Baker: Adaptive Selection Methods for Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, 101–111, L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1985).

[115] T. Bäck, M. Schütz: Intelligent mutation rate control in canonical genetic algorithms. In Z. W. Raś, M. Michalewicz (editors), *Foundations of Intelligent Systems*, number 1079 in Lecture Notes in Computer Science, 158–167, Springer Berlin Heidelberg (1996).

[116] J. E. Smith: *Self adaptation in evolutionary algorithms*. PHD thesis, University of the West of England (1998).

[117] T. Bäck: Self-Adaptation in Genetic Algorithms. In *Proceedings of the First European Conference on Artificial Life*, 263–271, MIT Press (1992).

[118] K. Deb, H.-G. Beyer: Self-Adaptation in Real-Parameter Genetic Algorithms with Simulated Binary Crossover. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, 172–179, Morgan Kaufmann (1999).

[119] J. Smith, T. Fogarty: Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, 318–323 (1996).

[120] H. Wolfson, I. Rigoutsos: Geometric hashing: an overview. *Computational Science Engineering, IEEE*, **4**(4), 10–21 (1997).

[121] A. C. Tsipis, A. G. Orpen, J. N. Harvey: Substituent effects and the mechanism of alkene metathesis catalyzed by ruthenium dichloride catalysts. *Dalton Trans.*, (17), 2849–2858 (2005).

[122] T. Simonson, C. L. Brooks: Charge Screening and the Dielectric Constant of Proteins: Insights from Molecular Dynamics. *J. Am. Chem. Soc.*, **118**(35), 8452–8458 (1996).

[123] J. DeChancie, F. R. Clemente, A. J. Smith, H. Gunaydin, Y.-L. Zhao, X. Zhang, K. Houk: How similar are enzyme active site geometries derived from quantum mechanical theozymes to crystal structures of enzyme-inhibitor complexes? Implications for enzyme design. *Protein Sci*, **16**(9), 1851–1866 (2007).

[124] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, D. J. Fox: Gaussian 09 Revision D.01. Gaussian Inc. Wallingford CT 2009.

[125] TURBOMOLE V6.5 2013, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2007, TURBOMOLE GmbH, since 2007; available from `http://www.turbomole.com`.

[126] Y. Zhao, D. G. Truhlar: Attractive Noncovalent Interactions in the Mechanism of Grubbs Second-Generation Ru Catalysts for Olefin Metathesis. *Org. Lett.*, **9**(10), 1967–1970 (2007).

[127] M. M. Francl: Self-consistent molecular orbital methods. XXIII. a polarization-type basis set for second-row elements. *J. Chem. Phys.*, **77**(7), 3654 (1982).

[128] P. C. Hariharan, J. A. Pople: The influence of polarization functions on molecular orbital hydrogenation energies. *Theoret. Chim. Acta*, **28**(3), 213–222 (1973).

[129] F. Weigend, R. Ahlrichs: Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for h to rn: Design and assessment of accuracy. *Phys. Chem. Chem. Phys.*, **7**(18), 3297–3305 (2005).

[130] D. Andrae, U. Häußermann, M. Dolg, H. Stoll, H. Preuß: Energy-adjusted ab initio pseudopotentials for the second and third row transition elements. *Theoret. Chim. Acta*, **77**(2), 123–141 (1990).

[131] M. Sierka, A. Hogekamp, R. Ahlrichs: Fast evaluation of the coulomb potential for electron densities using multipole accelerated resolution of identity approximation. *J. Phys. Chem.*, **118**(20), 9136–9148 (2003).

[132] C. Peng, H. B. Schlegel: Combining Synchronous Transit and Quasi-Newton Methods to Find Transition States. *Chem. Eur. J.*, **33**, 449 (1993).

[133] G. van Rossum, F. L. Drake (eds): Python reference manual. PythonLabs, Virginia, USA, 2001; Available at `http://www.python.org`.

[134] D. L. Nelson, M. M. Cox: *Lehninger Principles of Biochemistry*. Springer, fourth edition (2004).

[135] G. Marsaglia: Choosing a Point from the Surface of a Sphere. *Ann. Math. Stat.*, **43**(2), 645–646 (1972).

[136] B. K. P. Horn: Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, **4**(4), 629–642 (1987).

[137] C. F. Karney: Quaternions in molecular modeling. *Journal of Molecular Graphics and Modelling*, **25**(5), 595–604 (2007).

[138] Y. Minenkov, Å. Singstad, G. Occhipinti, V. R. Jensen: The accuracy of DFT-optimized geometries of functional transition metal compounds: a validation study of catalysts for olefin metathesis and other reactions in the homogeneous phase. *Dalton Trans.*, **41**(18), 5526–5541 (2012).

[139] M. P. Waller, H. Braun, N. Hojdis, M. Bühl: Geometries of Second-Row Transition-Metal Complexes from Density-Functional Theory. *J. Chem. Theory Comput.*, **3**(6), 2234–2242 (2007).

[140] T. Ziegler: Approximate density functional theory as a practical tool in molecular energetics and dynamics. *Chem. Rev.*, **91**(5), 651–667 (1991).

[141] V. Jonas, W. Thiel: Theoretical study of the vibrational spectra of the transition metal carbonyls $M(CO)_6$ [M=Cr, Mo, W], $M(CO)_5$ [M=Fe, Ru, Os], and $M(CO)_4$ [M=Ni, Pd, Pt]. *J. Chem. Phys.*, **102**(21), 8474 (1995).

[142] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne: The Protein Data Bank. *Nucl. Acids Res.*, **28**(1), 235–242 (2000), www.rcsb.org.

[143] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, J. Thornton: CATH – a hierarchic classification of protein domain structures. *Structure*, **5**(8), 1093–1109 (1997).

[144] W. Sheffler, D. Baker: RosettaHoles: rapid assessment of protein core packing for structure prediction, refinement, design, and validation. *Protein Sci*, **18**(1), 229–239 (2009).

[145] B. Kuhlman, D. Baker: Native protein sequences are close to optimal for their structures. *Proc. Natl. Acad. Sci*, **97**(19), 10383–10388 (2000).

[146] G. Dantas, C. Corrent, S. L. Reichow, J. J. Havranek, Z. M. Eletr, N. G. Isern, B. Kuhlman, G. Varani, E. A. Merritt, D. Baker: High-resolution structural and thermodynamic analysis of extreme stabilization of human procarboxypeptidase by computational protein design. *J. Mol. Biol.*, **366**(4), 1209–1221 (2007).

[147] E. F. v. d. Eide, W. E. Piers: Mechanistic insights into the ruthenium-catalysed diene ring-closing metathesis reaction. *Nat. Chem.*, **2**(7), 571–576 (2010).

[148] V. Polshettiwar, R. S. Varma: Olefin Ring Closing Metathesis and Hydrosilylation Reaction in Aqueous Medium by Grubbs Second Generation Ruthenium Catalyst. *J. Org. Chem.*, **73**(18), 7417–7419 (2008).

[149] G. Occhipinti, H.-R. Bjørsvik, V. R. Jensen: Quantitative Structure-Activity Relationships of Ruthenium Catalysts for Olefin Metathesis. *J. Am. Chem. Soc.*, **128**(21), 6952–6964 (2006).

[150] F. Richter, A. Leaver-Fay, S. D. Khare, S. Bjelic, D. Baker: De Novo Enzyme Design Using Rosetta3. *PLoS ONE*, **6**(5), e19230 (2011).

[151] S. N. Ruzheinikov, J. Burke, S. Sedelnikova, P. J. Baker, R. Taylor, P. A. Bullough, N. M. Muir, M. G. Gore, D. W. Rice: Glycerol Dehydrogenase: Structure, Specificity, and Mechanism of a Family III Polyol Dehydrogenase. *Structure*, **9**(9), 789–802 (2001).

[152] J. Burke, S. N. Ruzheinikov, S. Sedelnikova, P. J. Baker, D. Holmes, N. M. Muir, M. G. Gore, D. W. Rice: Purification, crystallization and quaternary structure analysis of a glycerol dehydrogenase S305C mutant from *Bacillus stearothermophilus*. *Acta Crystallogr. Sect. D*, **57**(1), 165–167 (2001).

[153] B. Kuhlman, G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, D. Baker: Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science*, **302**(5649), 1364–1368 (2003).

[154] K. K. Ojo, T. L. Arakaki, A. J. Napuli, K. K. Inampudi, K. R. Keyloun, L. Zhang, W. G. Hol, C. L. Verlinde, E. A. Merritt, W. C. Van Voorhis: Structure determination of glycogen synthase kinase-3 from leishmania major and comparative inhibitor structure-activity relationships with trypanosoma brucei GSK-3. *Mol. Biochem. Parasit.*, **176**(2), 98–108 (2011).

[155] G. Scapin: Structural biology in drug design: selective protein kinase inhibitors. *Drug Discov. Today*, **7**(11), 601–611 (2002).

[156] E. Jakobsson, G. Alvite, T. Bergfors, A. Esteves, G. J. Kleywegt: The crystal structure of Echinococcus granulosus fatty-acid-binding protein 1. *BBA - Proteins Proteom*, **1649**(1), 40–50 (2003).

[157] K. M. McCulloch, C. Kinsland, T. P. Begley, S. E. Ealick: Structural Studies of Thiamin Monophosphate Kinase in Complex with Substrates and Products. *Biochemistry*, **47**(12), 3810–3821 (2008).

[158] H. K. Privett, G. Kiss, T. M. Lee, R. Blomberg, R. A. Chica, L. M. Thomas, D. Hilvert, K. N. Houk, S. L. Mayo: Iterative approach to computational enzyme design. *PNAS* (2012).

[159] T. Murata, H. Ishibuchi: MOGA: multi-objective genetic algorithms. In , *IEEE International Conference on Evolutionary Computation, 1995*, volume 1, 289– (1995).

[160] C. Riplinger, F. Neese: An efficient and near linear scaling pair natural orbital based local coupled cluster method. *J. Chem. Phys.*, **138**(3), 034106 (2013).

[161] K. H. Marti, M. Reiher: New electron correlation theories for transition metal chemistry. *Phys. Chem. Chem. Phys.*, **13**(15), 6750–6759 (2011).

[162] Y. Jin: A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, **9**(1), 3–12 (2005).

[163] J. Behler, M. Parrinello: Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Phys. Rev. Lett.*, **98**(14), 146401 (2007).

[164] J. R. Desjarlais, T. M. Handel: Side-chain and backbone flexibility in protein core design. *J. Mol. Biol.*, **290**(1), 305–318 (1999).

[165] J. J. Havranek, D. Baker: Motif-directed flexible backbone design of functional interactions. *Protein Sci.*, **18**(6), 1293–1305 (2009).

[166] J. J. Havranek, P. B. Harbury: Automated design of specificity in molecular recognition. *Nat Struct Mol Biol*, **10**(1), 45–52 (2003).

[167] T. E. Oliphant: Python for Scientific Computing. *Comput. Sci. Eng.*, **9**(3), 10–20 (2007).

ACKNOWLEDGEMENTS

The content of this master thesis was developed in the quantum chemistry group of Prof. Leticia González and the bioinformatics group of Prof. Ivo Hofacker at the Institut für Theoretische Chemie of the University of Vienna from February 2013 to November 2013.

First and foremost I offer my sincerest gratitude to Prof. Leticia González. Not only did she give me the opportunity to work in her group, as well as provide continuous guidance and support, but she also sparked my interest in theoretical chemistry with her quantum chemistry course.

This thesis would not have been possible without the aid of Dr. Philipp Marquetand. The helpful and inspiring discussions with him guided my progress and he never ran out of answers and patience when faced with my numerous questions. Moreover, his efficient and extensive proofreading played a major role in the fast convergence of my thesis to its final form.

The project in its entirety is the brainchild of Dr. Christoph Flamm, who introduced me to this exciting research topic encompassing various computational fields and who also was the main supporter of my bioinformatic endeavours.

Prof. Christian Becker and Dr. Aleksandr Kravchuk offered valuable advice based on their substantial practical experience, which proved to be extremely helpful in the enzyme design process. I am also very grateful for their willingness to embark on the task of synthesising the enzymes designed in this thesis.

The devotion of Dr. Markus Oppel and Jackie Klaura to their task of maintaining our systems allowed me to pursue my work without interruptions, as they kept my mind free of computational worries of the non quantum-chemical kind.

My office mates Leon Freitag, Sebastian May and Clemens Rauer provided me with many a fruitful discussion. Along with Federico Latorre, Rana Obaid, Martin Richter, Christoph Bauer, Dr. Daniel Kinzel, Edith Steinwidder, Dr. Juan Jose Nogueira Perez, Dr. Matthias Ruckenbauer, Kathrine Baumann and David Ferro, they are also responsible for creating a working atmosphere so enjoyable, I forgot I was at work at all.

And last but not least I want to thank my family and my friends for their patience and their continuous support not only during this thesis, but during the whole of my chemistry studies.

Curriculum Vitae

Personal data

Last name:	Gastegger
First name:	Michael
Address:	Institute of Theoretical Chemistry Währinger Str. 17, 1090 Vienna, Austria
Birth date/place:	July 24, 1988, Lilienfeld
Nationality:	Austrian

Academic career

2/2013 - 11/2013	Master thesis in the group of Prof. Dr. L. González Title: *De-novo* enzyme design for olefin metathesis
10/2011- 09/2013	Master studies in Chemistry, **University of Vienna**
06/2011	Bachelor thesis in the Group of Prof. Dr. L. Brecker Title: Isolation und Strukturaufklärung von Naturstoffen aus *Palicourea padifolia*
2007-2011	Bachelor studies in Chemistry, **University of Vienna**
06/2006	Matura, **Bundesrealgymnasium Lilenfeld**

Further qualifications

Language skills	German (mother tongue) English (fluent)