Ab-initio protein modeling using fragment matching, simulated annealing and an energy function

Caiwei Wang, Xiaokai Qian, Sean Lander,
Haipei Fan, Puneet Gaddam, Brett Koonce

University of Missouri - Columbia

March 12, 2013

1 Abstract

Our goal is to develop a simple prototype of fragment assembly template-free modeling system. First, we convert a collection of PDB's into residue-torsion pairs and use a sliding window to build a fragment database. Next, we randomly build a model our our target protein from matches in the database. After initialization, we replace a randomly chose fragment with a similar (BLOSUM) one, then score the new protein model (DFIRE), and finally decide if the new protein can be accepted according to simulated annealing. Finally, we repeat this process for many steps and save statistics and models at each step for evaluation. Ultimately, we create a movie of the top-scoring protein and discuss how to improve our process.

2 Introduction

Template free modeling is an important technique in modern bioinformatics since many proteins do not have good templates to base a computational model on. First, we begin with PDB sequences converting into torsion angles using third party tools RAMA/LIPA (part of CRONKITE). Using a sliding window algorithm, we can add millions of fragments to our database. Then, for each segment, we can do a query for a match/similar protein in the database.

We use simulated annealing to score, when the current solution's neighbor is better, accepted. If not, we select it with a probability based on the current temperature. For our project, we implemented a basic homology modeling pipeline in python, using a number of external tools like DFIRE, PyMOL and TM-score to build/present a final visualized model.

2.1 Pipeline

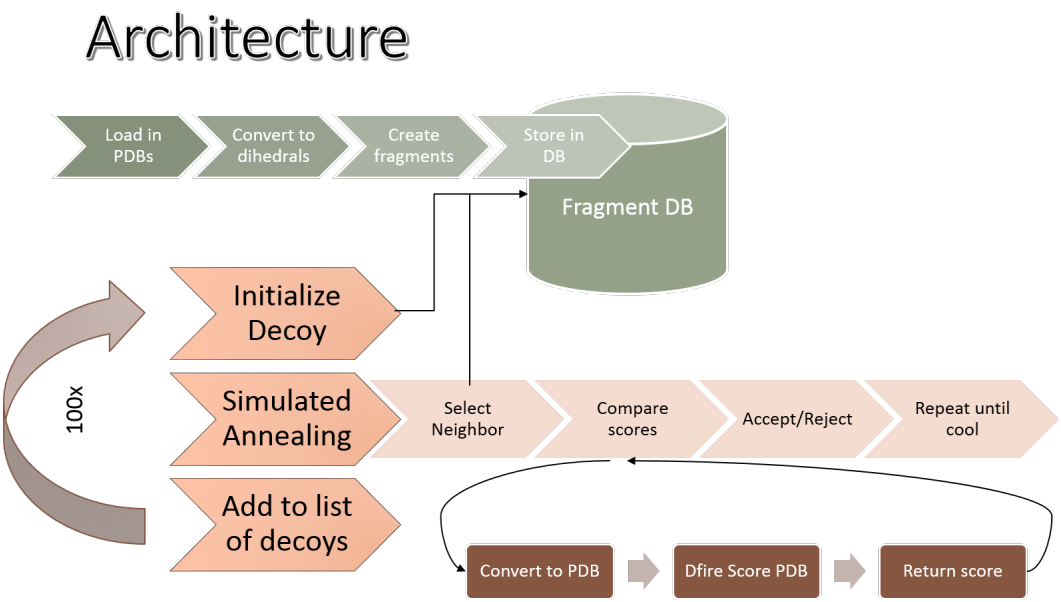

Figure 1: An overview of the TFPy pipeline

In order to more easily assemble our workflow we break it into a simple pipeline. Doing this allows for quick iteration on multiple fronts without the fear of breaking the model as a whole.

1. The first task involves building robust and well-indexed fragment databases for the three and nine length sequences. Once this is complete we are free to iterate over the simulated annealing algorithm itself.

2. Once the databases are built we create the tools necessary to initialize the candidate, choose its neighbor and score the neighbor, all of which are independent of the simulated annealing algorithm itself.

3. Initialization is less important in simulated annealing than most due to its stochastic nature, so a close approximation using fragments from the database are sufficient to build starting point.

4. Neighbor selection is more important, and fragment selection is a key component separated from the core code, as is described in our section on neighbor using Blosum62.

5. Scoring features in a similar manner, as the neighbor must be converted to a PDB file, scored with an external system, and then have its score returned and compared. In our current model D-fire energy is used to score the model.

6. Finally, multiple cooling solutions may be used, so allowing the user a choice using functional methods are preferred. We currently provide both linear and inverse-sigmoid cooling functions, with a temperature ranging from 2500 to 0 over 1000 iterations.

Creating a system in this way leads to easy parallelization, as each decoy can be generated independent of the others. This allows for rapid decoy creation and comparison with no human component necessary beyond initial input.

3 CASP Target goal

We selected the following template-free targets from the CASP database to build a model for: T0693 (100 residues), T0666 (196 residues), and T0695 (537 residues). This allows us a combination of small, medium, and large targets to test.

4 Fragment database

First, we selected a set of 17,000 PDB database with sequence lengths of 100-150 residues from an internal collection at MU. For each PDB file, we split it into a list of (residue, phi angle, psi angle) chunks. We discard the first and last residue, since they only have one valid angle. First, we go through each PDB file sequentially for each sequence of three residues:

for $i := 1$ to $(length(PDB) - 1) - 3$ step 1 do
    $a := PDB[i], b := PDB[i + 1], c := PDB[i + 2]$
    $addToFragmentTrimersDatabase(a, \phi_a, \psi_a, b, \phi_b, \psi_b, c, \phi_c, \psi_c)$
end

Then, we repeat the process for each sequence of nine residues:

for $i := 1$ to $(length(PDB) - 1) - 9$ step 1 do
    $a := PDB[i], b := PDB[i + 1], c := PDB[i + 2], \ldots$
    $addToFragmentNinemersDatabase(a, \phi_a, \psi_a, b, \phi_b, \psi_b, c, \phi_c, \psi_c, \ldots)$
end

Finally, we create an index for each sequence in our databases, in order to speed up future lookups. This produces a database of approximately 1.7 million sequence fragments of a reasonable size (Trimer: 150MB, Ninemer: 500MB) which can be searched very quickly.

4.1 BLOSUM62 selection

Neighbor selection is an important part of greedy and semi-greedy algorithms, as changes which are too large can disrupt the process of finding global minima and maxima. In order to generate neighbors which are close to the current sequence we use a two-fold approach: find a complete match; if a match is not found, find a match for a very similar sequence. The first approach is easy to understand, but the second requires domain knowledge of amino acids, as some acids are more similar than others.

$$S_{ij} = \left(\frac{1}{\lambda}\right) \log\left(\frac{p_{ij}}{q_i * q_j}\right)$$

The Blosum62 matrix is a pair-wise "distance" matrix of sorts which can be used to determine the similarity of two amino acids. This matrix is commonly used to determine sequence similarity for alignment purposes, but can also be used as a generative model. By finding similar amino acids to those in the current fragment sequence, we can generate multiple sequences which are very similar in terms of "distance," allowing us to expand our search criteria on limited databases.

5 Simulated annealing algorithm

```
# Simulated Annealing for Score Maximization
WHILE count < max_steps:
          temperature := temperature - cooling_rate
          candidate := solution.neighbor()
          score_difference := score(solution) - score(candidate)

          # Candidate score is worse than current solution
          IF score_difference > 0:
                    # Higher the temp, closer prob_to_accept gets to 1
                    prob_to_accept := e^(score_difference/temperature)

                    # Select from uniform distribution. If higher than
                    # prob_to_accept skip that candidate and try again
                    IF prob_to_accept > random():
                              continue
                    END
          END
          solution := candidate
END
```

5.1 Simulated Annealing

After building our fragments database, we build a initial model of our target using sequences from the database. Then, to improve our model, we use simulated annealing, a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete and can prove quite efficient. Unlike hill climbing, simulated annealing can allow a poorer fragment replacement (a weaker score) early on when the function has a high temperature, which often leads to a better overall model structure.

In each cycle, we build a random neighbor of the current best solution. If the neighbor's score is better, we promote it to the current solution. Otherwise, we select one of the two with a probability based on the current global temperature. Allowing a "bad" fragment replacement can allow the SA algorithm to avoid local optima.

This cooling schedule is an important factor in the SA algorithm. Choosing the correct

parameters can efficiently and quickly help us find the optimal solution. In our project, we tested both sigmoid and linear models. We used the following candidate functions/parameters for our project:

$$T_s(k) = -5000/(1 + exp(-k/200)) + 5000$$
$$T_l(k) = (-2500/1000) * k + 2500$$

5.2 d-DFIRE score

To provide an objective score of our new protein for the purposes of picking which model to accept, we use DFIRE. The DFIRE-based statistical potential is a more physically accurate potential than other all-atom statistical potentials because the potential satisfies the physical requirement that the same water-mediated interaction between amino acid residues is responsible for folding and binding. This potential is more accurate for longer loops (9 to 12 residues). The performance of the DFIRE potential for loop selections is useful as well because the computational cost of a statistical potential is only a fraction of what is needed for physical-based energy functions with implicit solvation models. As such, a single-term DFIRE-statistical energy function can provide an accurate loop prediction at a fraction of computing cost required for more complicate physical-based energy functions.

6 Visualization

After we obtain a PDB file for each step of the simulated annealing process, we use PyMOL to visualize the process by displaying each model, then playing them back sequentially in the same order as they were generated. PyMOL is an open source, OpenGL based molecular visualization system. It excels at 3D visualization of proteins, small molecules, density, surfaces, and trajectories. It also includes molecular editing, ray tracing, and movies. PyMOL can produce high-quality 3D images of small molecules and biological macromolecules, such as proteins. We loaded our PDB temporary files (produced at each simulated annealing step), and utilized PyMOL's cartoon mode to see the way the algorithm procedurally improves the protein backbone.

7 Results

As before, our targets are: T0693 (100 residues), T0666 (196 residues), and T0695 (537 residues). Each score contains the best result from the 9-sequence replacement pass, followed by the score from the best 3-sequence replacement pass.

7.1 Scores

T0693

	Linear - 9	Linear - 3	Sigmoid - 9	Sigmoid - 3
dDFIRE	-175.6	-177.39	-180.98	-194.71
TM-Score	0.0891	0.0745	0.1162	0.0908
RMSD	16.856	16.868	14.447	14.554

T0666

	Linear - 9	Linear - 3	Sigmoid - 9	Sigmoid - 3
dDFIRE	-75.27	-121.34	-82.85	-125.35
TM-Score	0.16113	0.162	0.1779	0.1565
RMSD	23.162	22.569	25.193	30.457

T0695

	Linear - 9	Linear - 3	Sigmoid - 9	Sigmoid - 3
dDFIRE	-372.27	-375.93	-425.11	-431.09
TM-Score	0.1061	0.1362	0.13	0.119
RMSD	36.765	37.403	39.106	41.541

7.2 Future improvements

Finally, we want to check whether our refinement is actually valid and find out which cooling schedule function improves our model the most. So, we plot three histograms for each target to compare their results using RMSD. From figures, we can find that our 3-sequence refinement can improve the quality of predictions, but only by a little bit. The sigmoid function performs better than our linear function with respect to DFIRE scores, but in general we cannot claim that it produces better results when we look at the objective RMSD and TM-Score functions. We need to test more targets.

8 Citations

We thank the following tools and papers:

Zhang, C., Liu, S., and Zhou, Y., Accurate and efficient loop selections using DFIRE-based all-atom statistical potential., Protein Sci. 13, 391-399 (2004).

PyMOL: http://sourceforge.net/projects/pymol/

Podtelezhnikov, Alexei and Wild, David. CRANKITE: A fast polypeptide backbone conformation sampler, Source Code for Biology and Medicine. Vol 3, 2008.
Url: http://www.scfbm.org/content/3/1/12

CASP: Critical Assessment of Techniques for Protein Structure Prediction.
http://predictioncenter.org/casp10/

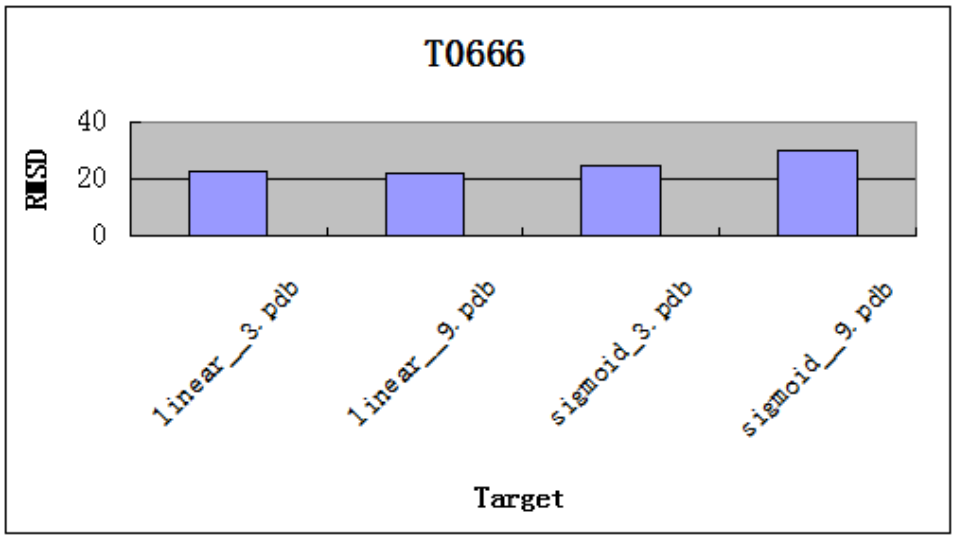

Figure 2

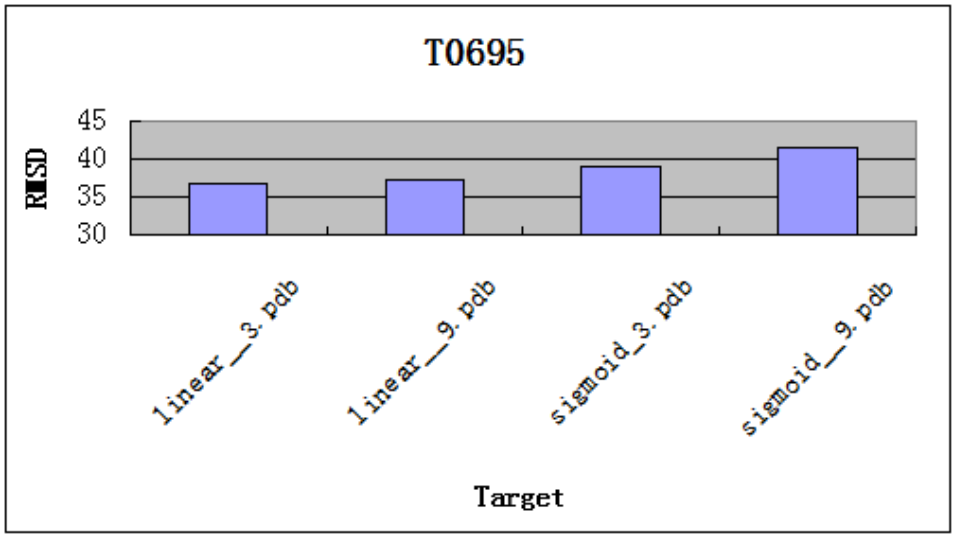

Figure 3

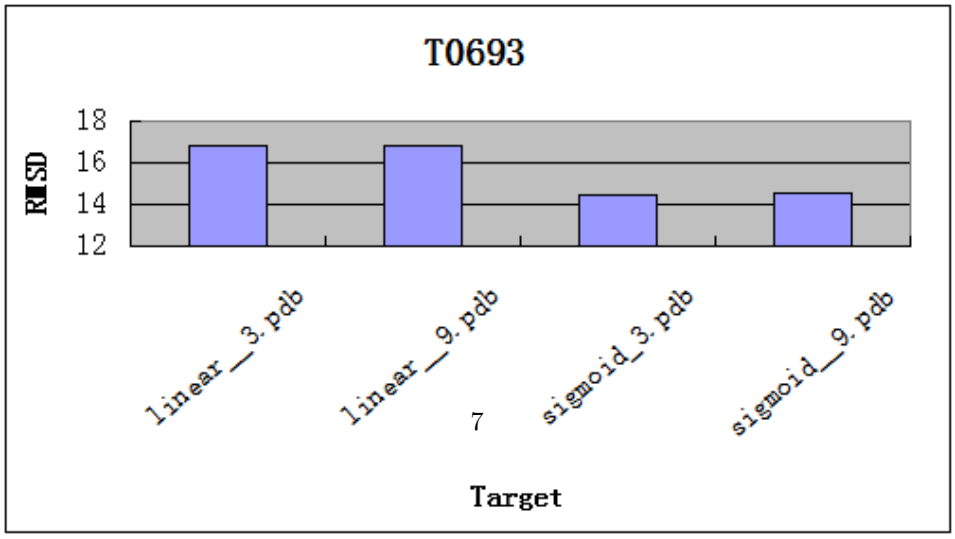

Figure 4