Macromolecular Docking - project seminar

Peter Mühlbacher - a1253030

June 10, 2015

Contents

Abstract

Weekly reports on advances, encountered difficulties and implementations for the seminar on artificial intelligence will be gathered in this document.

Goals

In this seminar I aim to examine the problem of (flexible) protein-protein docking.

1 Week 1

1.1 The Original Goal

Originally I wanted to approach the problem of protein folding by investigating the asymptotic ($t \to \infty$) behaviour of the probability density function of a given protein. As the changes over time of a given protein[1] can be modelled by a Langevin equation $\dot{x} = -\nabla U(x) + \beta \dot{w}$, the corresponding probability density function $p(t,x)$ is described by the forward Fokker-Planck equation $\frac{\partial p}{\partial t} = \tilde{\beta}\Delta p + \operatorname{div}(p\nabla U)$ whose asymptotic behaviour has been investigated in Nadler et al. (2008).

However, there are (at least) two reasons why this does not work:

- In order to get a meaningful low dimensional representation of the system, a considerable margin between two adjacent eigenvalues $\lambda_{k+1} \gg \lambda_k$ of the Fokker-Planck operator is needed and it can be shown that this roughly corresponds to having a potential function U with k local

[1] understood as a single point in high dimensional space

minima[2]. Taking these properties into consideration it seems futile to employ this method of dimensionality reduction as the potential function in the area of protein folding has thousands of local minima.

- Even if solutions could be found explicitly in a reasonable amount of time the problem of finding and exact potential function U still remains unsolved[3]. As we are working with approximations it does not seem to make any sense trying to study its asymptotic long-term behaviour.

As a result of above considerations my topic of this seminar was restricted to docking. In the special case of rigid docking both molecules are assumed to be in a metastable state which drastically cuts the the complexity of the state space.

2 Week 2

2.1 Benchmarks

Visit `http://zlab.umassmed.edu/benchmark/` for a benchmark on various docking problems that are ordered by difficulty. There are test cases for rigid docking as well as flexible docking, again ordered by amount of typically changing parameters like torsion angles.

2.2 PDB

The *Protein Data Bank Format* is a plain text data file, storing (amongst other information) the coordinates of every single atom in a protein.

A typical *PDB*-file's row might look like this:

$$\text{ATOM}\underbrace{\ 2\ }_{\text{atom number}}\underbrace{\ \ldots\ }_{\text{various identifiers (1)}}\underbrace{57.827}_{\text{x-coordinate (Å)}}\underbrace{\ \ldots\ }_{\text{y- and z-coordinates}}\underbrace{1.00}_{\text{occupancy (2)}}\underbrace{66.32}_{\text{temperature factor (3)}}\underbrace{\ \text{N}\ }_{\text{element symbol}}$$

[2]c.f. (Nadler et al., 2008, pp.9), p.9
[3]c.f. Neumaier (2006), p.14ff

1. These identifiers are not of importance for this project.
2. In case an atom has been found in more than one locations all the different coordinates and according occupancies are listede.g. if it has been found in two different locations, both locations are represented as distinct rows with an occupancy factor of 0.5 each.
3. The temperature factor is an indication of how rigidly the atom is held in place in the given structure. Lower values imply that it is a more "stable" position.

For a list of PDB files for various proteins visit `http://www.rcsb.org/`.

2.3 Modelling the Potential

Given two metastable molecules $M_i = (\mathbf{x_i}, \alpha_i, \beta_i, \gamma_i, \{\mathbf{a_{ij}}\}_{j=1}^{n_i})$, $i \in \{1,2\}$. In the following $\mathbf{x_i} \in \mathbb{R}^3$ will be called the molecule's (or the system's) point of reference, $\alpha_i, \beta_i, \gamma_i \in [0, 2\pi)$ the system's rotation (where α_i corresponds to the rotation around the x-axis relative to its point of reference; analogously for β_i, γ_i) and $\mathbf{a_{ij}} \in \mathbb{R}^3$ the system's atoms with (absolute) coordinates $\mathbf{x_i}+\mathbf{a_{ij}}$, where n_i is the number of atoms.

The Lennard-Jones potential of a system consisting of two points $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ with distance $r^2 = ||a - b||$ is defined as

$$V_{LJ}(r) = \varepsilon \left(\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^{6} \right), \tag{1}$$

where ε is the size of the potential well and σ the distance at which the potential reaches its minimum. Both are usually given by experiments.

In order to investigate the energy landscape for a docking problem it is natural to consider the Lennard-Jones potential for a system consisting of two proteins M_1, M_2 given by

$$\tilde{V}_{LJ}(M_1, M_2) = \sum_{i=1}^{n_1+n_2} \sum_{j>i}^{n_1+n_2} V_{LJ}(r_{ij}) \tag{2}$$

if we set r_{ij} to be the distance between the ith and jth atom[4].

[4] $i, j \in \{1, \ldots, n_1 + n_2\}$

If we want to explicitly calculate its gradient with respect to the free parameters $\mathbf{x_1}, \alpha_1, \beta_1, \gamma_1$ it is instructive to write down the entire formula as a function of those variables. First of all, to faciliate computation, we split up the summation into three parts $\tilde{V}_{LJ}(M_1, M_2) = \tilde{V}^1_{LJ}(M_1) + \tilde{V}^2_{LJ}(M_2) + \tilde{V}^3_{LJ}(M_1, M_2)$, where $\tilde{V}^1_{LJ}$ refers to all combinations of atoms of M_1, $\tilde{V}^2_{LJ}$ to those of M_2 and $\tilde{V}^3_{LJ}$ to the summation over all pairs $(a_{1i}, a_{2j})_{i\in\{1,\dots,n_1\}, j\in\{1,\dots,n_2\}}$. It is apparent that $\tilde{V}^1_{LJ}$ and $\tilde{V}^2_{LJ}$ do not change as a function of the arguments chosen above (however, they would in case we also introduced variable torsion angles and such). To calculate the gradient it thus suffices to inspect $\tilde{V}^3_{LJ}$ as a function of the free parameters $x, y, z, \alpha, \beta, \gamma$ describing the position of M_1:

$$\tilde{V}^3_{LJ}(\mathbf{x}_1, \alpha_1, \beta_1, \gamma_1) = \varepsilon \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(\frac{\sigma}{r_{ij}}\right)^{12} - 2\left(\frac{\sigma}{r_{ij}}\right)^{6}, \tag{3}$$

where $r^2_{ij} = \|\tilde{a}_{1i} - a_{2j}\|$ and $\tilde{a}_{1i}$ are the first system's ith particle's absolute coordinates given by

$$R_x(\alpha_1)R_y(\beta_1)R_z(\gamma_1)\mathbf{a}_{1i} + \mathbf{x}_1,$$

where

$$R_x(\alpha_1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_1 & -\sin\alpha_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{pmatrix},$$

$$R_y(\beta_1) = \begin{pmatrix} \cos\beta_1 & 0 & -\sin\beta_1 \\ 0 & 1 & 0 \\ \sin\beta_1 & 0 & \cos\beta_1 \end{pmatrix}$$

and

$$R_z(\gamma_1) = \begin{pmatrix} \cos\gamma_1 & -\sin\gamma_1 & 0 \\ \sin\gamma_1 & \cos\gamma_1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Now one can explicitly compute the partial derivatives $\frac{\partial \tilde{V}_{LJ}}{\partial p} = \frac{\partial \tilde{V}^3_{LJ}}{\partial p}$, for p being one of the parameters.

As of now we can compute the potential $\tilde{V}_{LJ}(\mathbf{X})$ for a given point $\mathbf{X} = (\mathbf{x}_1, \alpha_1, \beta_1, \gamma_1)$ in the state space and its gradient $\nabla\tilde{V}_{LJ}(\mathbf{X})$ and that is all we need for algorithms like steepest descent (which would still require rescaling of the parameters as to avoid extensive zig-zagging) or diffusion maps.

Furthermore, note that the computation of the potential and the gradient can be combined into one double loop, making[5] it $O\left((n_1+n_2)^2\right)$.

3 Week 3

3.1 Ordering Dihedral Angles by Variance

To be able to approximate the real docking process as closely as possible while only letting some chosen dihedral angles be variable, one first needs to order them accordingly. One such order might be given by regarding each such angle Θ_i as a discrete random variable and ordering them by their variances.

More precisely, a set of a single molecule's uniformly distributed[6] conformations $\{M_k\}_{k=0}^n$ induces a discrete probability space $(\{M_k\}_{k=0}^n, \mathscr{A}, P)$ where $P(M_k) := \frac{V_{LJ}(M_k)}{Z}$, where $Z := \sum_{k=1}^n P(M_k)$ is a normalising factor and for all A in $\mathscr{A} : P(A) = \sum_{M \in A} P(M)$.

Another approach might be to compute sample trajectories and use them (regardless of their distribution) instead of uniformly distributed samples.

For a fixed Θ_i we obtain the expected value $E[\Theta_i] = \sum_{k=1}^n P(M_k)\Theta_{ik}$, were Θ_{ik} is the k-th conformation's i-th dihedral angle. However, this approach does not consider the fact that these values are circularly distributed; thus it may be better to assume that the angles' values are normally distributed and to estimate the parameters of the von Mises distribution.

The variance $\text{Var}[\Theta_i]$ is given by $E[(\Theta_i - E[\Theta_i])^2]$.

Other ideas included:

- Looking if such a list already existed for some molecules, but I couldn't find any.

- Taking different conformations from `rcsb.com` and using PCA on the phase space consisting of the dihedral angles, but given the number of dimensions compared to the amount of samples available[7] this is not a promising approach either.

[5] although the computation of the gradient is likely to result in a big constant factor of operations

[6] This requirement is necessary because we do not want to count "very similar" conformations twice.

[7] For albumin there are 104 entries as of 14.04.2015.

- Take Ramachandran plots, interpret them as conditional probabilities and combine all of them. The data for this approach should be available, but combining all those single results may be hard.

3.2 Rotation About an Arbitrary Axis in 3D

To simulate flexible docking it is essential to be able to determine the single atoms' Cartesian coordinates, given one point of reference, the distance between two adjacent atoms, the angles of three consecutive atoms and the dihedral angles.

Given four atoms $\{a_i\}_{i=0}^{3}$, s.t. a_j and a_{j+1} are covalently bonded their dihedral angle is the smallest angle between the two planes π_0 and π_1 which are uniquely determined by $\{a_i\}_{i=0}^{2}$ and $\{a_i\}_{i=1}^{3}$, respectively.

Given $\{a_i\}_{i=0}^{2}$ one sets a_3 in $_0$, s.t. it has the prescribed distance as well as the angle between a_1, a_2 and a_3. Now we want to rotate a_3 about the line $a_2 + \overrightarrow{a_1 - a_2}$.

Clearly this is not a linear mapping unless $a_2 = 0$, but by introducing a forth dimension we can turn three-dimensional translations into linear mappings.

See `http://inside.mines.edu/fs_home/gmurray/ArbitraryAxisRotation/` for a derivation of the according formula.

4 Week 4

4.1 Ordering Dihedral Angles by Variance (continued)

A solution that is better suited for this problem only considers the current configuration. Thus it suffices to consider the given configuration[8] $C = (\Theta_i)_{i=1}^{n}$, calculate its potential energy $V_{LJ}(C)$ and then, for every dihedral angle Θ_i, evaluate the difference $\Delta_i^{\varepsilon} := V_{LJ}(C) - V_{LJ}(C_i^{\varepsilon})$, where C_i^{ε} is the configuration $(\Theta_1, \ldots, \Theta_{i-1}, \Theta_i + \varepsilon, \Theta_{i+1}, \ldots, \Theta_n)$.

To get an ordering of the dihedral angles' variance we set $k <_{\varepsilon} j$ if $\Delta_k^{\varepsilon} + \Delta_k^{-\varepsilon} < \Delta_j^{\varepsilon} + \Delta_j^{-\varepsilon}$ which clearly defines an absolute order for every ε.

[8]All other parameters are assumed to be fixed.

5 Weeks 5,6,7

5.1 The Node Class

To store the data I introduced a *Node* class with the following fields:

1. `atomname`(String): atom name, e.g. `_C__`, `_CG1`, where underscores are used in lieu of whitespaces to remove ambiguities; so far this field doesn't serve any particular purpose except making debugging easier

2. `coord`(double[]): coordinates of the node given as a list (e.g. $[0, 0, 0]$; usually that's what we're calculating from the other arguments, also referred to as "internal coordinates")

3. `dist`(double): distance to the parent node (bond length)

4. `phi`(double): bond angle between `this`, `this.parent`[9] and `this.parent.parent`

5. `theta`(double): dihedral angle between `this`, `this.parent`, `this.parent.parent` `and this.parent.parent.parent`

6. `children`(Node[]): list of other node elements

In a real world applications does not seem to be the way to go since languages like Python have a recursion limit which is easily reached. In Python one can kind of avoid this problem by manually increasing this limit by `import sys` and `sys.setrecursionlimit(n)`, where n should be adjusted to the molecule one is currently dealing with. It seems to be common practice[10] to rewrite the code to use iterative instead of recursive methods in those cases, but as I couldn't find a source stating the recursive approach is slower I kept the current, recursive methods which are easier to implement.

5.2 Absolute to Internal Coordinates

The task of extracting a tree-like structure as defined in the `Node` class from the PDB file was much more tedious than I originally thought. The main

[9] `this.parent` is only pseudo syntax, a parent field is not implemented because its recursive nature would slow down the program considerably.

[10] e.g. c.f. `http://stackoverflow.com/questions/8177073/python-maximum-recursion-depth-exceeded`

problem was that no `parent` fields were in use (see above why) and thus a separate structure containing information about the current item's predecessors had to be maintained. In a linear setting this would not have caused any problems, but if there are branching points it gets harder and harder to keep track of what's happening. I am really lucky to have chosen Python as my programming language as variables are "passed by assignment"[11] which makes everything much more convenient as if variables were passed by value. Note that the "link" created by the "passing by pointer" method for mutable objects like lists can be broken by an assignment which was used a lot in the actual implementation.

Another issue was that there are actually a lot more implicit assumptions about protein structure in the PDB files than I had previously imagined and from author to author further undocumented "conventions" are introduced.[12]

Obvious conventions are the structure of the backbone which is always assumed to be obvious. However, when it comes to hydrogen atoms it gets complicated. In some PDB files they are ignored completely, in others *some* are given[13] and then there are some where every H atom's position got an entry. Due to the inconsistency with H atoms I choose to ignore them completely.

Concerning the "atomname" field (e.g. `_C__`, `_CG1`, `_NE2`) the official documentation[14] was no help; what really helped was this unofficial "guide": `http://haldane.bu.edu/needle-doc/new/atom-format.html` and taking a look at a real PDB file[15] while comparing each residue to its corresponding Wikipedia entry.

Another convention is that if branches were already in use and an atom doesn't have a branch, nor a distance identifier then it is assumed to be a "child" of the previous atom.

Also, it is clear that ring-like structures like in Tryptophan are not compatible with the tree-like structures. The simple solution is to just ignore

[11] `https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference`

[12] c.f. `http://chemistry.stackexchange.com/questions/30807/about-undocumented-conventions-in-pdb-files` for an example with H.

[13] However, it is not clear what they are connected to, as there are no additional `CONECT` records and the usual convention when it comes to numbering branches is ignored.

[14] `http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html#ATOM`.

[15] I chose `http://www.rcsb.org/pdb/files/1E7I.pdb`.

one bond and not let the ring's dihedral angles vary.

The actual implementation is not particularly interesting from a mathematical point of view as it is basically a long list of `if...else` branches. In case you are interested, take a look at `http://htmlpreview.github.io/?https://github.com/petermuehlbacher/reports/blob/master/docking/implementation/proteindocking.html`.

5.3 The NeRF Algorithm

As proposed in `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.8235&rep=rep1&type=pdf` we calculate an atom's absolute coordinates. This function, depending on the atom's internal and its three ancestors' absolute coordinates, will henceforth be referred to as `nerf()`.

5.4 Calculating the Children's Absolute Coordinates

First of all we define a recursive function

```
Data: parent(Node), A(double[]), B(double[])
Result: A recursive method to traverse a tree-like structure and
        apply the NeRF algorithm to every child. Also adds the
        calculated coordinates to the molecule(double[][]) list.
C ← parent.coord;
for child in parent.children do
    child.coord ←
    nerf(A,B,C,child.dist,child.theta,child.phi);
    molecule.append(child.coord);
    buildChildren(child,B,C);
end
```

Algorithm 1: buildChildren()

Note that in the actual implementation the NeRF algorithm is called with `parent.dist` as additional parameter to get rid of one additional computation.

Also, the first three levels' atoms are assumed to be static due to the internal coordinates' nature. In practice these first three levels would be the first residue's N, CA, CB (in case there is one) and C. Note that, in order to bypass this problem, one could have defined "imaginary" auxiliary atoms at some coordinates, fixed by convention, in order to retrieve internal coordinates like a dihedral angle for the first residue's CA, etc.

6 Weeks 8,9

6.1 New Data Structure

As the progress of the implementation went on it got more and more apparent that the recursive, tree-like structure was not the way to go. Direct access to certain nodes was only possible by traversing all its ancestors or creating and maintaining another, linearised data structure (i.e. a list for each relevant property).

As a result, not only the problem with Python and its limitations when it comes to depth of recursions disappear, but calculations speed up considerably and there is no need to maintain a second, linearised structure.

The new data structure looks like this:

`mol = [atomnames,X,Y,Z,phis,thetas,dists,children]`, where `atomnames` is a list of atomnames as they are given in the PDB file, i.e. `atomnames[0]=" N "`, `atomnames[1]=" CA "`, etc., `X[i]`,`Y[i]`,`Z[i]` are i-th atom's coordinates, `phis[i]` the bond angles between the $(i-2)$-th, the $(i-1)$-th and i-th atom, `thetas[i]` the dihedral angle between the dihedral angle between the $(i-3)$-th, the $(i-2)$-th, the $(i-1)$-th and i-th atom, `dists[i]` the distance between the $(i-1)$-th and i-th atom and `children[i]` a list of indices of atoms that are "children" of the i-th atom. In this paragraph the $(i-1)$-th atom refers to the i-th atom's parent, the $(i-2)$-th atom to the parent of the i-th atom's parent, etc.

6.2 New Lennard-Jones Potential

Instead of using $\mathrm{LJ}_{\varepsilon,\sigma}(r) = \varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - 2\left(\frac{\sigma}{r}\right)^{6}\right)$ I decided to go with

$$\mathrm{LJ}_{A,B}(r) = \frac{A}{r^{12}} - \frac{B}{r^{6}} + C$$

which is computationally more efficient, thus easier to optimize and (bar the constant C which will be dealt with later) is connected to the previously used parameters ε, σ as follows:

ε being the depth of the potential well, r_m the distance at which the potential reaches its minimum (i.e. $LJ(r_m) = -\varepsilon$ and σ the distance at which the inter-particle potential is zero (i.e. $LJ(\sigma) = 0$ for the first time), then

$$A = 4\varepsilon\sigma^{12},$$
$$B = 4\varepsilon\sigma^{6},$$

or alternatively:

$$\varepsilon = \frac{B^2}{4A},$$
$$\sigma = \sqrt[6]{\frac{A}{B}},$$
$$r_m = \sqrt[6]{2}\sigma.$$

As such A may be regarded as the strength of the Pauli-repulsion and B as the attractive long-range term.

The constant C is chosen such that, for varying A, B (which happens when trying to find the optimal set of parameters (c.f. next subsection) $\mathrm{LJ}_{A,B}(r) > 0 \,\forall r > 0, (A, B) \in$ parameter space (for reasons being explained in the next subsection).

6.3 Finding Parameters for the Lennard-Jones Potential

As electrostatic, bond, angle and dihedral forces are not included in this simulation they have to be accounted for by choosing the parameters of the Lennard-Jones potential in a way.

6.3.1 Casting it as an Optimization Problem

The general idea is that proteins, as they are given in PDB files, are approximately in an equilibrium state; this translates to

$$\|\nabla_X \mathrm{VLJ}(a_{\mathrm{PDB}}, X_a)\| \approx 0,$$

where a_{PDB} is the protein as it is given in its PDB file with internal coordinates X, consisting of the dihedral angles[16] and VLJ is the Lennard-Jones potential of a_{PDB} as defined in week 2.

[16]Note that the potential is invariant under spacial translations and rotations, so they are not taken into account for.

A, B depends on the material, so: $A = A(\text{el1}, \text{el2})$, $B = B(\text{el1}, \text{el2})$ which turns $LJ(r)$ into $LJ(r, \text{el1}, \text{el2})$, where el1,el2 $\in$ {N,C,CA,O,Rest} $=: I$.

Thus it seems natural to set the set of optimal parameters $\Theta := ((A_{ij})_{(i,j)\in I^2}, (B_{ij})_{(i,j)\in I^2})$ to

$$\Theta = \text{argmin}_\Theta \sum_{a_\text{PDB}} \frac{\|\nabla_X \text{VLJ}_\Theta(a_\text{PDB}, X_a)\|}{|\text{VLJ}_\Theta(a_\text{PDB}, X_a)|},$$

where the division through $|\text{VLJ}_\Theta(a_\text{PDB}, X_a)|$ serves to purpose of normalizing so that this procedure doesn't yield the trivial parameters Θ equals zero for all entries.

Note that the $+C$ above is necessary to avoid systems with negative potential $-V, V > 0$ which cannot be differentiated from systems that are far more unstable, but whose absolute value is the same (i.e. a system with potential $+V$).

For a good initial guess I decided to set $A_{i,j} = A_{k,l}$ and $B_{i,j} = B_{k,l}$ for all $i, j, k, l \in I$ to simplify a brute force approach by reducing it from a $\frac{|I|(|I|+1)}{2}$-dimensional optimisation problem to a 2-dimensional one.

By using reverse automatic differentiation the calculation of the gradient only results in about the double amount of time that is needed to calculate the value itself. On my laptop one evaluation of $\frac{\|\nabla_X \text{VLJ}_\Theta(a_\text{PDB}, X_a)\|}{|\text{VLJ}_\Theta(a_\text{PDB}, X_a)|}$ takes approximately one second which makes brute forcing in more than two dimensions impossible pretty fast.

So the current approach is:

- set $A = A_{ij}$ and $B = B_{ij}$ and choose them as

$$(A, B) = \text{argmin}_{A,B} \sum_{a_\text{PDB}} \frac{\|\nabla_X \text{VLJ}_{A,B}(a_\text{PDB}, X_a)\|}{|\text{VLJ}_{A,B}(a_\text{PDB}, X_a)|}.$$

- use them as initial guess for some optimisation algorithm like gradient descent which works in a reasonable amount of time even in $\frac{|I|(|I|+1)}{2}$ dimensions.

6.4 Using the AutoGrad Library

6.4.1 Using a Library versus Manually Implementing Reverse Automatic Differentiation

6.4.2 Common Pitfalls Using AutoGrad

7 Week 10

7.0.3 Problems With Overdetermined Optimization

References

Nadler, Boaz, Ronald R. Coifman, Ioannis G. Kevrekidis, Stéphane Lafon, and Mauro Maggioni (2008), "Diffusion maps, reduction coordinates, and low dimensional representation of stochastic systems." *Society for Industrial and Applied Mathematics*, URL `http://www.wisdom.weizmann.ac.il/~nadler/Publications/diffusion_map_MMS.pdf`. Multiscale Model. Simul.

Neumaier, Arnold (2006), "Molecular modeling of proteins and mathematical prediction of protein structure." URL `http://www.mat.univie.ac.at/~neum/ms/protein.pdf`.