Thesis for the title Bachelor of Science in Chemistry

Implementation of Auxiliary Restraints for Relative Binding Free Energy Calculations in the 'common-core serial-atom-insertion' Framework `Transformato`

Supervised by: Univ-Prof. Dr. Stefan Boresch
Submitted by: Alexander Grasser

June 2022

Department of Computational Biological Chemistry
University of Vienna

Abstract

For protein-ligand affinity at binding sites, the relative (and absolute) binding free energy is arguably the most important indicator. Traditionally, calculating these energies is done using comprehensive, GPU-assisted "soft-core" modeling. The 'common-core serial-atom-insertion' framework `Transformato` instead allows to efficiently divide the problem into sequential fragments solvable by traditional MD calculations. However, issues arise when ligands leave the binding site during simulation. This work presents a possible solution by enabling both automatic and manual addition of restraints to the system.

Notes

The data compiled for this thesis may be accessed at GitHub under `https://github.com/agrass15268/AuxiliaryRestraintsTransformato/tree/data` or with the QR - Code to the right.

Source code and packaged versions of the `Transformato` package may be retrieved from its GitHub repository at `https://github.com/wiederm/transformato` or with the QR - Code to the right.

Table of Contents

1 Introduction

Almost since the inception of computer-assisted molecular modeling, a significant focus has been on the interaction of pharmaceutically active biomolecules. Designing and testing lead compounds to catalyze or inhibit biochemical processes is a major focus of research and development efforts, and a comparatively easy-to-access parameter with nevertheless good predictive capabilities has long been the relative binding free energy difference, especially with recent advancements in computational power [1].

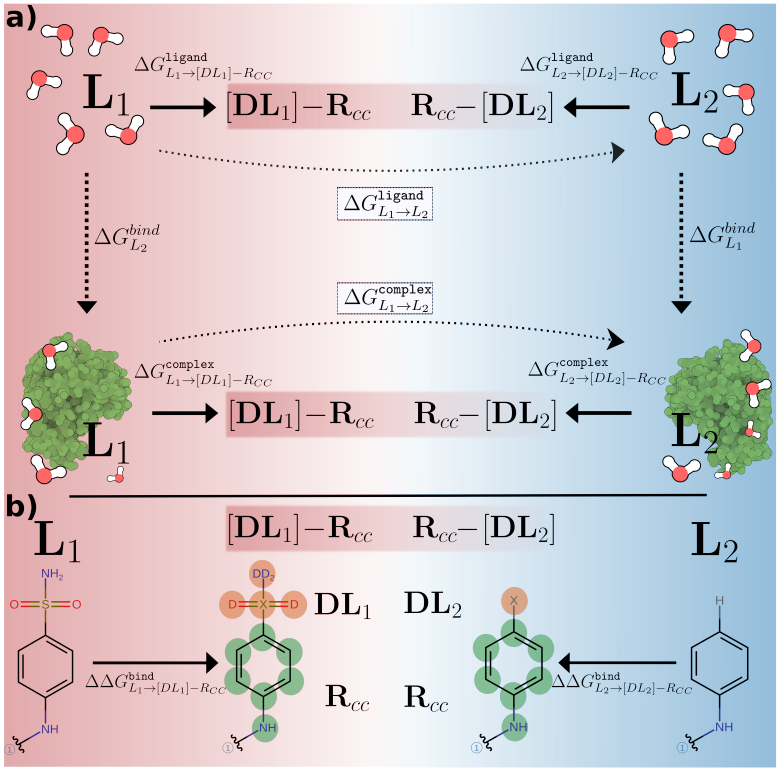

Figure 1.1: Overview of `Transformato`'s workflow
a) Comparison between alchemical transformations undertaken by traditional FES (dotted arrows) and the approach taken by `Transformato` (bold arrows). b) Example mutation path taken to transform two simple molecules into their common core. Green represents the common core, red those atoms that undergo mutation effects or are transferred into dummy atoms.

Recently, the package `Transformato` was introduced by Karwounopoulos, Wieder, Braunsfeld, and Boresch[2–4], implementing a novel approach to bypass these problems. Being a python package, `Transformato` is easy to install, use, and modify to adapt to system-specific challenges or support different MD engines. Instead of full alchemical

transformations of one compound into the other, it instead relies on computing a *common core* (CC) between the two physical endpoints. Summing up the energy differences between the CC and the endstates allows calculation of the free energy difference between the endstates themselves (see Fig. 1.1).

With this method, atoms outside the common core are transformed into so-called *dummy atoms*, a term for particles that, while still present in the simulated system, no longer have nonbonded interactions and are only kept in place by weak bonded interactions. If handled properly, these do not contribute to the calculated free energies[5].

These dummy atoms are introduced along several intermediate states, alongside various levels of scaling of nonbonded interactions. These intermediate states are then analyzed using the *multistate Bennett Acceptance Ratio*[6] (mBAR), an expanded version of Bennet's acceptance ratio[7], ideally giving a close approximation of the sought relative free energy $\Delta\Delta G$. However, mBAR requires topological overlap between the various intermediate states - a problematic proposition if the ligand is liable to abscond its binding site during the simulation timeframe. To prevent this, an additional module for `Transformato` called `restraints.py` was developed and integrated into `Transformato`. It allows the user to either request automatic restraints or define their own. These restraints as implemented act upon the center of mass of the selected ligand atoms, restraining them to their original positions relative to the protein, with variable force.

To suit a variety of sampling needs, both harmonic and flat-bottomed energy potentials may be used as energy functions for these restraints. This way, a ligand may be specifically allowed to sample the entire binding site without being constrained, yet never leave it. This allows for better sampling and longer runtimes even at increased temperatures, without fear of losing the ligand.

An attempt at stringent validation of the methods described here was not undertaken; however, a number of qualitative simulations with a variety of testing parameters were conducted and evaluated, finding no significant impact of reasonable restraints on the calculated free energy differences.

2 Theory

2.1 Theoretical Background

Simulating the dynamics of (bio-)chemical processes (referred to as *Molecular Dynamics* (MD)) in their entirety can be divided into two classes: calculations utilizing *Quantum Mechanics* (QM), where interactions are calculated according to Schrödinger's Equation, and *Molecular Mechanics* (MM), where calculations instead use a simplified mass-charge-force model to calculate potential energies. Combination methods (QM/MM) also exist, calculating the forces applied to the MM model using QM methods. While there is no doubt that methods utilizing QM can deliver more accurate results (especially if using "pure" QM and not using semi-empirical methods), computational costs prohibit their use in large biological systems ("large" in this context referring to even just a single protein and its ligand) [1]. Thus, for free energy calculations of the sort `transformato` is designed to accomplish, QM methods are unsuitable - which leaves MM.

This is not to say that QM has no place in MD simulations. QM is routinely used to improve geometries and improve calculation precision, with QM/MM methods allowing for enhanced accuracy while limiting computational costs, especially useful in complicated chemical environments without suitable force - fields. MD simulations based on MM also suffer from the fact that they cannot simulate the movement of electrons - meaning no bonds can break or form. This issue can be sidestepped by calculating the area of interest via QM [8]. But perhaps most importantly, the force fields used for MD simulations start out as QM calculations that are then fitted to experimental data [9]. Still, for this thesis, only MD based on Molecular Mechanics was used.

2.1.1 Basics of Molecular Dynamics Modeling

Since the discovery of cells, the inner workings of biological systems have always been a mystery science has raced to solve. While, of course, physical methods have allowed some degree of monitoring biochemical processes, these have significant limits - most glaringly being limited to physically-present structures. With Molecular Dynamics (MD) on the other hand, it is entirely possible to observe molecular interactions as they would happen in reality - at the leisure and convenience of the observer, with an unmatched resolution and clarity, and without the need to synthesize a single atom - the so-called *in silico* - approach.

In general, MD aims to simulate the movement of atoms in a molecular system over time, governed by the physical interactions present in the molecule [10]. Unlike QM, which tries to calculate interactions using various approximations or even solutions of the Schrödinger equation, MM simulations are inherently uncertain. Forces are approximated by what are essentially the mechanical principles of Newton: Atoms are perfect spheres, with mass and charges, and all their interactions are defined by a set of forces governing the interactions of specific types of atoms. As such, the kinetic energy K of

an MD system is simply the kinetic energy of its atom masses m_i at speed v, using the equation familiar from Netwonian mechanics (Eq. 2.1), while their potential energy is defined by the potentials governing their interactions.

$$K = \sum_i \frac{m_i v_i^2}{2} \tag{2.1}$$

Force Fields For almost all force fields, forces in MM simulations may be divided into two categories - *bonded* and *nonbonded* interactions. These are often used synonymously with *intramolecular* and *intermolecular* forces, but are far from the same - it is quite possible, and especially in biological systems the rule more than the exception, that nonbonded but intramolecular interactions contribute a significant part to the protein structure, with electrostatic forces and the hydrophobic effect being especially important for protein folding and thus functionality [11].

Bonded forces are generally given explicitly for each atom type. They are:

- Bonds (across two atoms)
- Angles (the angle across three atoms)
- Dihedrals (the angle across four atoms)
- Impropers (also called *improper dihedrals*, the angle of a fourth atom being out-of-plane with the other three)
- Urey-Bradley (sometimes grouped with the angle terms; describes noncovalent bonds over larger distances)

All these are stored in *parameter files*, which contain the force parameters for each of these.

As apparent in figure 2.1, force-field parameter files are structured very simply: The first few columns define the "atom types" for which a given set of parameters is valid. The others define the numeric value of the various parameters for the potential; for simple bonds for example the first value denotes the force constant, and the second the equilibrium distance. These values are inserted into a harmonic potential and then used to calculate the force.

In a MD system, the combined potential energy is simply the sum of bonded and nonbonded potential energy terms, both of which are dependent on the relative positions ($\vec{r}$) of the atoms. Transformato uses the Charmm General Force Field[9].

The sum for bonded potentials is the sum of the individual potentials, whereas the nonbonded potentials *generally* consist of electrostatic (Morse) and VdW (Lennard-Jones) potentials.

However, these equations do not offer the information we require. What we want is the movement of atoms across the time dimension (thus Molecular *Dynamics*). However, these potentials are a significant part of the way there. To simulate a MM system, all atoms are assigned random initial velocities. For each step, the forces resulting

```
BONDS
CG1N1  CG2R51  375.00     1.4220 ! DCG, yxu, RNA
CG1N1  CG2R61  345.00     1.4350 ! 3CYP, 3-pyridine (PYRIDINE pyr-CN)

ANGLES
CG2R51 CG1N1  NG1T1    40.00    180.00 ! DCG, yxu, RNA
CG2R61 CG1N1  NG1T1    40.00    180.00 ! 3CYP, 3-pyridine (PYRIDINE pyr-CN)

DIHEDRALS
NG1T1  CG1N1  CG2R61 CG2R61  0.0100 2    0.00 ! CNP2, by ac_aa
NG1T1  CG1N1  SG311  CG321   0.0060 1    0.00 ! XCN, by ac_aa

IMPROPERS
CG2D1  CG331  NG2D1  HGA4      25.00  0     0.00 ! SCH1, xxwy
CG2D1  CG331  NG2P1  HGR52     18.00  0     0.00 ! SCH2, xxwy
```

Figure 2.1: Illustrative excerpt from a parameter file created by CGenFF with a few definitions for bonds, angles dihedrals and impropers each.

from bonded and nonbonded interactions are calculated and applied to these atoms in a manner resembling classical mechanics. The atoms are then moved as dictated by their momentum and the timestep chosen, and the cycle repeated. QM/MM differs from classical MM in the way the forces are calculated: while the interactions are still governed by classical mechanics, the force contributions are derived form quantum mechanical calculations rather than the simple force potentials used in MD.

Following these steps results in positions for each atoms for every step you calculate. By combining these into a single file you get a *trajectory*, containing the movements of your atoms along the entire simulation. However, trajectories typically do not contain every single calculated position, but rather only every n-th step, as trajectory files would otherwise be prohibitively large.

2.1.2 Relative and absolute binding free energies

Calculating the free energies from these trajectories, however, requires additional steps. For our purposes, we need to differentiate two types of free energy differences:

- *absolute* free energies (ΔG_{bind}); Here, the ligand energy difference is between the ligand bound in complex, and the ligand free to explore all other possible configurations.
- *relative* free energies ($\Delta\Delta G_{bind}^{A\to B}$): Here, the ligand energy difference is compared between two specific states A and B.

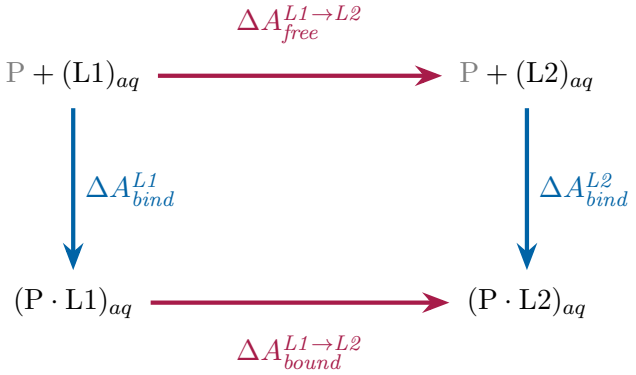

Figure 2.2: A classic thermodynamic cycle to calculate free binding energy differences $\Delta A_{binding}$ between ligands L_1 and L_2. The physical path is marked in blue; the alchemical one in wine red.

Calculating them typically exploits thermodynamic cycles. In Fig. 2.2 you can see a typical example of such a cycle to calculate relative *binding* free energies: two different Ligands L_1 and L_2 are in aqueous solution with the same Protein P. The quantity of interest is the relative binding free energy $\Delta\Delta A_{bind}$, which can easily be calculated by using the nonphysical - *alchemical* transformations marked in red (Eq. 2.2). While it would, theoretically, be possible to calculate ΔA_{bind}^{L1} directly, doing so still requires massive amounts of computational power and research time, making it prohibitive for typical usage [12].

$$\Delta\Delta A_{bind} = A_{bind}^{L2} - A_{bind}^{L1} = A_{bound}^{L1\to L2} - A_{free}^{L1\to L2} \tag{2.2}$$

For the alchemical transformations however, all that needs to be done is to replace one ligand binding to the protein with the other. Doing so directly, however, does not work (well). Calculating free energies requires *phase space overlap*. *Phase space* in this context refers to the total allowed set of positions and momenta of the system. Two completely different ligands will overlap barely, if at all - giving error bars bigger than the calculated energy difference. Thus, the amount of changes in the calculation needs to be reduced by introducing *intermediate states*. Free Energy differences are then not calculated between the endstates, but rather as sum of energy differences between the endstates (Eq. 2.3). This allows for much more overlap between the individual states, leading to much smaller margins of error in total.

$$\Delta A_{1,K} = \sum_{i=1}^{K-1} \Delta A_{i,i+1} \tag{2.3}$$

The same approach also enables calculating relative *solvation* free energies. Here, the "physical" path leads from an unsolvated compound to a solvated one, whereas the alchemical pathways again lead through simply changing the compound (Fig. 2.3).

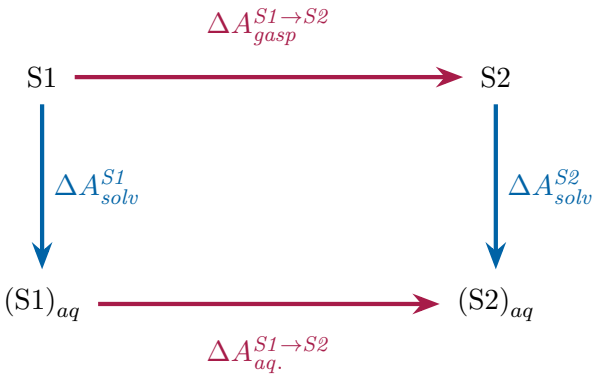

Figure 2.3: A classic thermodynamic cycle to calculate free solvation energy differences $\Delta\Delta A_{solvation}$ between two substances S_1 and S_2 in gaseous phase and aqueous solution, with physical paths in blue and alchemical paths in wine red.

2.1.3 Generation of intermediate states and dummy atoms

The simplest method of generating intermediate states as used in Eq. 2.3 is to linearly "mix" the potentials of the endstates 0 and 1 U_λ from $U_0 \to U_1$, with λ being known as the *coupling parameter*. Notably, λ only affects a subset of potentials (usually nonbonded interactions, meaning charges and Lennard-Jones-interactions, will be most heavily affected), while most of the potential remains unaffected. The total potential for a given intermediate state i as required by Eq. 2.3 is thus as given by Eq. 2.4, usually with $U_0; U_1 << U_{environment}$.

$$U_i^{total} = (1-\lambda)U_0 + \lambda U_1 + U_{environment} \tag{2.4}$$

This kind of linear scaling, however, creates a number of problems; most pressingly a distinct lack of phase space overlap required for analysis resulting from the van-der-Waals endpoint problem (see section 2.1.5). `Transformato` does *not* use these kinds of "λ-scaling" methods, instead opting for an approach called "serial atom insertion" (see section 2.2.1).

An even more egregious problem arises when the number of atoms changes between endstates: Typical implementations of statistical thermodynamics as required for MD do not allow for changing the number of particles in the system. To sidestep this problem, *dummy atoms* are introduced by turning off all interactions but their bonded terms - either just for surplus atoms (known as the *single topology approach*) or by having both topologies fully present in all states, with the currently unused one handled as dummy atoms (known as the *dual topology approach*). It has been shown that, used correctly and provided they remain well-defined in position and orientation, their influence on calculated energies is zero as their terms in $\Delta\Delta A$ cancel each other out[5]. This allows the system to retain a constant number of particles.

2.1.4 Intermediate state analysis, postprocessing and calculation of free energy

But one question has so far remained unanswered: how to actually get to $\Delta A_{i,i+1}$ from Eq. 2.3? FES conducted by the methods above typically rely on either Thermodynamic Integration (TI) [13] or Bennett's Acceptance Ratio (BAR) [6, 7] to analyse the results from the various intermediate states, both of which start from the identity of free energy (Eq. 2.5), which defines the energy difference between two states i, j as dependent on the logarithm of the *partition functions* $Q_{i,j}$.

$$\Delta A_{i,j} = -k_B T \cdot \ln \frac{Q_i}{Q_j} \tag{2.5}$$

These partition functions describe the thermodynamic properties of the system dependent on the used *ensemble*, the set of possible states the system can have while exchanging energy with the outside world in a state of thermal equilibrium. For this thesis, all simulations were conducted using an **NPT** ensemble, meaning the **number** of particles, and the average **pressure** and **temperature** of the system were held constant. Eq. 2.6 is the partition function for the *canonical* or NVT ensemble (the derivation for the NPT ensemble follows the same pattern, but gives Gibbs instead of Helmholtz free energies). As can be seen, the function integrates over the entire phase space volume Γ, allowing derivatives to express a single expected value (represented by $\langle\rangle$).

$$Q = \int_\Gamma e^{-\frac{U(\vec{q})}{k_B T} d\vec{q}} \tag{2.6}$$

Thermodynamic Integration Of the two methods, TI is the older one. It relies on simply integrating the total potential energy dU over the coupling factor $d\lambda$ as shown in Eq. 2.7. However, as we cannot calculate the integral in a continous manner for $\lambda = [0 \to 1]$, a numeric approximation as shown in Eq. 2.8 is required, where the difference of free energy is given as the sum of the integral for actually calculated states K, each weighted according to the weight factor w_k. w_k may be computed a number of different ways; the common all-purpose method is the trapezoidal method. The weighting method should be choosen carefully, as it has significant impact on result quality [14].

$$\Delta A = \int_0^1 \left\langle \frac{dU(\lambda, \vec{q})}{d\lambda} \right\rangle_\lambda d\lambda \tag{2.7}$$

$$\Delta A \approx \sum_{k=1}^{K} w_k \left\langle \frac{dU(\lambda, \vec{q})}{d\lambda} \right\rangle_k \tag{2.8}$$

While the calculation is rather simple, there are a few significant drawbacks to TI. First, it requires the computation of $\frac{dU}{d\lambda}$ for all $\vec{p}$, which requires a MD engine with facilities capable of the appropriate derivations. Secondly, TI does badly with sudden steps in the integral, requiring either a highly efficient numeric integrator, the use of

techniques increasing phase space overlap (such as soft-core potentials) or resulting in large uncertainties. It should be noted that this, alongside a lack of continuous, defined λ makes it unsuitable for the Serial-Atom-Insertion approach used in `Transformato` (see section 2.2.1).

Bennett's Acceptance Ratio BAR and its multistate extension mBAR on the other hand do not face such limitations, at the cost of significantly increased complexity. First a set of weighting functions $\alpha_{i,j}(\vec{q})$ are generated for the overlap space of the state variables $K \times K$. The actual derivation then starts from the same identity of free energy (Eq. 2.5) as TI, but by introducing the weighting factor, an equivalency between Q_i and Q_j can be expressed (Eq. 2.9).

$$Q_i \langle \alpha_{i,j} e^{-\beta U_j} \rangle_i = Q_j \langle \alpha_{i,j} e^{-\beta U_i} \rangle_j \tag{2.9}$$

Introducing the empirical estimator $\langle g \rangle_i = N_i^{-1} \sum_{n=1}^{N_i} g(\vec{q_i}, n)$ allows to write the equivalency in Eq. 2.9 as a sum (Eq. 2.10). Through the use of a *numerical bridge integrator* [15] it is possible to arrive at the free energy equation 2.11. Notably, this equation actually gives a *single* free energy instead of an energy difference; however the use of an undefined additive constant means that all calculations need to be put in reference to the energy of one of the states (typically the starting endpoint), again arriving at an energy difference.

$$\sum_{j=1}^{K} \frac{\hat{Q}_i}{N_i} \sum_{n=1}^{N_i} \alpha_{i,j} e^{(-\beta U_j(\vec{q_i},n))} = \sum_{j=1}^{K} \frac{\hat{Q}_j}{N_j} \sum_{n=1}^{N_j} \alpha_{i,j} e^{(-\beta U_i(\vec{q_j},n))} \tag{2.10}$$

$$\hat{A}_i = -\beta^{-1} \ln \sum_{j=1}^{K} \sum_{n=1}^{N_j} \frac{e^{(-\beta U_i)}}{\sum_{k=1}^{K} N_k e^{\beta \hat{A}_k - \beta U_k}} \tag{2.11}$$

Unlike TI, BAR and mBAR are able to give accurate results even with significant changes to K and correspondingly little phase space overlap. Notably, it is able to deal with the significant changes caused by the Serial-Atom-Insertion method used by `Transformato` and is thus used to calculate the free energies of interest.

2.1.5 Soft-core potentials and the van-der-Waals endpoint problem

In a standard MD simulation, the Lennard - Jones potential U_{LJ} is given by formula 2.12, with r as interparticle distance, and $A; B$ as atom-type dependent constant factors.

$$U_{LJ}(r, \lambda) = (1 - \lambda)(\frac{A}{r^{12}} - \frac{B}{r^6}) \tag{2.12}$$

This usually works rather well - the shape of the potential ensures that no two LJ particles inhabit the same spot. However, free energy calculations, for the reasons discussed in section 2.1.3, usually require the presence of *dummy atoms* - which obviously do not have Lennard - Jones interactions. It's here where problems may now occur - as there are no

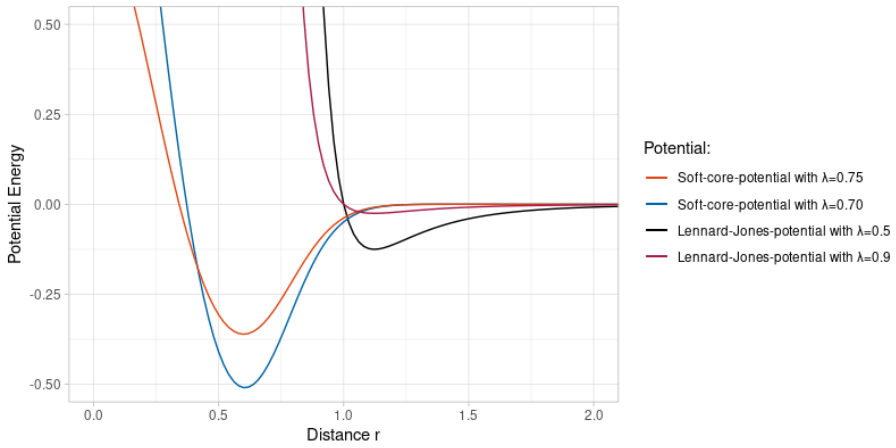

Figure 2.4: Comparison between behaviour of standard Lennard-Jones and soft-core potentials for various states of λ. Visibly: For the standard LJ potential, well size decreases with increasing λ, while steepness of the potential wall increases. The reverse is true for soft-core potentials, which also remain defined at $r = 0$

LJ interactions, atoms may move freely into each other. This may not seem like such a significant problem - but forces still need to be calculated. Should particles manage to inhabit the exact same position, this would result in a division by zero - problematic, but most modern MD software can handle this.

Both more problematic and more common, however, is the case that two particles get very close to each other. This is especially common in simulations with linear coupling between λ and the LJ potential, as for low values of λ the interaction range may decrease to such an extent that a particle may skip the entire "well" of the potential, and directly encounter its "wall", leading to extremely high forces being applied [16]. (see Fig. 2.4). In this case, (i) the derivative $\langle \frac{\delta U}{\delta \lambda} \rangle$ may diverge, and (ii) because of the sudden change in energy the entire simulation may become unstable. This is known as the *van-der-Waals endpoint problem* (or *-catastrophe*, for the more dramatically inclined).

To avoid this problem, soft-core potentials dependent on the coupling factor λ were introduced as shown in Equation 2.13[17]:

$$U_{LJ}^{SC}(r, \lambda) = (1 - \lambda)\left(\frac{A}{(r^2 + \lambda\delta)^6} - \frac{B}{(r^6 + \lambda\delta)^3}\right) \tag{2.13}$$

These mostly solve the problem, as there can now be no division by zero and the fractions remain finite in all circumstances. They can, however, be computationally expensive - especially for MD engines lacking native support for them[18]. It is important to note that `Transformato` for this reason does **not** use soft-core potentials, but rather the "serial-atom-insertion" approach (see section 2.2.1).

2.2 The common-core serial-atom-insertion framework `Transformato`

To avoid these problems and reduce general computational cost of FES, the common-core serial-atom-insertion framework `Transformato` was conceptualized and implemented. Being engine-agnostic - its outputs theoretically being computable with any of the widely-used Molecular Dynamics engines (openMM, CHARMM, AMBER, GROMACS, etc.). Currently, while the outputs are computable with any MD engine, `Transformato` only provides output scripts for openMM and CHARMM. Within these limitations however, it allows straightforward calculation of relative binding and solvation free energy differences in the form of a single, easy-to install python package.

2.2.1 Theoretical Principles

`Transformato` is built on two distinct theoretical pillars: The *Serial Atom Insertion* approach to generating intermediate states, and the *Common-Core* approach to handling thermodynamic cycles, together referred to as the "serial-atom-insertion common-core" method (SAI/CC).

Serial Atom Insertion (SAI)[16] refers to an alternative method of avoiding the van-der-Waals-Endpoint problem without relying on the soft-core potentials discussed in section 2.1.5, developed by Boresch and Bruckner in 2011. This allows for a significant decrease of computational cost in MD engines without native, GPU-assisted soft-core modelling, at the drawback of having significant, stepwise changes to the states present in phase space, making it unsuitable for analysis via Thermodynamic Integration. Central to this method is to turn off the changed atoms in successive fashion, rather than scaling all of their interactions as one λ-dependent potential to generate the intermediate states. Thus, rather than depending on potentially very small λ values (Eq. 2.14 for the simplest case, outlined in section 2.1.3), the total potential energy is calculated by having the each atom turned on or off be its own intermediate state, with no intermediate scaling of the nonbonded parameters (Eq. 2.15, for a molecule in solution with M Lennard-Jones particles, itself containing m particles with $u_{i,j}^{LJ}$ denoting the Lennard-Jones interaction between two particles).

$$U(\lambda) = \sum_{1 \leq i < j \leq M} u_{i,j}^{LJ} + \lambda \sum_{1 \leq i \leq M} \sum_{1 \leq l \leq m} u_{i,j}^{LJ} \qquad \lambda = [0, 1] \tag{2.14}$$

$$U(k) = \sum_{1 \leq i < j \leq M} u_{i,j}^{LJ} + \sum_{1 \leq i \leq M} \sum_{1 \leq l \leq k} u_{i,l}^{LJ} \qquad k = [0, m] \tag{2.15}$$

The term *Common Core* [3, 4] refers to the atypical alchemical path taken by `Transformato`. Whereas traditional FES transform the endstates directly into each other, `Transformato` instead transforms each of them into their maximum common topology - the *Common Core*.

Routing via the common core is useful, as the complete free energy difference between two ligands on the alchemical pathway (illustrated in Fig. 2.2) can be expressed as the

sum of the individual free energy differences (Eq. 2.16).

$$A_{bound}^{L_1 \to L_2} = \Delta\Delta G_{bound}^{L_1 \to L_2} = \Delta\Delta G_{bound}^{L_1 \to [D_{L_1}] - R_{CC}} - \Delta\Delta G_{bound}^{L_2 \to [D_{L_2}] - R_{CC}} \tag{2.16}$$

To reach the common core, the *mutation path* is generated by the program from each ligand L_1, L_2 to the CC R_{CC}, along which atoms not belonging to the CC are decoupled and transformed into dummy atoms. First, electrostatic interactions are turned off, followed by Lennard-Jones interactions of first hydrogen, then separately non-hydrogen atoms successively. The last atom connecting the *dummy regions* DL_1, DL_2 with the CC is called the *terminal junction* X. Interaction terms involving X are scaled linearly to have the bond specifications of L_1 match those of L_2. As the SAI/CC requires redistribution of charges[1], this atom retains some Lennard-Jones interaction and forms the anchor point for the dummy region. Each stage of this mutation forms an intermediate state similar to those used with λ-scaling methods, though not dependent on a numeric factor. After each intermediate state is simulated, $U_{i,j}$ may be calculated in postprocessing, allowing the calculation of free energy differences via mBAR. This is done both for the ligand in complex with the protein, as well as for the ligand in simple solution, allowing calculation of $A_{bound}^{L_1 \to L_2}$ via exploitation of the cycle in Fig. 2.2.

It should be noted the two methods are independent from each other; it is quite possible to compute a direct transformation using SAI or to use the CC approach alongside soft-core potentials, with increasing adoption of native soft-core methods in MD engines making the latter especially a useful proposition. However, this is as of yet not implemented in `Transformato`.

2.2.2 Theoretical considerations for Restraints in `Transformato`

While `Transformato` yields excellent results for almost all test systems, problems may arise in unstable systems with a ligand prone to leaving the binding site of its associated protein within the timeframe of the simulation. This may even be a problem in comparatively stable systems, assuming a larger-than-usual timeframe is desired. To circumvent this, restraints need to be applied within the system, keeping the ligand within the binding site while still allowing sampling within it. However, irrespective of other considerations, the addition of restraints must not influence the calculated free energy differences, as this would render the entire exercise pointless. Restraints as implemented act as bonded force, and are a simple addition to the sum forming U_{bonded}. Given that the free energy is directly dependent on U via the partition functions (Eq. 2.6) and the pathway energy results from the subtraction of the L_2 pathway from the L_1 pathway (Eq. 2.16), this means that $U_{L_1}^{restraints}$ must be equal to $U_{L_2}^{restraints}$ for the restraints to cancel each other out.

However, this is not possible for restraints involving the entire system. In case of a simple harmonic restraint as given by Eq. 2.17), with r the distance between the restrained particles - one anchored on the ligand, one anchored to the protein -, r_0 the

[1] The sum partial charge of the terminal junction at the end needs to be 0 to cancel itself out across the thermodynamic cycle, see [2, 4] for details

equilibrium distance, and k the force constant, the amount of energy added to the system is proportional to the movement of the restrained systems vis-a-vis each other.

$$U^{rest} = 0.5 \cdot k(r - r_0)^2 \tag{2.17}$$

To achieve $\langle U^{rest}_{L_1} \rangle == \langle U^{rest}_{L_1} \rangle$, the first condition is the same number of restraints in the system. However, as $\langle U \rangle$ is dependent on r, the expectation value for r also must not change. As r is dependent on the atom interactions and thus its given force field parameters, that means that the restraint needs to be connected to the same atom types (or groups of atom types) at both endstates. The current implementation in `Transformato` achieves this by restricting its restraints to the common core, as that represents a useful list of atoms that will both remain constant during the mutations and have the same atom type in both ligands, barring the terminal junction.

However, the common core does not exist in a vacuum, and r is dependent not only on movement by the ligand, but that of the protein as well. To satisfactorily keep ligands restrained in the binding site while still allowing for low enough force constants to facilitate sampling, current automatic restraints anchor to protein carbon-alphas near the ligand. These are affected by the ligand, with different ligand structures producing different effects. These differing interactions will also have differing effects on the movement of the common core structures, further diverging $\langle (r - r_0) \rangle$ between L_1 and L_2. While the anchoring of the restraints exclusively to alpha carbons is an attempt to minimize this and contributions are likely to be small, is is nevertheless an inaccuracy introduced into the calculations.

To minimize this inaccuracy, restraints may simply be turned off for the endstates. With mBAR, as the $\hat{A}_i$ is formed from the sum differences between two adjacent states (Eq. 2.11), as long as the restraints are introduced away from the endstates, their energy contribution is considered in $\Delta\hat{A}_1 \rightarrow \hat{A}_2$ at both sides instead of being part of the initial system. As the contribution at the common core is necessarily the same, the terms thus cancel each other out. `Transformato` provides the keyword `scaling` for this, which scales U^{restr} linearly across the first four intermediate states, with the endstates receiving no contribution at all.

On the topic of contribution, harmonic restraints as in Eq. 2.17 suffer the disadvantage of always having a contribution to the total potential, as given the high numerical precision of simulations along with initial velocities assigned at the start of the system means that $r - r_0$ never evaluates to 0. Similarly, the harmonic shape of the restraint means that a restraint capable of stopping escape from a binding site will have significant contributions even within that binding site, discouraging sampling and possibly introducing errors into the energy calculations that way. To alleviate this, it is recommended to instead use one of the provided flat-bottom potentials, e.g. Eq.2.18.

$$U^{rest} = k \cdot step(|r - r_0| - wellsize) \cdot (r - r_0)^2 \tag{2.18}$$

With r once again representing distance, r_0 being the equilibrium distance and k the force constant. New additions are the *wellsize* and a *step* function. The wellsize represents a radius around the equilibrium distance which the restraint is free to move around in.

As long as the expression $|r - r_0| - wellsize$ evaluates negative, $step(x) = 0$. Should it however evaluate positive, meaning that the restraint has left the well, the step function evaluates to 1 and the standard harmonic restraints are applied. This allows definition of fairly stringent restraints without impact on sampling within the binding site.

2.2.3 Installation and Usage

Installing `Transformato` is fairly straightforward: it requires a working installation of `conda` and `python`. Using `git clone`, download the package from the repository[2] and run `python setup.py install`. Install the `conda` environment called `fep` consisting of `Transformato` and all its dependencies, located in `transformato/test_environments/`. Activate the environment and you're done.

To calculate free energies, retrieve a PDB containing your ligand-protein complex for one endpoint, then modify the ligand to suit your purposes, using e.g. CHARMM-GUI's[3] ligand builder. For each endpoint, solvate once the complex including the ligand and once just the ligand using e.g. CHARMM-GUI's Solvation Builder. Take the output folders, equilibrate them and name the 'complex' and 'waterbox', respectively. You will then need to create a config.yaml file containing the names of your structures and parameters you want your simulation to have - any restraints you wish to apply must also be defined here. The simplest case for specifying restraints is simply done by adding the keyword `restraints: auto` to the *simulation* part of the configuration file. This restrains the entire ligand backbone via its center of mass to the center of mass of the protein backbone. Inside the code, this is referred to as `simple` mode. Another possibility is the generation of restraints on the extremities of the ligand using the `extremities=[int]` keyword. This algorithmically selects those carbon atoms furthest from the center of mass (and each other), and restrains these and their surroundings to the protein, thus cutting down on rotational freedom. The downside of this is that it requires some prior knowledge of the ligand's shape - not usually a problem, but potentially in large-scale applications. Of course, this also reduces sampling volume even further than the simple restraint. Lastly, it is also possible to manually define restraints in addition to - or instead of - the automatic ones, using the keyword `restraints: manual` and defining manual restraints below. This allows full use of the MDAnalysis[19, 20] selection syntax, limited only by the common core on the ligand.

To actually create the simulation runs, you will need a submit script, templates of which are readily available at the repository. These files sequentially load the .yaml config and the structure topologies, then propose a common core. After this point, you may add (or remove) atoms from the common core. Afterward, `Transformato` will propose a mutation route. If it looks satisfactory, it will then create several folders containing intermediate states, each a self-contained simulation. Simulate these (as this is the most computationally intensive step, it is highly recommended to use a cluster for this step) and run the analytics script (a cluster is also recommended here). In the end, you should

[2] `https://github.com/wiederm/transformato`
[3] `https://www.charmm-gui.org/`

receive the energy difference along with an uncertainty interval (e.g. `Free energy to common core: 12.02382 [kT] with uncertanty: 0.5843413703 [kT]`).

Significantly more extensive documentation of `Transformato`'s abilities and features is available through the package documentation[4], along with sample input and data files.

[4] at `https://wiederm.github.io/transformato/`

3 Methods

3.1 The preexisting codebase of `Transformato`

`Transformato` does not contain a main program or executable. Instead, as alluded to in section 2.2.3, it is subdivided into a number of modules that mostly operate independently of each other and are called by the user script on an as-needed basis. The most important of these modules are:

utils.py The first module to be called for RFE calculations, `utils.py` handles miscellaneous functions. Importantly, it handles user I/O, providing the facilities needed to read and (in part) interpret the `config.yaml` provided by the user. It also handles postprocessing of trajectories.

mutate.py As the name indicates, `mutate.py` is mainly responsible for calculating the necessary steps to transform the endstates into the common core. Its main tool for doing so is the `ProposeMutationRoute` class, which also provides user-interface commands to visualize and manipulate the common core (as the CC generated by `Transformato` may not always be the ideal one, or one suitable to the task at hand). On a technical level, this is accomplished by generating a new *Protein Structure File* (.psf, representing the physical locations of and atom types of the molecule's atoms) and *Parameter Files* (generally copies of the original as in Fig. 2.1 , with atom types added and parameters modified to represent the mutation at hand) for the various mutations, a set of each representing a new intermediate state.

state.py While `mutate.py` does most of the heavy lifting in calculating and performing the mutations, it is `state.py` and its `IntermediateStateFactory` class that actually writes the generated mutation steps to the intermediary folders, taking the .psf and .prm data provided by `mutate.py` along with the other necessary data, parameter and script files copied from either `Transformato` or the original provided structure, creating a folder with an entirely self-contained simulation, independent of `Transformato`.

analysis.py Lastly, `analysis.py` does not get called during simulation, but rather exists to extract the relative free energy difference from the trajectories post-processed by `utils.py`. It does this using the pyMBAR module developed by Shirts and Chodera [6, 21]. As secondary function, it also provides overlap and potential energy plots used for troubleshooting and quality assurance.

A further number of small, miscellaneous modules (`systems.py`, `charmm_factory.py`...) exist, providing helper functions to the above-mentioned modules or for testing.

3.2 Implementation of restraints into `Transformato`

The process of applying restraints was divided into three parts:

1. Processing user input: the general demand for restraints, and parameters for these restraints
2. Finding suitable binding sites and generating the atom selections for the restraints
3. Generating an openMM Force object and applying this force to the actual simulation

These processes were complicated by the fact that the script that generates the intermediate states for processing (and thus has access to user input) is separate from those that do the actual simulations (which need to be independent to be able to run on distributed computing networks).

As much code as possible was exiled into a separate module file called `restraints.py`, with only minimal changes to `state.py` and `mutate.py` being necessary.

Currently, restraints are only available as proof-of-concept for openMM. OpenMMs *CustomCentroidBondForce* was chosen as the basis for all restraints, meaning the restraint forces act upon the atom group's center of mass. To facilitate analysis of ligand and protein structures, MDAnalysis[19, 20] was used.

3.2.1 Processing user input

To leverage the existing codebase as well as keeping with the established design principles of `Transformato`, additional user input was restricted to the configuration `.yaml` already required; to ensure both backward compatibility and prevent accidental use of restraints all commands to use restraints are purely optional. If a user is unaware of the possibility of restraints, they will not run into the danger of accidentally using them. Further, there is no additional workflow in the users' submit script - all of it is done during `utils.py::load_config_yaml()` automatically.

Due to the structure of the existing code, no changes to `utils.py` were necessary - all relevant information was available for further processing immediately. During the run of the initial submit script, if `Transformato` finds defined restraints in the configuration file, a separate `restraints.yaml` is created in each of the intermediate states, containing the restraint information passed through from the configuration file along with the atoms in the common core and possible scaling effects.

These `restraints.yaml` are then read in by the `openmm_run.py` during simulation startup. Their mere presence informs the simulation to run the code applying restraints. It thus starts evaluating the restraints specified, cross-checks them against the atoms in the common core, and creates/applies the force.

3.2.2 Generation of restraints

While ideally every system would be run after careful, manual inspection, using artisanally crafted restraints, perfectly weighed and modified to the specific system, the reality is that for most purposes, ease of use and speed of "good-enough" restraints is paramount. For this reason, a number of facilities for generating automatic restraints were implemented. These generally come in two flavors: the first is the "simple" restraint, which anchors the combined carbon backbone of the ligand to the surrounding alpha-carbons (by default: alpha-carbons between 5 and 15 angstroms from the ligand). This effectively restrains the ligand in the binding site, while allowing for movement of said binding site relative to the protein as a whole without affecting the restraint.

The second flavor are the 'extremities' - restraints. Instead of acting upon the entire ligand, these act only upon N areas of ligand carbons furthest from the ligand COM, restraining them to the protein alpha carbons that surround them. This is significantly more restrictive than the simple restraint as rotational movement is now limited as well.

For all of this, the automatic facility assumes that the ligand is already near its binding site at the protein. Should this not be the case, a molecular docking step must be prepended to the RBFE calculation to find a suitable starting position for the ligand. If for one reason or another, such a starting position cannot be found, no sensible RBFE can be calculated, as it is intrinsically linked to the site-ligand interaction.

Technical details On a technical level, when running the simulation the `restraints.yaml` created by the Intermediate State Factory is read in. Then, molecular types and networks are analyzed using MDAnalysis. In the first step, the ligand carbons are cross-referenced against the CC provided by the .yaml, and any carbon not found within is discarded. These as a whole then constitute the initial ligand group `group1`. For a simple restraint, the program then takes any protein alpha-carbon from within 0.5 to 1.5 nm of the carbon and uses these as the anchor group (internally referenced as `group2`.

For an "extremities" - type restraint, the process is slightly more complicated. The algorithm first selects the carbon within `group1` that is furthest from the group's center of mass (C1). It then selects the carbon furthest from that carbon (C2). It then retrieves additional carbons up to the amount specified by the `n_extremities` value set by the user by selecting the carbon where the sum distance to all previously selected carbons is highest, repeating this process for every carbon it requires.

Once all extremity carbons have been selected, the individual restraints are created. Every extremity gets its own restraint, with `group1` consisting of the selected extremity carbon and ligand carbons in close proximity, and `group2` being comprised of protein alpha carbons in a spherical layer between 0.3 and 1 nm around `group1`. Note that these groups exist individually for each restraint, and each extremity is endowed with its own restraint. As such, a value of `n_extremities = 4` would produce 4 restraints with 8 groups total, each applying forces independently of the others. Also, note that no cross-check between these restraints is done; a carbon may very well be a member of multiple restraints at the same time.

Manual restraints Manual restraints offer expanded functionality compared to automatic restraints. `group2` is transcribed directly from the user-provided selection string, `group1` is similar but undergoes the same CC adjustment as the automatic restraints. Importantly, unlike automatic restraints, manual restraints are *not* constrained to carbon atoms but may be anything you can express using the MDAnalysis syntax - if you'd like to restrain a ligand entirely by its sulfur-bonded hydrogens to a zinc atom on the opposite side of the protein, that is entirely possible.

3.2.3 Applying the Force

Once the atom selection groups have been found, their constituent atoms need to be referenced and a restraint created within the underlying molecular dynamics engine. Whereas all previous steps are engine-agnostic, at this stage engine-dependent facilities need to be used. For this thesis, only facilities supporting openMM were created. Specifically, all restraints are mapped as openMMs' CustomCentroidBondForce, which applies a given energy expression between the centers of mass of two atom groups. These atom groups are simply arrays of atom indices, similar but different to those used by openMM. User-defined parameters relevant to the energy calculations alongside the intermediate state-dependent scaling factor are passed directly to the energy expression. A list of implemented energy expressions is provided in table 3.1.

Keyword	Expression	Description
harmonic	0.5*k*(distance(g1,g2)-r0)$\hat{}$2	A standard harmonic potential
flatbottom-oneside-sharp	step(distance(g1,g2)-r0) * (k/2)*(distance(g1,g2))$\hat{}$2	Flatbottom with no potential towards the origin, but a step outside the well
flatbottom-oneside	step(distance(g1,g2)-r0) * (k/2)*(distance(g1,g2)-r0)$\hat{}$2	Flatbottom with no potential towards the origin and no step out of the well
flatbottom-twoside	step(abs(distance(g1,g2)-r0)-w)*k*(distance(g1,g2)-r0)$\hat{}$2	A two-sided flatbottom potential, with steps outside the well

Table 3.1: Overview of the available energy expressions and their corresponding potential shapes. k refers to the spring constant, r is the current and r_0 the initial distance between atom groups g_1, and g_2

All of these operations happen within the `openmm_run.py` file responsible for running the openMM simulation. After creation, these forces are then injected into the openMM system where they were evaluated alongside the remaining constituent forces during runtime.

3.3 Simulations

All inputs were generated using CHARMM-GUI[22], and used the CHARMM36m forcefield[23]. For easier handling and allowing larger timesteps, CHARMM-GUI's facility for hydrogen mass repartitioning[24] was used unless noted. Equilibration was done using the openMM - Inputs provided by CHARMM-GUI[25, 26]. To modify the ligand, CHARMM-GUI's ligand designer[27] was used alongside the commercial program Maestro[28]. The simulations themselves were carried out using openMM 7.5[29]. Calculations were conducted via distributed computing utilizing consumer-grade Nvidia GPUs of the RTX 2080, RTX 1080, and RTX 1060 series.

3.3.1 Involved Molecules

2OJ9 with modified BMI as ligand

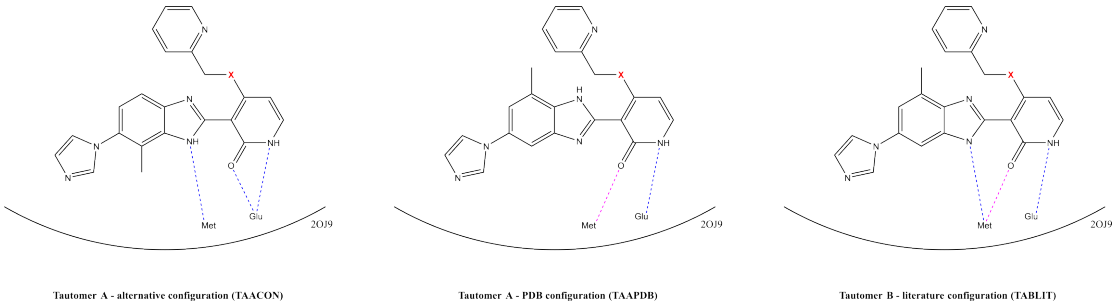

Figure 3.1: The varieties of BMI bound to 2OJ9. The red X is substituted with the appropriated functional group of the modification observed, with standard BMI possessing an NH group in its place. Dashed bonds indicate likely interactions with protein amino acids, with violet bonds indicating deviations to TAACON

2OJ9[1] is a complex of the IGF-1-R insulin growth factor with 3-[5-(1H-imidazol-1-yl)-7-methyl-1H-benzimidazol-2-yl]-4-[(pyridin-2-ylmethyl)amino]pyridin-2(1H)-one (BMI) as inhibitor, first discovered by Velaparthi et. al. in 2007 [30]. IGF-1-R has continued to gather medical attention, especially due to its likely role in tumor promulgation [31]. Velaparthi et. al. discuss the impact a variety of substitute operations in BMI have on inhibition. A number of these proposed substitutes, using their numeric identifiers were used for simulations: 24 - NH was used as the baseline comparison due to its high reported binding affinity, with 25 - sulfur and 26 - oxygen also simulated (refer to Fig. 3.3.1 for placement).

All 2OJ9 derivatives were prepared using the standard procedure outlined in section 3.3.

VIM-2 with zinc ligands

VIM-2 is a Carbapenem-Hydrolyzing Metallo-β-Lactamase, responsible for a subtype of bacterial drug resistance.[32]. ZN148, ZN223, and their derivatives are specific zinc ligands first presented by Samuelsen et al.[33]. These ligands are designed to inhibit Vim2 activity and suppress its drug-resistant activity, thus providing a treatment vector for strains containing it.

To prepare the structures for use in transformato, protonation states were determined by use of ProToss[34, 35] and manual observation, after which ligand and protein were solvated using CHARMM-GUI's solvation builder, including patching of the C-Terminus.

[1] https://www.rcsb.org/structure/2oj9

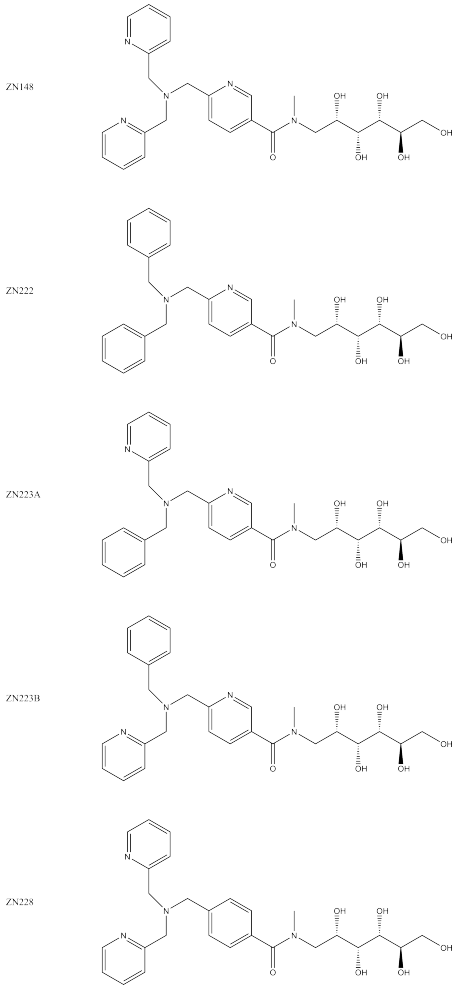

Figure 3.2: The various ZN* - ligands tested against VIM-2.

As certain binding modes require the presence of an OH^- group nearby, but CHARMM-GUI's solvation builder is unable to process the system with it, a roundabout way was chosen where a separate .crd file for the OH group was created and patched in after solvation using `macha`[36]. As this also complicated HMR, Hydrogen Mass Repartitioning was accomplished using `parmed`[21]. Afterwards, use in `Transformato` continued as normal.

Tests included a total of five different ZN* - ligands as denoted in Figure 3.3.1. Nomenclature follows the original used by Samuelsen [33].

3.3.2 Binding site dynamics simulations

Initial efforts were focused on workable implementations of restraints into an openMM system. To that effect, a number of simulations were conducted using 2OJ9. These did

not utilize `Transformato` but rather a variety of dedicated testing scripts. Trajectories were analyzed with regard to the relative distance allowed by various restraints and parameters, the effects of their applications on the complex' total potential energy and its root mean square deviation of atomic position (RMSD), a method to measure positional changes of non-hydrogen protein atoms against the original structure [37].

For 2OJ9 derivatives, relative distances were measured according to the likely interaction partners depending on the structure used (refer to Fig. 3.3.1. Limited simulations were also undertaken using VIM-2. Here, the distance between the nitrogen in the heterocycles to the next acidic hydrogen was measured.

3.3.3 `Transformato` RBFE calculations

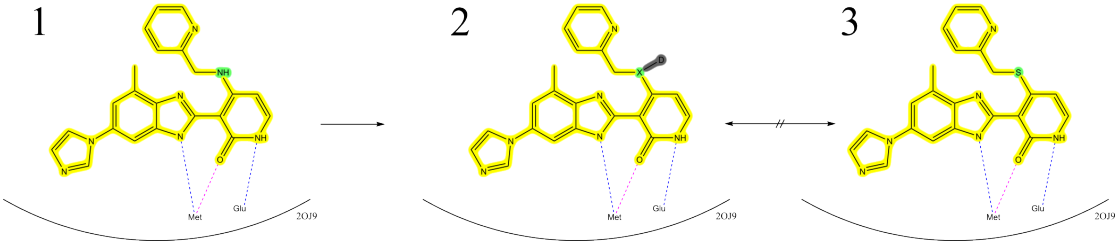

Figure 3.3: Schematic display of mutation occurring for 2OJ9 TABLIT 24 to the common core with TABLIT 25. Yellow: Common Core. Green: Objects that will be mutated. Grey: The dummy region resulting from the superfluous hydrogen. As apparent, the common core (2) resulting from mutation of (1) is not exactly equivalent to the unmated structure (3)

Here, the focus lay primarily on showing whether the introduction had a significant effect on calculated RBFE. No RSFE simulations were conducted. To facilitate this, 2OJ9 and VIM-2 systems were restrained using a variety of parameters and compared to equal-conditioned unrestrained simulations. Three replicates were used for each set of parameters. As the simulations were resource-intensive, simulations were only conducted one-sided, meaning that only one endstate was mutated into the common core, with the other endstate being chosen in such a way that it was structurally identical to the common core. However, due to differences arising from charge compensation in the terminal junction (see Fig. 3.3.3) this means that despite the identical structure this does not supplant a full two-sided simulation. As such, all results are qualitative rather than quantitative.

4 Results

4.1 Binding site dynamics simulations

A total of 181 simulations were conducted, of which only a small subset is presented here (full data available via the repository). To visualise the relative distances of the virtual bonds as defined in section 3.3, distance data was aggregated across time and intermediate states, and are displayed as arithmetic mean plus confidence interval. *Y - axes not equally scaled, all using a harmonic potential.*

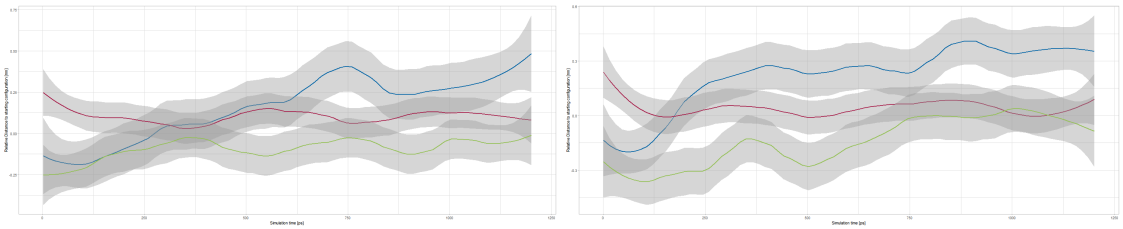

Figure 4.1: Relative distance plot for TAAPDB 24 to 25, k=0 (left) and k=400 (right), three extremities restraint.

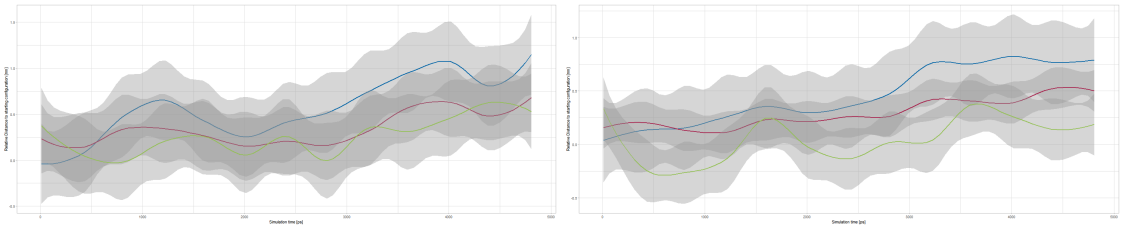

Figure 4.2: Relative distance plot for TABLIT 24 to 25, k=3 (left) and k=100 (right), three extremities restraint.

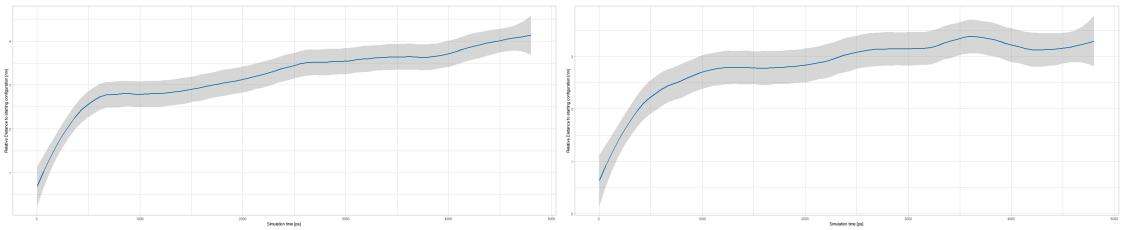

Figure 4.3: Relative distance plot for ZN222 to ZN148, k=3 (left) and k=100 (right), simple restraint.

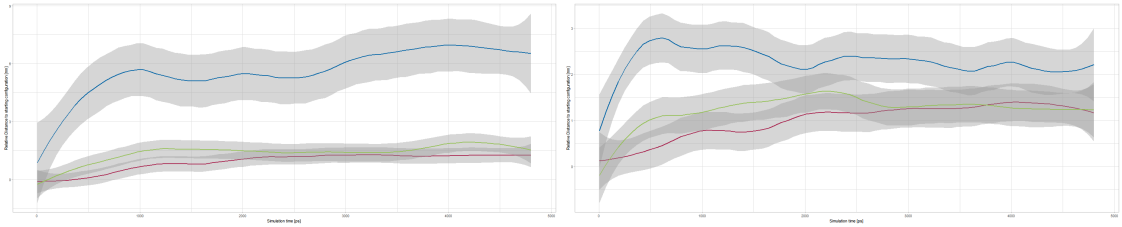

Figure 4.4: Relative distance plot for ZN222 to ZN148, k=0 (left) and k=100 (right), three extremities restraint.

4.2 `Transformato` RBFE calculations

RBFE calculations were conducted as outlined in section 3.3, with hydrogen mass repartitioning and a timestep of 0.004 ns, varying parameters such as force constant and restraint type. Due to time and computational constraints, not all variations for all systems were explored. Boxes display first to third quartile of data, outliers are shown as dots. A minimum of three replicates were used for each data point. Equal - colored boxes represent equivalent values of the force constant k. Y-axis scale is fixed across comparative results. Datasets may include both scaled and nonscaled results; an unpaired Student's T-Test [38] was applied to test for significant differences between the means of the samples. As it failed to find such differences at a confidence level of 95%, scaled and unscaled results were combined for the purposes of these plots, effectively giving $n = 6$ for parameter sets where scaled and nonscaled simulations were conducted.

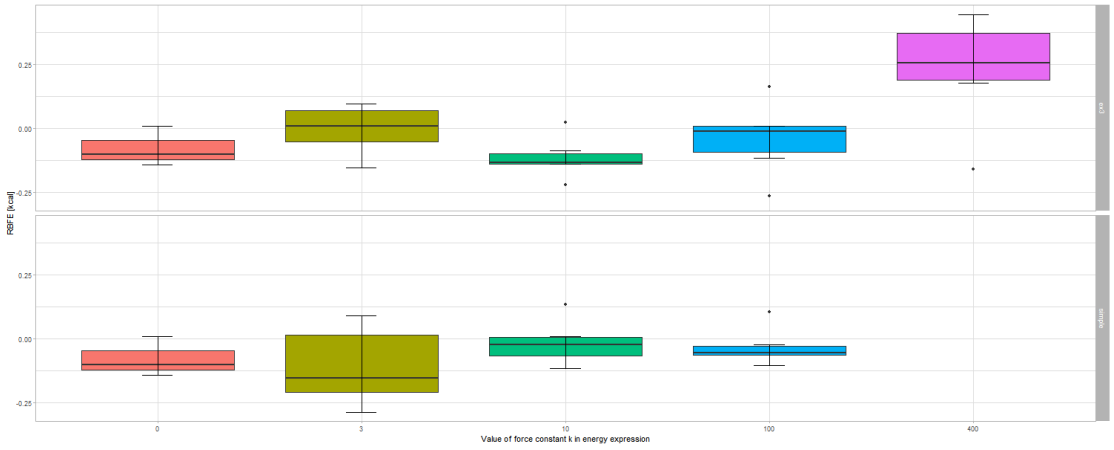

Figure 4.5: RBFE results for TAAPDB 24 to TAAPDB 25 at a runtime of 1.25 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

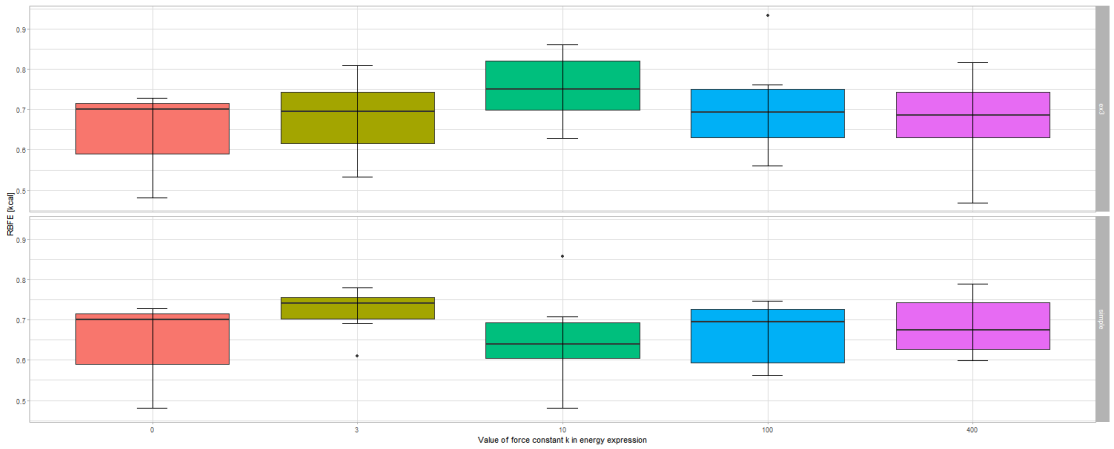

Figure 4.6: RBFE results for TABLIT 24 to TABLIT 25 at a runtime of 1.25 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

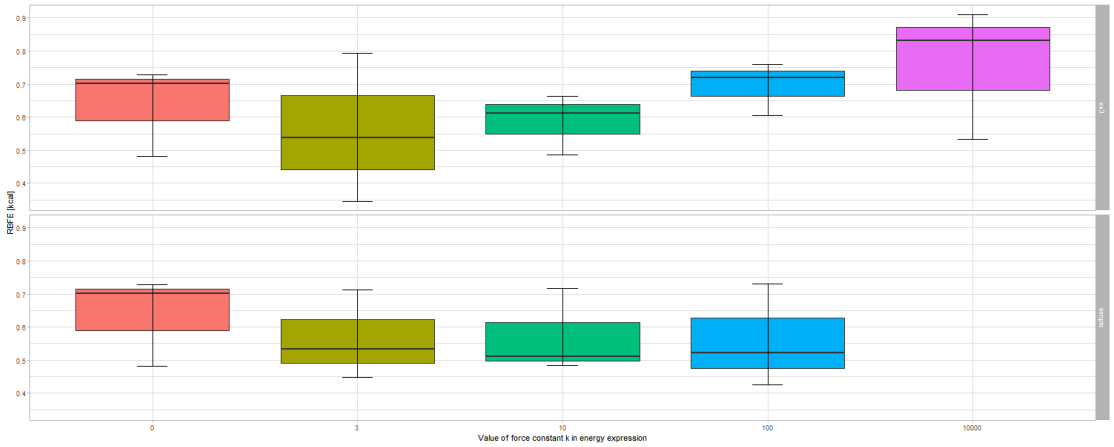

Figure 4.7: RBFE results for TABLIT 24 to TABLIT 25 at a runtime of 5 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

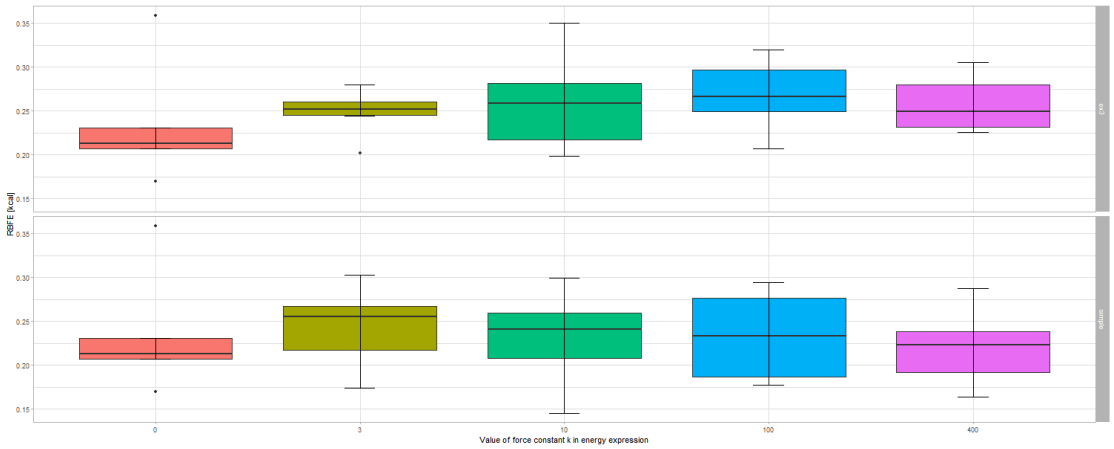

Figure 4.8: RBFE results for TABLIT 24 to TABLIT 26 at a runtime of 1.25 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

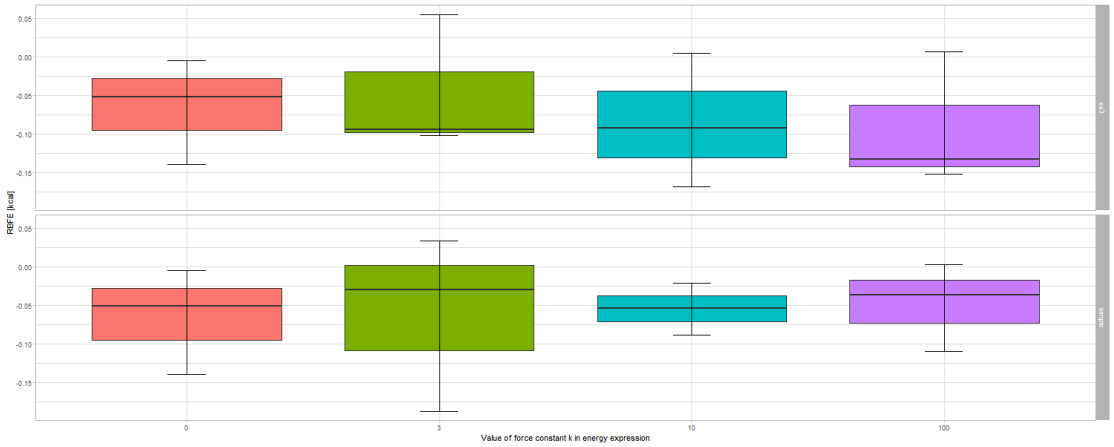

Figure 4.9: RBFE results for ZN148 to ZN222 at a runtime of 5 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

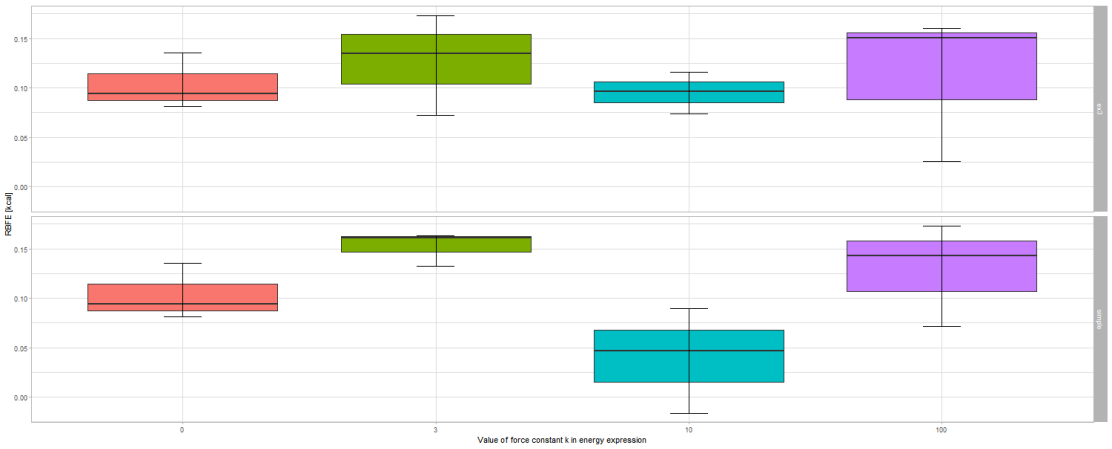

Figure 4.10: RBFE results for ZN148 to ZN223a at a runtime of 5 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

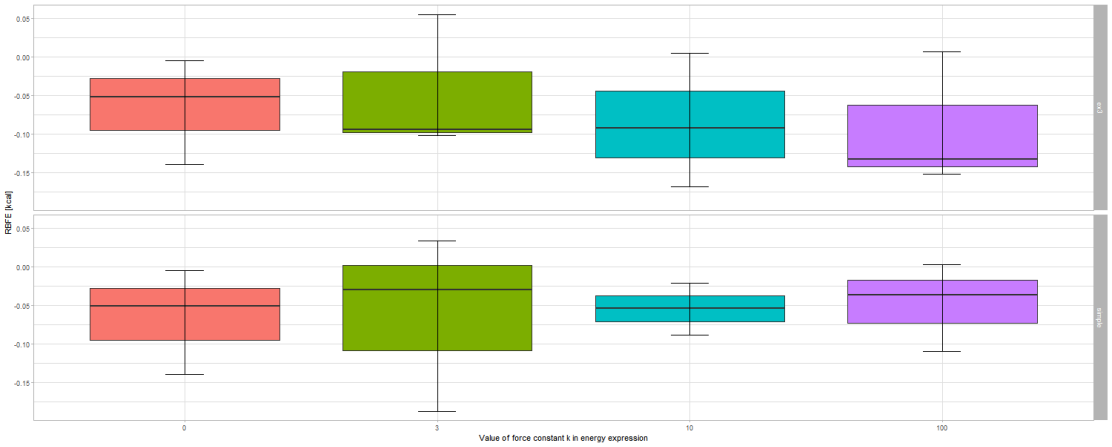

Figure 4.11: RBFE results for ZN148 to ZN223b at a runtime of 5 ns for both simple and three-extremities restraints. $k = 0$ represents the unrestrained comparison system.

5 Discussion

5.1 Comparison of restrained to unrestrained results

Binding site dynamics (Section 4.1) Here, the first obvious result is the fact that restraints themselves work; significant deviations can be observed between the unrestrained and heavily restrained systems. Perhaps most obvious in Fig. 4.4, a VIM-2 derived system where the unrestrained system moved away an average of 9 angstrom during the simulation, whereas the restrained system only moved away 3 angstrom. Similar, though less drastic results can be observed in the other data presented. As such, it can be concluded that at the basic technical level, the restraints function as designed. All of the data presented here is achieved through the use of harmonic potentials; while flat-bottomed potentials were implemented 3.2, significant modifications between the originally tested and now-implemented version along with time constraints preclude their inlcusion in the presented dataset. However, the results of the limited number of simulations conducted are aligned with expectations.

When discussing individual systems, it can be observed that VIM-2-based systems seem most likely to abscond their binding site, covering significant distances even within the simulations. A similar, but much less distinct behaviour can be observed for TAAPDB, with TABLIT being the most stable system in comparison.

Transformato RBFE calculations (Section 4.2) When discussing the results of the free energy calculations, TAAPDB is somewhat of an outlier. The most heavily restrained system with $k = 400$ is the only system to show significant deviation to the other restraint systems along with the unrestrained systems, being the only group in the dataset to fail Student's t-test when compared to the unrestrained system[1]. This would indicate that at a certain point, restraints do impact RBFE calculations. However, it should be noted that this behaviour is present in both scaled and nonscaled simulations, that a very high k of 400 is required, and that it is the only set to show such deviation. Notably, TABLIT does not significantly differ from the unrestrained set even at $k = 10000$. It should also be noted that it is only an outlier in the context of the very limited spread of the data presented here, with a mean derivation of 0.19 kcal to the unrestrained set. Given typical uncertainties of 1-2 kcal with RBFE calculations, this does not seem a significant factor.

Overall however, no impact of restraints on the calculated free energies can be observed. Within the limitations discussed in sections 2.2.1 and 3.3, this would seem to validate the theory that restraints do not have significant impact on RBFE calculations. This, on the other hand also means that there seem to be no significant *benefits* to using restraints; however this may be a factor of the very limited simulation times (1.25 or 5 ns). Restraints may also find utility in non-standard calculations, such as those approximating a pull-mechanism.

[1] After removal of the outlier aligned with the unrestrained set

5.2 Limitations of the current approach and future possibilities

While the results thus far look promising, some inherent technical problems do present themselves that may limit applicability in some cases. Firstly, the strict delineation between the common core (to which the restraints are applied) and everything else (where they are not) may present a problem in cases where a significant portion of the ligand is mutated. This problem also arises when connecting parts are mutated if the outlying parts are considered separate by `Transformato`, though in this case it may be circumvented by manually modifying the common core. In a similar vein, there currently exist no mechanism to accommodate changes in the protein.

Automation improvements With `Transformato`'s general push towards accessibility and quick setup even for large batches, the current process to define even automatic restraints is unsatisfactory. In particular, the productive use of extremity restraints requires prior manual inspection of the ligand in question, and parameter setup at this stage is mostly guesswork. While the default values should be sensible for most use cases, further automation with consideration of molecule and binding site geometry along with charge interactions should make it possible to generate good data without human intervention in the loop, significantly increasing its usability for lead generation.

Quality-of-life improvements Currently, the `restraints.py` API does not expose much of the underlying framework, complicating efforts to expand or modify the provided functions. In particular, not all parameters used for the generation of restraints are readily accessible, including the algebraic expression used to calculate restraint forces. Modifying these parameters currently requires modifications of the `restraints.py` itself, necessitating reinstallation. This would complicate multi-platform deployments especially on clusters, troubleshooting etc. Ideally, an API would be provided to expose these parameters directly through the `submit.ipynb`. This, however, would require extensive expansion of the I/O capabilities between the submit and simulation stages, as the current YAML framework seems ill-suited for these kinds of modifications.

Package-independent cluster computation As of now, `restraints.py` is the only `Transformato` module that requires it be installed on the cluster serving the simulation to function. This also includes some of its dependencies, notably MDAnalysis. This complicates the workflow compared to standard `Transformato` usage, and may preclude deployment in non-permissive environments, or at the very least introduces an additional layer of complexity with involvement of IT staff. Given the current limitations applied to the restraint, in particular the fact it is only applied to the immutable common core, it should be possible to frontload the structure analysis underpinning the creation of the restraint, having it be handled at the `submit.ipynb` stage instead of the `openmm_run.py` stage. Assuming a satisfactory mechanism can be implemented to keep the openMM-generated atom idx constant, there would be no need for MDAnalysis or the restraints module being loaded during simulation.

CHARMM integration Currently, restraints are an experimental feature only available when using openMM. However, as otherwise the complete feature set of `Transformato` is available under CHARMM, it could be considered a waste to not have restraints implemented as well. This would be non-trivial, as CHARMM does not have the same type of centroid prototype force openMM posesses which is exploited for these restraints. Given language limitations, it would likely also require creation of dynamic simulation scripts during the submit stage instead of the static script used for openMM, thus creating a significant opportunity for critical bugs to arise.

5.3 Conclusion

A method to introduce restraints into `Transformato` was successfully implemented and performed adequately in testing. In the context of unstable systems, restraints accomplish their goal of retaining ligand-protein closeness without significant impact on calculated RBFE values. Conversely, no significant benefit from using restraints has been observed. As testing was done dissimilar to production conditions and within very limited timeframes, further research is required to draw definite conclusions.

6 Annex

6.1 Definitions

As far as possible, equations in this thesis are notated and styled as in the Alchemistry Wiki maintained by Shirts and Chodera [39]. The symbols and notations used resolve as follows:

Q	the partition function of the ensemble
k_B	Boltzmann's constant
T	Temperature
P	Pressure
V	(Box) Volume
U	Sum of internal energy in a MD simulation, both potential and kinetic energies
N	Number of particles in the system
β	Substitute for $(k_B T)^{-1}$ by convention
Γ	Phase Space Volume
λ	the *alchemical variable* or *coupling parameter*. Used to describe the progress of a transformation along the alchemical path
K	The *state variable*, describing all conditions and parameters of the thermodynamic system
k	A specific state found in K
u	The *reduced potential* $u = \beta(U + PV - \sum \mu N_i)$. For NPT ensembles, $\sum \mu N_i = 0$

List of Figures

List of Tables

References

(1) Cournia, Z.; Allen, B.; Sherman, W. *Journal of Chemical Information and Modeling* **2017**, *57*, 2911–2937.

(2) Karwounopoulos, J.; Wieder, M.; Boresch, S. Relative binding free energy calculations with Transformato: a molecular dynamics engine-independent tool, 2022.

(3) Braunsfeld, B. **2021**, DOI: 10.25365/thesis.66300.

(4) Wieder, M.; Fleck, M.; Braunsfeld, B.; Boresch, S. *Journal of Computational Chemistry* **2022**, *43*, 1151–1160.

(5) Fleck, M.; Wieder, M.; Boresch, S. *Journal of Chemical Theory and Computation* **2021**, *17*, 4403–4419.

(6) Shirts, M. R.; Chodera, J. D. *The Journal of Chemical Physics* **2008**, *129*, 124105.

(7) Bennett, C. H. *Journal of Computational Physics* **1976**, *22*, 245–268.

(8) Tzeliou, C. E.; Mermigki, M. A.; Tzeli, D. *Molecules* **2022**, *27*, 2660.

(9) Vanommeslaeghe, K.; Hatcher, E.; Acharya, C.; Kundu, S.; Zhong, S.; Shim, J.; Darian, E.; Guvench, O.; Lopes, P.; Vorobyov, I.; MacKerell, A. D. *Journal of computational chemistry* **2010**, *31*, 671–690.

(10) Hollingsworth, S. A.; Dror, R. O. *Neuron* **2018**, *99*, 1129–1143.

(11) Kumar, S.; Nussinov, R. *ChemBioChem* **2002**, *3*, 604–617.

(12) Fujitani, H.; Tanida, Y.; Ito, M.; Jayachandran, G.; Snow, C. D.; Shirts, M. R.; Sorin, E. J.; Pande, V. S. *The Journal of Chemical Physics* **2005**, *123*, 084108.

(13) Straatsma, T. P.; Berendsen, H. J. C.; Postma, J. P. M. *The Journal of Chemical Physics* **1986**, *85*, 6720–6727.

(14) Boresch, S.; Bruckner, S. *Journal of Computational Chemistry* **2011**, *32*, DOI: 10.1002/jcc.21713.

(15) Tan, Z. *Journal of the American Statistical Association* **2004**, *99*, 1027–1036.

(16) Boresch, S.; Bruckner, S. *Journal of Computational Chemistry* **2011**, *32*, 2449–2458.

(17) Beutler, T. C.; Mark, A. E.; van Schaik, R. C.; Gerber, P. R.; van Gunsteren, W. F. *Chemical Physics Letters* **1994**, *222*, 529–539.

(18) Li, Y.; Nam, K. *Journal of Chemical Theory and Computation* **2020**, *16*, 4776.

(19) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. *Journal of Computational Chemistry* **2011**, *32*, 2319–2327.

(20) Gowers, R. J.; Linke, M.; Barnoud, J.; Reddy, T. J. E.; Melo, M. N.; Seyler, S. L.; Domański, J.; Dotson, D. L.; Buchoux, S.; Kenney, I. M.; Oliver Beckstein In *Proceedings of the 15th Python in Science Conference*, ed. by Benthall, S.; Scott Rostrup, 2016, pp 98–105.

(21) Shirts, M. R.; Klein, C.; Swails, J. M.; Yin, J.; Gilson, M. K.; Mobley, D. L.; Case, D. A.; Zhong, E. D. *bioRxiv : the preprint server for biology* **2016**, 077248.

(22) Jo, S.; Kim, T.; Iyer, V. G.; Im, W. *Journal of Computational Chemistry* **2008**, *29*, 1859–1865.

(23) Huang, J.; Rauscher, S.; Nawrocki, G.; Ran, T.; Feig, M.; de Groot, B. L.; Grubmüller, H.; MacKerell, A. D. *Nature Methods* **2017**, *14*, 71–73.

(24) Gao, Y.; Lee, J.; Smith, I. P. S.; Lee, H.; Kim, S.; Qi, Y.; Klauda, J. B.; Widmalm, G.; Khalid, S.; Im, W. *Journal of Chemical Information and Modeling* **2021**, *61*, 831–839.

(25) Brooks, B. R. et al. *Journal of Computational Chemistry* **2009**, *30*, 1545–1614.

(26) Lee, J. et al. *Journal of Chemical Theory and Computation* **2016**, *12*, 405–413.

(27) Guterres, H.; Park, S.-J.; Cao, Y.; Im, W. *Journal of Chemical Information and Modeling* **2021**, *61*, 5336–5342.

(28) Schrödinger release 2022-2, 2021.

(29) Eastman, P.; Swails, J.; Chodera, J. D.; McGibbon, R. T.; Zhao, Y.; Beauchamp, K. A.; Wang, L.-P.; Simmonett, A. C.; Harrigan, M. P.; Stern, C. D.; Wiewiora, R. P.; Brooks, B. R.; Pande, V. S. *Plos Computational Biology* **2017**, *13*, e1005659.

(30) Velaparthi, U. et al. *Bioorganic & Medicinal Chemistry Letters* **2007**, *17*, 2317–2321.

(31) Chiu, Y.-J.; Hour, M.-J.; Jin, Y.-A.; Lu, C.-C.; Tsai, F.-J.; Chen, T.-L.; Ma, H.; Juan, Y.-N.; Yang, J.-S. *International Journal of Oncology* **2018**, *52*, 1465–1478.

(32) Poirel, L.; Naas, T.; Nicolas, D.; Collet, L.; Bellais, S.; Cavallo, J.-D.; Nordmann, P. *Antimicrobial Agents and Chemotherapy* **2000**, *44*, 891.

(33) Samuelsen, Ø. et al. *Antimicrobial Agents and Chemotherapy* **2020**, *64*, DOI: 10.1128/AAC.02415-19.

(34) Lippert, T.; Rarey, M. *Journal of Cheminformatics* **2009**, *1*, 1–12.

(35) Bietz, S.; Urbaczek, S.; Schulz, B.; Rarey, M. *Journal of Cheminformatics* **2014**, *6*, 1–12.

(36) Åsmund Kaupang macha: MAnual CHArmm, 2022.

(37) Yusuf, D.; Davis, A. M.; Kleywegt, G. J.; Schmitt, S. *Journal of Chemical Information and Modeling* **2008**, *48*, 1411–1422.

(38) Lüroth, J. **1876**, DOI: 10.1002/asna.18760871402.

(39) Shirts, M. R.; Chodera, J. D.; Naden, L.; Mobley, D. L. AlchemistryWiki.