Iterative Knowledge-Based Scoring of Protein-Ligand Complexes

Thanh Lai

1. Problem and goals

1.1. Broad problem defined at the beginning of the project

Protein-ligand binding is a process that occurs in all biological scope—from endogenous ligand-receptor binding for signal transduction, to selective drugging of a protein target. Therefore, it is of utmost importance to understand the thermodynamics and kinetics that govern such process, as doing so has implications for fields such as drug discovery. One can gauge the stability of a protein-ligand complex by considering only the thermodynamics of the binding, however this consideration has multiple levels of theory, from empirically-derived scoring functions to *ab inito* calculations. In the early stages of a computational drug discovery program, scoring functions are favored as a quick method of filtering thousands of protein-ligand (ligand = drug candidate) complexes. However, as scoring functions trade accuracy for computing speed, it is not far-fetched to say that the trajectory of a drug discovery program greatly hinges on identifying strong drug candidates from a pool of several thousands. I want to replicate the codebase of an iterative knowledge-based scoring function called ITScore [1]. A knowledge-based scoring function assumes that information from co-crystallized structures of protein-ligand complexes can be used to construct atomic potentials. These atomic potentials are then used to evaluate (or “score”) the stability of protein-ligand complexes [2]. By making ITScore open source, one can improve on it by changing the training data or the mathematics that governs it.

1.2. Conceptual/technical motivations for choosing this problem

I chose this project because of its complexity and dataset. Conceptually, I am interested in the idea that we are able to extract atomic potentials from observations in nature (co-crystallized structures of protein-ligand complexes). With regards to the dataset, I’ve worked a lot with protein-ligand datasets before, so I am familiar with their format (called “PDB”) and how to obtain them. However, I’ve never used protein-ligand data in the context of python before (other than machine learning). The problem is intriguing to me because I have to write code to extract pairwise distance information from the PDB dataset (the “knowledge” in knowledge-based scoring), and so I would need to learn how to use a module named Biopython [3]. Finally, my research has always been on the side of applications, so I’ve always aimed for methods development projects in my classes. I’ve never worked with developing scoring functions before, so this project was a good way for me to try it out.

1.3. Changes in project goals during the semester

There are two major changes to this project. Firstly, the authors of ITScore, Huang et. Al., categorize each atom in the PDB files as one of the 26 SYBYL atom types. In the beginning of the project, I did not realize that PDB files do not include SYBYL atom types, but instead

labels each amino acid atom based on their position, and labels each ligand atom arbitrarily. I wrote the code assuming that the atom labels in the PDB data are SYBYL types, and generated wacky-looking potentials. Recently, I've figured out a way to incorporate SYBYL atom types into my code (will be elaborated later in the report), though it is very messy and does not encapsulate all 26 atom types used by Huang et. Al.

The second major change in the project goal is to forgo the iterative process of the ITScore training step. My dataset contains over 500 protein-ligand PDB files, and I need to generate 200 "decoy" structures for each PDB file by docking the ligand onto the protein. I could not find a way to automate this step since special care needs to be taken for each protein-ligand system. A lot of manual preparation has to be done for the protein before docking a ligand onto it. For example, one needs to adjust the pH to a physiological state, create a grid box of the binding cavity (different for each protein and requires manual inspection), etc. This takes considerable effort to do manually for 500+ proteins. My project has the code to calculate the initial potentials from the PDB files, and code that uses the atomic potentials to score a complex.

2. Datasets

2.1. Data plan at the beginning of the project

The initial training and testing dataset will be from the list of protein-ligand crystal structures curated by Huang et. Al (formatted as PDB files from the RCSB database [4]). The selected PDBs fulfills the following criteria: the resolution is greater than 2.5 Å; the protein-ligand interaction is noncovalent and the ligand is not DNA, RNA, or peptide (for information on scoring functions for these kinds of systems, please see future publications by Huang et. Al.); the ligands must contain between 5 to 66 heavy atoms; the crystallization must be at normal pH conditions (6.5 to 7.5); the structure must contain no metal ions other than Na+ and K+ near the binding site; and finally, complex must be absent of steric clashes (<1.75 Å for all protein-ligand atoms). The curated dataset contains a total of 786 complexes. As of February 14th 2021, 677 out of 786 of these complexes are still present in the RCSB database and will be used for this study.

2.2. Progress with data plan

The data plan progressed well until half way through the project when I realized there are problems with SYBYL atom types.

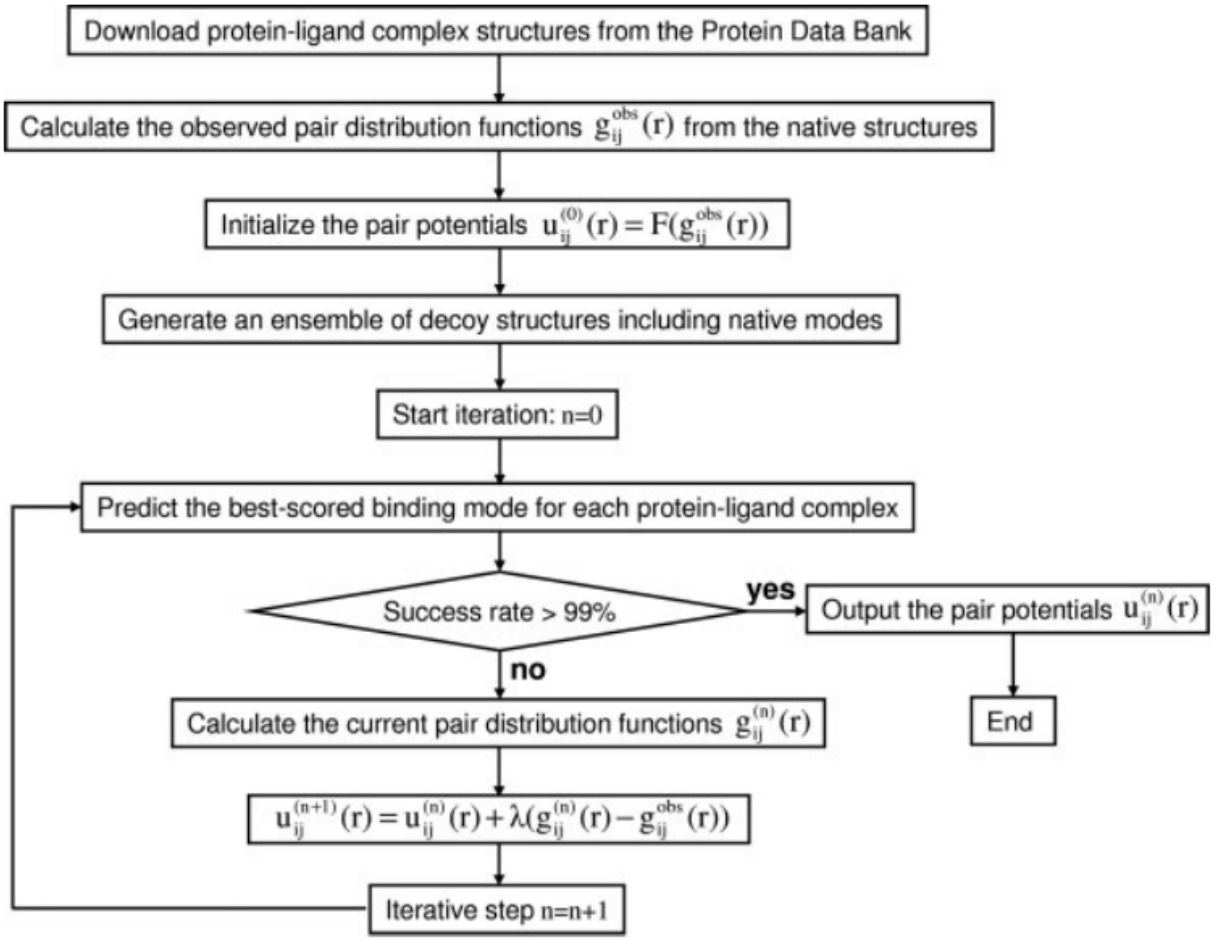
2.3. Changes to data plan in the later half of the semester

As mentioned in **section 1.3**, the PDB files do not contain SYBYL atom types. Briefly, SYBYL atom types are short strings that describe an atom based on its element, hybridization, and bonding. For example, "C.ar" means that the carbon is aromatic. "N.3" means that the nitrogen is sp^3 hybridized. I obtained these SYBYL atom types by converting the PDB files into mol2 files via the OpenBabel software [5]. Mol2 files are similar to PDB files in that they contain

the atoms, their coordinates in 3D space, and the atoms they are bonded to. These files also label each atom via the SYBYL atom type convention.

3. Computational approach

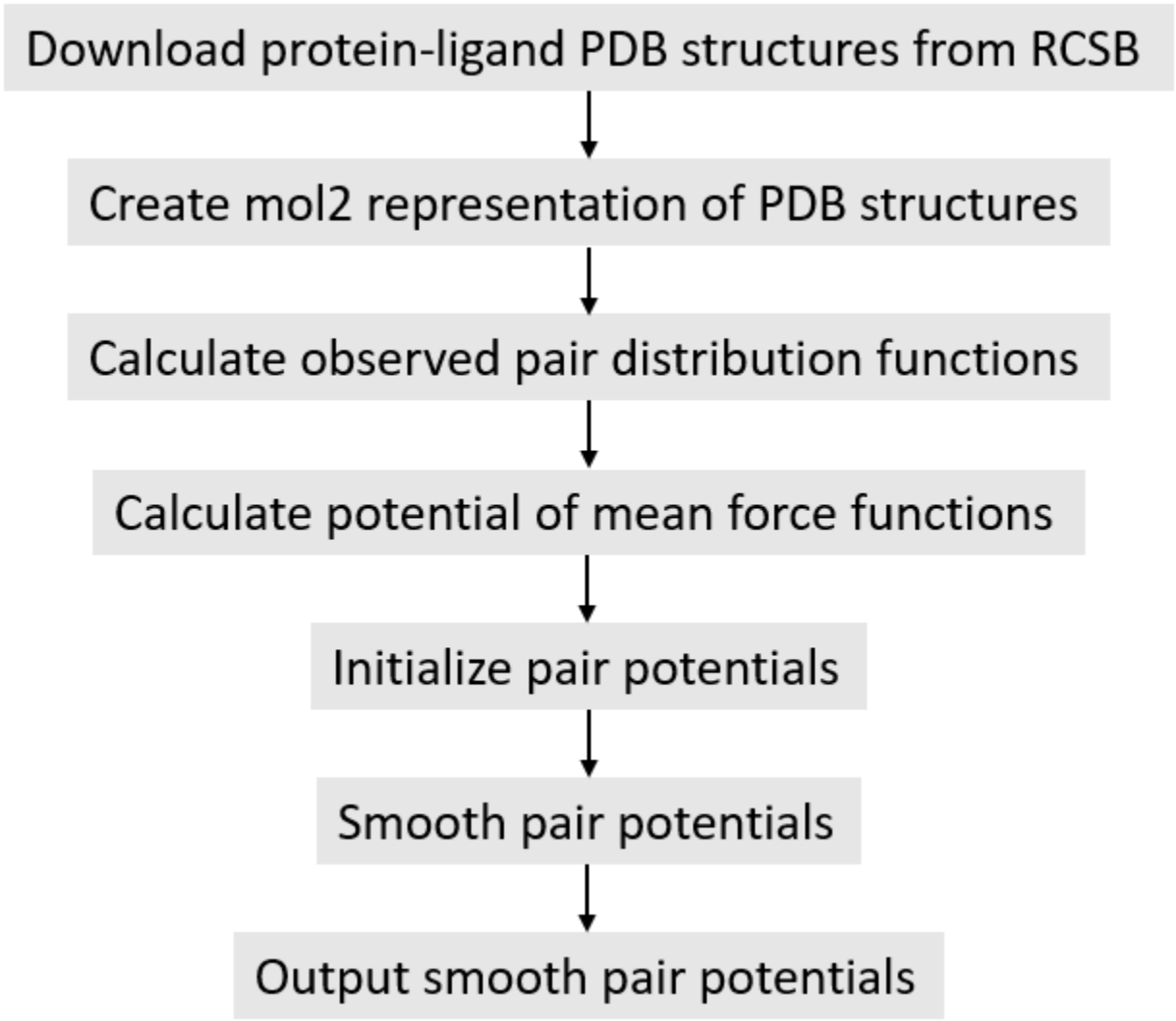
3.1. Flowchart of the originally proposed approach



The original plan is based on the flowchart from Huang et. Al. The first step is to obtain the protein-ligand complex PDB structures from the RCSB data bank (“native” structures). Next, the observed pair distribution functions $G_{obs}(r)$ are calculated from the protein-ligand structures. There is a $G_{obs}(r)$ function for each protein atom, ligand atom pair. Essentially, this function describes the occurrences of the protein atom in a radial shell of distance **r** centered at the ligand atom. At the end of this calculation, there will be a $G_{obs}(r)$ function for every protein atom, ligand atom, pair. The $G_{obs}(r)$ function is used to calculate the initial atomic potential of the protein atom, ligand atom pair. Next, a docking program is used to create 200 decoy structures for each native structure. The iterative training starts off by scoring all of the decoy and native structures by summing the potential energy of every protein atom, ligand atom pair using the calculated atomic potentials (if this is the first iteration then the initial atomic potentials are

used). For each protein-ligand complex, the “best” structure is defined as the one with the best score. If the atomic potentials are able to discriminate >99% of the native structures from the decoy structures, then the training ends. However, if the potentials discriminate less than 99% of the structures then it is recalculated. The current pair distribution function of the iteration, $G_{ij}^{n}(r)$, is calculated from the decoy structures selected by the atomic potentials. The new atomic potential is constructed by adding on the difference between the $G_{ij}^{n}(r)$ and $G_{obs}(r)$. The process repeats until the atomic potential can discriminate 99% of the structures from decoy structures.

3.2. Proposed approach / analysis plan



The new approach is writing the code up to the generation of decoy structures. This will output the smoothen potentials, but these are still crude as they aren’t refined by the iterative process.

3.3. Approach taken
Step 1: Download protein-ligand PDB structures from RCSB
Goal: Obtain the PDB structures from the RCSB dataset

Approach: Went to https://www.rcsb.org/ and downloaded all of the PDB codes used by Huang et. Al.

Outcome: The website gave a zip file that I had to unzip, and there were about 100+ obsolete PDB files

Learnings: My first time downloading multiple PDB files at once. Wasn't hard at all and I was glad the website provided a way to do so (at first I considered writing a python script to download each PDB from the website). I found out there was such a thing as obsolete PDB files.

Step 2: Extract ligand PDB from complex PDB
Goal: From a complex PDB file, save the ligand as a separate PDB file and complex PDB file into a directory called "train_decoupled/"

Approach: I used Biopython to iteratively scan through all residues in the complex PDB file and save the ones that are labeled as "HETATM" (usually the standard labeling for any molecules not part of the protein) and are not "H2O" (water).

Outcome: I now have a folder named "train_decoupled/" that contains the ligand pdb and protein pdb files. My approach assumes that every HETATM residue is a ligand in a given PDB file. This is false. For example, cofactors, coactivators, metal centers, etc residues. are often labeled as HETATM. So my approach extracts the ligand (often a co-crystallized drug molecule) as well as other kinds of molecules that one would not associate as a "ligand". I could not find a way to distinguish these ligands without manually inspecting all 500+ PDB files. Therefore I anticipate that my generated potentials will be different from Huang et. Al. since they only used the actual ligands in the calculation while ignoring cofactors, coactivators, etc.

Learnings: There is probably a better way of distinguishing ligands from PDB files. My method results in multiple ligands per protein receptor. This means that the observed pairwise distribution function will have more samples. Intuitively, this may lead to more accurate results since more pairwise distances are counted.

Step 3: Convert PDB to mol2 files
Goal: The PDB files in "train_decoupled/" are converted to mol2 files and saved into "train_decoupled_mol2/"

Approach: I downloaded and compiled Openbabel on HPCC and made a script to convert all PDB files found in "train_decoupled/" to mol2 files.

Outcome: I now have a folder named “train_decoupled_mol2/” that contains the mol2 version of the ligand and protein structure files.

Learnings: I learned how to compile a program after cloning it from github. I would usually use conda to avoid having to manually compile libraries. However, I couldn’t get the Openbabel library to work on anaconda. I spent a couple hours finding out how to clone & compile directly from the git repository. It involved cloning it, creating a separate folder and doing “cmake” to build the make files and “make” to compile the program. Finally, I edited my bashrc file to add openbabel to my path and wrote a bash script to convert PDB -> mol2.

Step 4: Count number of (protein atom, ligand atom) occurrences in radial shells

Goal: Count number of (protein atom, ligand atom) occurrences w.r.t. radial shells using both PDB and mol2 files.

Approach: The code stores the count in a **count dictionary** which is a dictionary of dictionaries. The key is a tuple of (protein atom SYBYL type, ligand atom SYBYL type) and the value is a dictionary that contains key: radius (0.1 to 10 angstrom), value: number of occurence. The code also keeps two **SYBYL dictionaries** that relate a given protein (or ligand) atom with their SYBYL atom type. The first dictionary’s key is a protein atom’s coordinate in 3D space (x, y, z) and the value is the protein atom’s SYBYL type. The second dictionary is a ligand atom’s coordinate in 3D space (x, y, z) and the value is the ligand atom’s SYBYL type. The program reads through all atoms in the PDB file, records their coordinates as a unique key in the dictionary, reads through all atoms in the corresponding mol2 file, matches the atom by their coordinate and assigns the SYBYL type as the value of the dictionary key. This results in a dictionary that relates the atom, uniquely identified by their 3D coordinate, and its SYBYL atom type. Next, the code iterates through all PDB ID, load in the corresponding protein PDB and ligand PDB(s). For all ligand atoms **i**. it iterates through all protein atoms **j**. Within this iteration, it iterates through a radius **r** of 0.1 to 10 angstrom in increments in 0.1 angstrom, and uses Biopython’s NeighborSearch function to count the number **j** protein atoms within the distance of **r**±0.05 angstroms of the ligand atom **i**. This is achieved by subtracting the set of the upper bound neighbors (**r**+0.05 angstroms) with the set of the lower bound neighbors (**r**-0.05 angstroms). Finally, for the **ith** ligand atom and **jth** protein atom it retrieves their respective SYBYL atom type from the **SYBYL dictionaries** using their 3D coordinates as the key. Once the SYBYL atom types for the ligand and protein atoms have been obtained, it initializes a key within the **count dictionary** of (protein atom SYBYL type, ligand atom SYBYL type) and sets the value as a dictionary whose key is **r** and value is the number of occurrences. This process occurs for all protein atoms, ligand atoms, and radial distances in all PDB files found in the “train_decoupled/” directory.

Outcome: The **count dictionary**, which is in the form of dict{key: tuple(protein SYBYL, ligand SYBYL), value: dict{key: float radius, value: int occurrence}}, is serialized by the Pickle library into a file called “p_obs.pkl”.

Learnings: I originally wrote a code without mapping the SYBYL types and it took only a few seconds to run. When I had to rewrite the code to include the SYBYL mapping, it took over 5 minutes to run because it had to read through every mol2 file and build a SYBYL dictionary. It took me weeks to think of a way to associate each atom in the PDB file with their SYBYL type from the mol2 file. After trying to find ways to read in mol2 files with Biopython, I figured that I could map each atom from the PDB file with their SYBYL type in the mol2 file using their 3D coordinates. Although there is a better way to do this (maybe Biopython has a hidden SYBYL function?), it served the purpose well. Another thing I learned is that Biopython has a very very useful function called NeighborSearch, which has a C backend and uses some fancy graph theory for search efficiency. This was a huge deal because I planned to write my own iteration function to do the neighbor search.

Step 5: Calculation of the observed pairwise function $G_{obs}(r)$

Goal: Calculate $G_{obs}(r)$ for all (protein atom, ligand atom) pair from the **count dictionary**

Approach: The program first filters out all (protein atom, ligand atom) pairs that have less than an occurrence of 500. To do this, it iterates through all keys of the **count dictionary** and adds up the occurrence of all radius keys and delete the keys (pairs) with less than 500 occurrences. Next, the program initializes a **G dictionary** whose keys are (protein atom, ligand atom) pairs and values are a dictionary of key: radius, value: **G value**. The **G value** is calculated as:

$$g_{ij}^{\mathrm{obs}}(r) = \rho_{ij}^{\mathrm{obs}}(r) / \rho_{ij,\mathrm{bulk}}^{\mathrm{obs}}$$

$$\rho_{ij}^{\mathrm{obs}}(r) = \frac{1}{M} \sum_{m}^{M} \frac{n_{ij}^{m}(r)}{4\pi r^2 dr} \quad \text{and} \quad \rho_{ij,\mathrm{bulk}}^{\mathrm{obs}} = \frac{1}{M} \sum_{m}^{M} \frac{N_{ij}^{m}}{V(R_{\mathrm{max}})}$$

Where n(r) is the number of (protein atom, ligand atom) occurrence at the radial shell of r, dr is equal to 0.1 angstrom, M is the total number of protein-ligand complexes analyzed, N is the total number of occurrences within a sphere of 10 angstroms, and $V(R_{max})$ is the volume of a sphere with radius of 10 angstroms. A loop iterates through all the keys of the **count dictionary** and calculates the **G value** at radius **r**. Then it stores the **G value** into the **G dictionary**.

Outcome: The **G dictionary** is serialized by Pickle as “g_obs.pkl”. The **G dictionary** is of the form dict{key: tuple(protein SYBYL, ligand SYBYL), value: dict{key: float radius, value: float G value}}.

Learnings: The creation of the **G dictionary** was easier than I expected, only involving 2 for loops. There wasn’t much to learn from this step.

Step 6: Calculation of the potential of mean force (PMF)

Goal: Calculate PMF for all (protein atom, ligand atom) pair from the **G dictionary**

Approach: Initialize the **PMF dictionary** with the atom pairs being the keys and values being a numpy array containing the potential of mean force (each element of the array corresponds to a radius from 0.1 to 10 angstrom). The pairs are iterated through the **G dictionary**, and for each **G value** the following equation is applied:

$$w_{ij}(r) \equiv -k_{\mathrm{B}}T \ln g_{ij}^{\mathrm{obs}}(r).$$

where $w_{ij}(r)$ is the potential of mean force of protein atom **i**, ligand atom **j** at distance **r**. k_B is the boltzmann constant,T is the temperature. k_BT is equal to 1 in this program.

Step 7: Calculation of the initial pair potentials

Goal: Use the **PMF dictionary** to make the initial pair potentials. These are called the initial pair potentials because in the original paper they are later refined into the final pair potentials by the iterative training process.

Approach: The **epsilon dictionary** and **sigma dictionary** are created with the keys being an element (represented as a string) and the value being either the lennard-jones well depth or bond length parameter from the AMBER molecular dynamics suite **[cite]** respectively. A list, **atom pairs**, consisting of (protein atom, ligand atom) tuples are created from the keys of the **epsilon dictionary**. Essentially, this list is used to filter out any atom pairs in the **PMF dictionary** that contain elements whose lennard-jones parameters are not defined by AMBER. The program iterates through the pairs in the **PMF dictionary** and does the following calculation on the PMF key (array) only if the elements in the pair are defined by AMBER (by checking if the pair is in the **atom pairs** list):

$$u_{ij}^{(0)}(r) = \begin{cases} w_{ij}(r) & \text{for hydrogen-bond pairs} \\ \dfrac{v_{ij}(r)\, e^{-v_{ij}(r)} + w_{ij}(r)\, e^{-w_{ij}(r)}}{e^{-v_{ij}(r)} + e^{-w_{ij}(r)}} & \text{otherwise} \end{cases}$$

where $w_{ij}(r)$ is the **PMF** at **r**, and $v_{ij}(r)$ is the 12-6 lennard jones potential (the lennard jones potential calculation uses values from the **epsilon** and **sigma dictionary**). The 12-6 lennard jones potential uses a geometric mean for the mixing rule of epsilon and arithmetic mean for the mixing rule of sigma (Huang et. Al. did not specify their mixing rule). To account for hydrophobic interactions, the **epsilon dictionary** contains well-depth values that are 3 times higher than specified in AMBER. The calculated values are stored in the **initial pair potential dictionary**, where the key is the atom pair tuple, and the value is an array of the potential energy at each distance (0.1 to 10 angstrom in 0.1 increments).

Learnings: I learned where to obtain the AMBER lennard jones parameters. My initial potentials look much better after I realize that I was using the wrong value for $v_{ij}(r)$. This also refreshed my memory on lennard jone parameter mixing rules.

Step 8: Smoothing of initial pair potentials

Goal: Use a smoothing function to improve the pair potential curves. At this stage the pair potential curves are very ragged due to statistical fluctuations in the data.

Approach: The following piecewise smoothing function is used:

$$F(r) = \begin{cases} \kappa(r_{\min} - r)^s - \epsilon_0 & r < r_{\min} \\ \epsilon_0(e^{-\alpha(r-r_{\min})} - 1)^2 - \epsilon_0 & r_{\min} \leq r \leq r_c \\ 0 & r > r_c \end{cases}$$

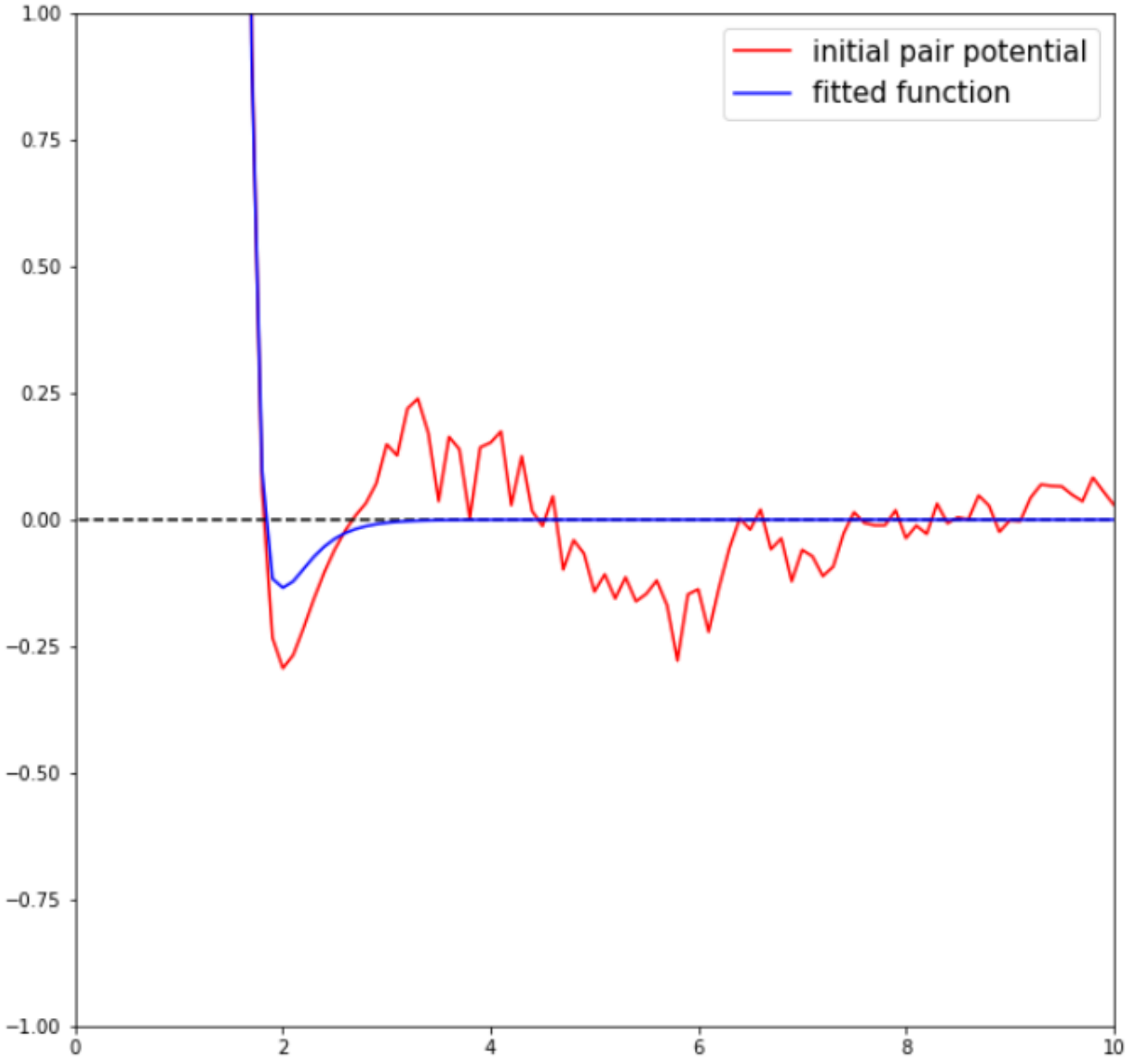
r_{min} is the distance at the potential minimum, r_c is the cut-off distance at which interactions are assumed to be 0 (6 angstroms), ε_0 is the epsilon value of the atom pair (geometric mixing rule), κ, α, and s are least squares fitted parameters. We know that the initial potentials are supposed to look like a morse potential. A morse potential behaves similarly to a lennard jones potential except after r_{min} it has an anharmonic curvature. Since a morse potential is a better description of interatomic interactions, the initial pair potentials are fitted to a smoothing function that takes the morse potential into account. Namely, the portion of the initial pair potentials before the minimum ($r < r_{min}$) is fitted to a VDW repulsion function. The portion from r_{min} to r_c is fitted to the morse potential in order to capture the anharmonic curvature. Any value beyond r_c is zeroed to represent no interaction between the atom pair. The program uses the least squares curve_fit function from the Scipy library. For all atom pairs in the **initial pair potential dictionary**, it calculates the r_{min} value using the sigma values of both atoms in the pair. Then, it obtains a subarray of the pair potential array up to the element that corresponds to r_{min} and fits the subarray to a VDW repulsion function. The program then obtains a subarray of the pair potential array that corresponds to the distance between r_{min} and r_c and fit it to a morse potential. Finally, it creates an array full of zeros that corresponds to the final part of the pair potential array beyond the element that corresponds to r_c. Numpy is used to concatenate the three fitted arrays into one single array. A **smooth dictionary** is created, whose key is the atom pair and value is the smoothen (fitted) array. This smooth dictionary is the final output of the program.

Learnings: This was my first time fitting a function to a piecewise function, and my method of doing it was to fit the subarrays to their respective function and concatenate them at the end. A problem I ran into was that each atom pair has their own epsilon and sigma value. In order to avoid having to create 200+ predefined fitting functions (a function for each epsilon and sigma pair), I made the epsilon and sigma as global variables in the fitting function. This allows me to vary the epsilon and sigma values during the fitting without having to explicitly define a different fitting function for each value. Of course, this is a somewhat “dangerous” solution, but I

preferred it over having to make 200+ predefined functions (or having to learn an entirely new curve fitting python library that has the ability to vary parameters).

4. Summary of key findings and learnings

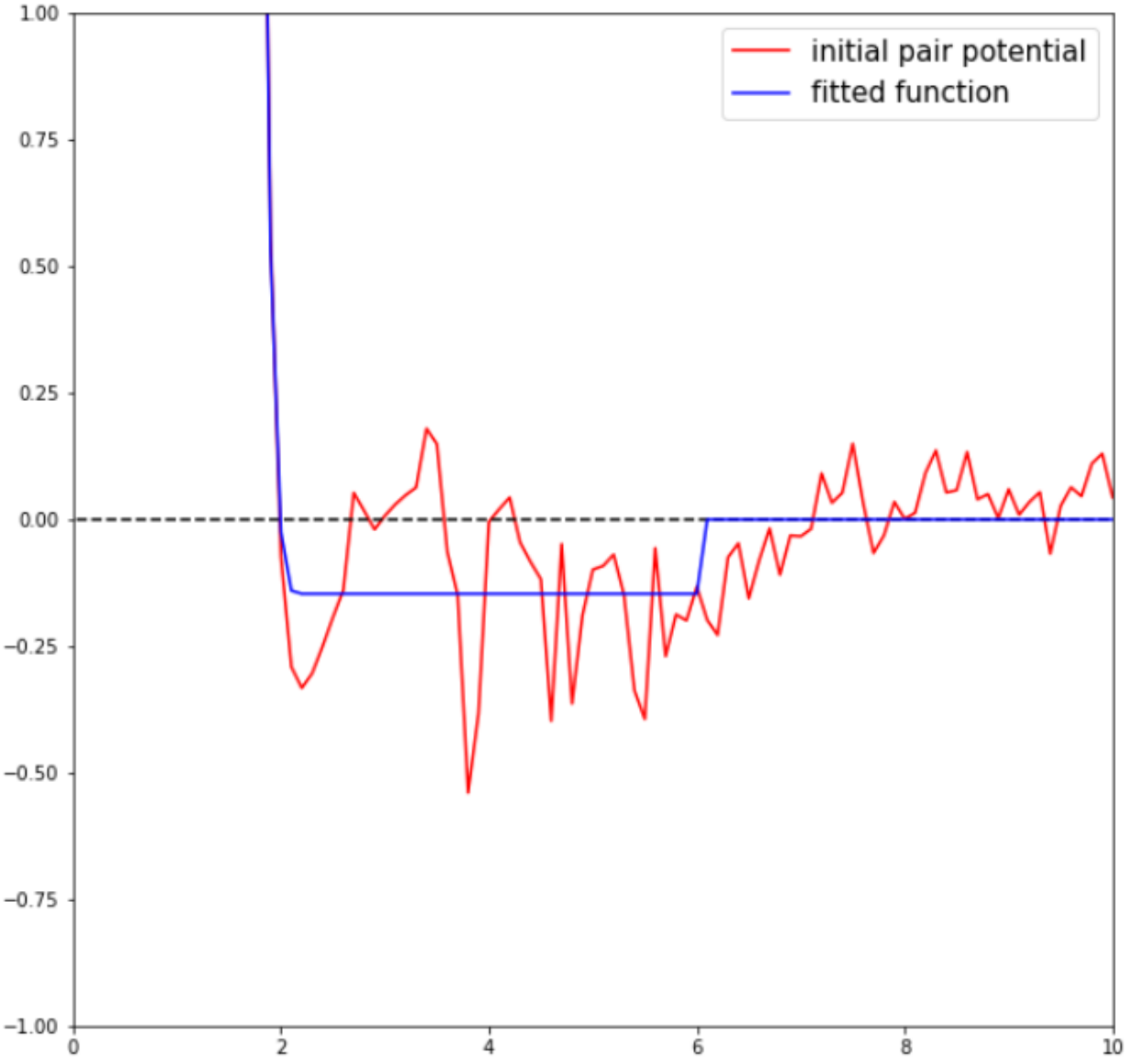
Finding / learning 1: Potential curve of (O.co2, C.ar) (example of a good potential curve)



The program produced qualitatively decent potential curves as well as egregious potential curves. The potential curve shown describes the interaction between protein carboxylate oxygen atoms and ligand aromatic carbon atoms. The red curve is the pair potential and the blue curve is the smooth/fitted curve. It is notable that the pair potential curve, although having obvious statistical noise, has a similar functional form to a morse potential. The blue curve is the

smooth function and looks very similar to a morse potential. This suggests that the (O.co2, C.ar) interaction is well captured from the training data.

Finding / learning 2: Potential curve of (S.3, C.ar) (example of a bad potential curve)



This is an example of a bad potential curve generated by the program. The pair potential curve doesn't resemble a morse potential at all, and the smooth function was not able to fit a morse potential. A possible reason for this result could be that there weren't enough interactions between sp3 hybridized sulfur protein atoms and aromatic carbon ligand atoms in the training set. Thus, the program could not capture their interaction well.

Finding/learning 3: Scoring example

VEGFR2 Complex	Score	Experimental Dissociation Constant (Kd)
1	-7.09	25e-6
2	-7.65	1.1e-9
3	-7.71	0.02e-9
4	-8.54	0.52e-9
5	-9.70	0.01e-9

I wrote a script called score.py which takes in a list of pdb id and scores the complexes based on the fitted parameters. A lower (more negative) score suggests a more stable complex. I compared it with the experimental dissociation constant (Kd) [6] which is also inversely proportional to the binding affinity. Although the scoring function is unrefined, it is able to rank the VEGFR2 complexes very well. Comparison with the experimental dissociation constant reveals a kendall tau score of 0.80 which suggests good correlation. However, it should be noted that this test compares different ligands to the same protein receptor. Scoring functions are often used to compare different docking poses of the same ligand to the same protein receptor. This is a harder task to rank because the docking poses could be very similar to each other. I would suspect that the unrefined scoring function won't do that well for scoring docked poses, and that the iterative training process is required to produce more reliable potential curves.

Code and data availability

The code is available at https://github.com/laithanh77/iterative-kb-score.git

It requires conda installations of numpy, scipy, and biopython. It also requires the installation & compilation of openbabel (openbabel should be added to your PATH as "obabel").

The code is reproducible. The only thing the user needs to do is download PDB files from RCSB and unzip it into the "train_coupled/" directory. The PDB files are listed here: zoulab.dalton.missouri.edu/download_files/ITScore_Training.txt

5. Challenges

Comments on specific results/outcomes/milestones set at the beginning

I did everything up to the iterative training part of ITScore.

Technical/scientific challenges faced

It took me nearly a week of re-reading the paper of Huang et. Al. to understand the mathematics of creating the potentials. There were a lot of software bugs and stuff in my python code but stackoverflow saved me.

Practical challenges faced

I could not find a way to automate the docking process to make decoys for each PDB file.

6. Reflection and future directions

If I started this project fresh, I would download openbabel immediately and write the code for mapping PDB files to mol2 SYBYL types. I spent a lot of time writing the code thinking the PDB files contain SYBYL types. Another thing I would do differently is look at automated docking solutions earlier on. I realized near the end of the project that I didn't have enough time to figure out a docking solution. Otherwise, I am satisfied with the project. Future directions would include implementing automated docking to generate the decoys, writing code for the iterative training process, and writing the code to score protein-ligand complexes.

7. Acknowledgements

Arjun, for telling me that there is probably a docking software preinstalled on HPCC. Turns out Schrodinger is installed, but I haven't had time to look into using the program.

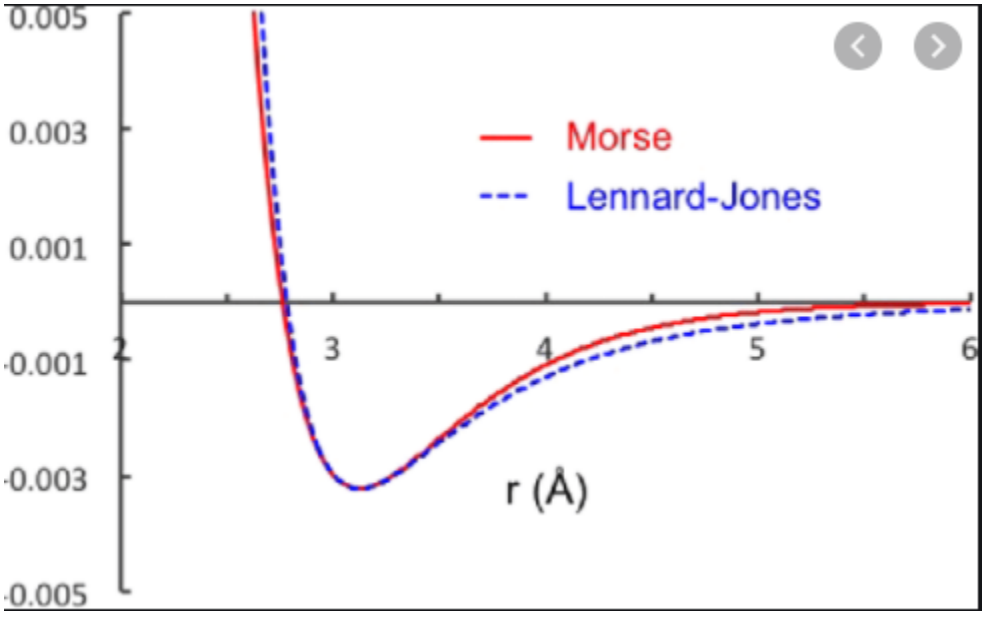
8. References

1. Huang, S.-Y.; Zou, X. An Iterative Knowledge-Based Scoring Function to Predict Protein–Ligand Interactions: I. Derivation of Interaction Potentials. J. Comput. Chem. 2006, 27 (15), 1866–1875. https://doi.org/10.1002/jcc.20504.
2. Thomas, P. D.; Dill, K. A. Statistical Potentials Extracted from Protein Structures: How Accurate Are They? J. Mol. Biol. 1996, 257 (2), 457–469. https://doi.org/10.1006/jmbi.1996.0175.
3. Cock, P. J. A.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski, B.; De Hoon, M. J. L. Biopython: Freely Available Python Tools for Computational Molecular Biology and Bioinformatics. Bioinformatics 2009, 25 (11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163.
4. Berman, H.; Henrick, K.; Nakamura, H. Announcing the Worldwide Protein Data Bank. Nat. Struct. Mol. Biol. 2003, 10 (12), 980–980. https://doi.org/10.1038/nsb1203-980
5. O'Boyle, N.M., Banck, M., James, C.A. *et al.* Open Babel: An open chemical toolbox. *J Cheminform* 3, 33 (2011). https://doi.org/10.1186/1758-2946-3-33

6. Wang, R.; Fang, X.; Lu, Y.; Wang, S. The PDBbind Database: Collection of Binding Affinities for Protein−Ligand Complexes with Known Three-Dimensional Structures. J. Med. Chem. 2004, 47 (12), 2977–2980. https://doi.org/10.1021/jm030580l.

9. Glossary

- **Knowledge Based Scoring Function:** a scoring function takes in multiple protein-ligand complexes and qualitatively ranks them based on affinity. A scoring function is knowledge-based if it uses statistical counting of crystallized protein-ligand complexes to generate statistical potential curves.
- **Potential Curve:** describes the intermolecular interaction between two atoms. Usually has a “well” which informs the equilibrium distance (distance at which the two atoms are more stable).
- **Lennard Jones Potential:** the basic formulation of describing intermolecular interactions. It uses an epsilon term which describes the well-depth and a sigma term which describes the bond length. Every element is associated with an empirical epsilon and sigma value which describes its intermolecular interaction with the same element. If the element interacts with a different element, then both element’s epsilon and sigma terms are combined to form a new lennard jones potential. The epsilon terms are “mixed” by a geometric mean and the sigma terms are “mixed” by an arithmetic mean.
- **Morse Potential:** Intermolecular interactions are best described by the morse potential which is very similar to a lennard jones potential except it has an anharmonic curvature that the lennard jones potential fails to capture.



- **Radial Distribution Function:** For a particular atom/molecule, the function essentially describes the probability of another molecule being in a certain distance from the central molecule. From my understanding, this function is often used to describe solutions such as pure water, and the function can be converted into a potential of mean force. Biophysicists decided to apply this concept to protein-ligand bindings because they

assume that ligand prefers to dock itself in a particular position such that favorable distances between pairs of atoms are satisfied.

- **Potential of Mean Force:** describes how a system's energy changes along a coordinate. In this project, the coordinate system is the distance between the protein atom and ligand atom.
- **SYBYL Types:** the SYBYL convention describes atoms based on their hybridization and surrounding. For example, "C.3" means sp3 hybridized carbon. "O.co2" means oxygen of a carboxylate. "C.ar" means aromatic carbon. By describing these atoms more specifically than just their element identity, more relationships between atoms can be formed. Therefore SYBYL types are used in this project instead of plainly labeling atoms solely by their elemental identity.