VULCAN - MAWS Software Improvement Team

Antoine Baize, Hannah Collins, Mariafernanda Hernandez, Nicholas Simone

Project Workflow - What we have tried

- Familiarize ourselves with MAWS.
- Run MAWS
    - Develop virtual machine to run MAWS on a virtual LINUX platform
        - MAWS required installation through conda however the installation required a lot of knowledge of the MAWS code due to it not being able to pull the MAWS files and compile the software
        - We were able to install MAWS correctly and access the software however it would not run correctly when given an input file.
        - Troubleshoot to develop pre-processing of our input pdb files. This is surmised from the README's on the MAWS Github, but this did not solve our problem. MAWS was still not functional.
        - Troubleshooting, we found that even running with valid pdb files ended up with output files that did not correctly extract the molecular information.
        - Project pivot in light of MAWS non-functionality, with some guidance we found pytraj, GitHub repository is here
- pytraj is seen as a viable option to replace the Amber call system MAWS uses - this was our goal to revive MAWS and make it functional - we believed pytraj would be able to extract all of the molecular information used to inform aptamer construction.
    - Viability included testing pdb input (same as MAWS flow) and formatting to get file into nested array format seen in MAWS, pulling atomic coordinates in the protein, uncovering residues and other structural features of the protein we load in.
    - More viability included altering .pdb files and writing to new .pdb files, similar to MAWS' ability to create a new .pdb of aptamer and append the optimal nucleotides and their coordinates.
- Investigated more into pytraj, documentation provided here .
    - Energy related functions (e.g. energy decomposition function) all require a library (sander) that has since been removed.
        - This was a vital function we were leaning on being able to leverage for entropy calculations.
- Alternatives to discover energy state information were investigated.
    - An alternative was Rosetta which requires licensing and high computing power, but would have been able to understand aptamer folding. Documentation for Rosetta can be found here .
- PYMOL development as a method of validating our results. The MAWS group used aptamers as substitutes for antibodies in immunoassays, but since we had no access to experimental techniques, we wanted to visualize overall structure to provide some validity to our results.

    - PYMOL is a molecular visualization software that shows the crystalline structure of proteins and other molecules. This would have been included in our workflow to show a completed aptamer that conforms in shape to the binding site.
- We attempted to pull the other MAWS libraries that they use. This would have allowed us to take the good parts of MAWS and have it be replaced with our pytraj calls.
    - One of the most important libraries was simtk which we discovered was no longer working. This library is called to conduct molecular force calculations given the .pdb file information. Torsion forces, and other bonding forces are used in entropy calculations.

Failures - What did not work

- MAWS code libraries, namely simtk no longer functional
    - Simtk used for molecular simulations to uncover torsion forces in the bonds, important in geometry and entropy calculations
- Amber calls they make are non-reproducible
    - Our plan was to replace these with our own calls to Amber using pytraj
- pytraj removed support for sander, a library that would have provided a coordinate-based energy decomposition of the protein that the aptamer was to be created for
    - The sander library was an essential piece for calculating entropies
- The MAWS creators have stepped away from the software, making troubleshooting difficult, especially since we cannot maintain communication with them.
- Overall size of the program, we bit off more than we could chew in terms of difficulty level, scope of the problem, and addressing the issues that MAWS has.

Summary of Failures

We planned on implementing alternative Amber calls as this would allow for a reproducible version of the MAWS software. In general, we searched for alternative libraries that can provide the same molecular information that Amber uses. We found a library that is more up-to-date than the original Amber, known as pytraj, that should have allowed us to pull molecular coordinates, trajectories, energy states, and other information regarding bond interactions. This pytraj library is also from Amber, but would function as our more general alternative method of calling molecular information that MAWS requires. However, our energy decomposition analysis functions in pytraj require access to other simulation softwares, mainly the sander library which has since been discontinued. This is currently the main barrier to implementing our solution with pytraj.

Another failure of the project is the fact that MAWS itself is not functional. We spent a large amount of time struggling to understand why, but it appears to stem from a mixture of factors. The custom Amber implementation is the first one that was brought to our attention, but after unveiling more information about their libraries, they use other unavailable libraries (simtk, as an example, which is for molecular dynamics information). Simtk and its functions appear to be called to assess Torsion forces and other molecular forces within the protein and aptamer. Other Git issues have been opened by other users being unable to run code, but since the Heidelberg team has disbanded, there is no one to contact regarding these issues and what other issues may be under the hood of MAWS. While we have struggled to get MAWS to function on its own and failed, we still had the belief that we could troubleshoot and fix the

issues it had. We admit that this is a naive assumption and this is a large contributor to the challenges we faced throughout the project.

Some insights into solving the aforementioned problems would be to acquire Rosetta licensing and acquire molecular folding state information from there. Although we would require licensing, this would be a good alternative and may allow us to circumvent the MAWS libraries that are no longer available such as simtk. Simtk was used for entropy calculations and is responsible for understanding torsion angles of the bonds of the protein-aptamer complex. Outside of simtk, we found that these bond deformations could likely be attainable through something like Rosetta, and there may be less scripting required to achieve the same functions, being that Rosetta is built for applications such as RNA folding calculations. The folding of the aptamer RNA is what would give aptamers their structure and interaction properties.

At the same time, we were extracting and thinking through what the MAWS process was based on their website and available code. Using the pieces which we believed to be the most relevant and useful, we came up with our own implementation of what MAWS' original goal was. Our process is detailed below. The steps in red are where we encountered a barrier when it came to implementing code.

Algorithm Plan - What we would have done without the barriers we are facing

1. **Import pdb file**

```
traj = pt.iterload("test.pdb")
```

2. **Getting the energy of the entire complex**

```
data = pt.energy_decomposition(traj, igb=8)
```

3. **We create a box which bounds the physical volume of the final protein-aptamer complex.**
    - This gives us set limits in the x, y, and z dimensions that locations (which are randomly generated in the following step) must lie within. We took this from the MAWS code.

```
# accessing the atom coordinates
xyz_raw = []

for i in traj[0]:
    xyz_raw.append(i)

# generating the box
longest_distance = max(power(sum_a(power2(list(xyz_raw), 2)), 0.5)) + 10
box_x = max([abs(xyz_raw[i][0]) for i in range(len(xyz_raw))]) + 10
box_y = max([abs(xyz_raw[i][1]) for i in range(len(xyz_raw))]) + 10
box_z = max([abs(xyz_raw[i][2]) for i in range(len(xyz_raw))]) + 10
```

4. **Generating X random locations (xyz) for the box**
    - These random locations are potential starting points for the aptamer and will be tested for their viability

```
# generating X random locations on the box
x_loc = 5
randX = np.random.uniform(low=0.0, high=box_x, size=(x_loc,))
randY = np.random.uniform(low=0.0, high=box_y, size=(x_loc,))
randZ = np.random.uniform(low=0.0, high=box_z, size=(x_loc,))
```

5. **Generate a X arrays each with 1 new location added to each of them**
    - This is done so that nucleotides can be added to the random locations. Each array contains the locations of the original atoms in the protein and the location where a nucleotide residue will be added (and where additional nucleotides will be concatenated) and binding affinity information will be computed.

```
# adding the random locations to their respective arrays
t1_arr = np.array([[[randX[0], randY[0], randZ[0]]]])
t1_xyz = traj.xyz
t1_xyz = np.column_stack((t1_xyz, t1_arr))

# same for t2
t2_arr = np.array([[[randX[1], randY[1], randZ[1]]]])
t2_xyz = traj.xyz
t2_xyz = np.column_stack((t2_xyz, t2_arr))

# t3
t3_arr = np.array([[[randX[2], randY[2], randZ[2]]]])
t3_xyz = traj.xyz
t3_xyz = np.column_stack((t3_xyz, t3_arr))

# t4
t4_arr = np.array([[[randX[3], randY[3], randZ[3]]]])
t4_xyz = traj.xyz
t4_xyz = np.column_stack((t4_xyz, t4_arr))

# t5
t5_arr = np.array([[[randX[4], randY[4], randZ[4]]]])
t5_xyz = traj.xyz
t5_xyz = np.column_stack((t5_xyz, t5_arr))
```

6. **We test each location in the arrays and implement the Metropolis Monte Carlo Algorithm (an initial sampling step to determine a set list of possible locations for the nucleotide sequence to begin, and the corresponding nucleotide and orientation of the nucleotide for each location)**
    a. Use the Metropolis Monte Carlo algorithm so that anything with negative free energy is automatically saved, and positive energies are saved with a probability $e^{-\Delta G/RT}$. Nucleotide-protein complex conformations that result in negative free energy are favorable and therefore potential starting points for an aptamer design (as additional nucleotides would be appended to the first). Conformations that result in positive free energy are unfavorable and would not occur in nature, but could potentially become favorable as additional nucleotides are appended to the sequence. Therefore, some positive states are saved. To decide which should be saved, a random number ζ is generated between 0 and 1 for each conformation resulting in positive free energy. If ζ is less than or equal to the probability of that conformation, the nucleotide position and energy is saved. If ζ is greater than the probability of that conformation, the information is discarded.

b. We test the 4 different nucleotides at each location for each array
    1. We test different angles for each nucleotide for each array
        a. Calculate the energy for each <array-location-nucleotide-angle> and save that energy according to the Metropolis Monte Carlo algorithm
        b. For saved energy conformations, calculate the conformational probability as $p = e^{-\Delta G/RT}/z$, where z is the molecular partition function (note that conformation probability is different from the probability factor used in part (a) to determine whether a value is saved). Conformation probability is used to calculate an entropy-like fitness function (adopted from MAWS). This is used as an alternative to the actual entropy that can be calculated directly from the partition function because it is more computationally efficient
    2. After completing this step, we have a set number of arrays (negative free energy arrays and the selected positive free energy arrays) to add more nucleotides to.
        a. Each array contains information about nucleotide xyz coordinates, which nucleotide (ATCG) has been chosen, energy of the complex, and entropy of the complex

c. The Metropolis Monte Carlo algorithm is an ideal choice for optimization here because the weight of $e^{-\Delta G/RT}$ will be very small and have a negligible probability at many of the random points. It would be extremely inefficient to run calculations for every random point generated within the box. Therefore, instead of randomly sampling and then weighing by a factor of $e^{-\Delta G/RT}$, we will choose the conformations with a probability factor of $e^{-\Delta G/RT}$ and only sample the chosen conformations. The python module pyMCMC could also be implemented in this step to aid in calculations

7. Append potential nucleotides by testing each location in the arrays and repeating Monte Carlo algorithm again

a. A nucleotide is appended to the previous one at each location
    i. Each of the 4 different nucleotides must be tested
    ii. Potential orientations of each nucleotide must be tested (rotate the nucleotide a set number of times by changing the angles phi and theta incrementally from 0 to 360 degrees)
b. For each <array-location-nucleotide-angle>, the free energy of the entire complex (protein and nucleotide sequence) is calculated.
c. As in step 6, values are saved according to the Metropolis Monte Carlo algorithm
    i. Negative free energy conformations are saved; positive free energy conformations are saved with probability $e^{-\Delta G/RT}$

d. Saved conformations are sampled to calculate conformation probability and the entropy-like fitness function
e. Each additional saved conformation marks an additional nucleotide appended to a potential sequence, and has associated entropy and energy values.

8. **Step 7 is repeated until there are 40 nucleotides in each sequence**
9. **Loop through all the energies generated and find the lowest**
    a. Sequences are ranked based on their entropy value
        i. For each sequence (<array-location-nucleotides-angles>) of 40 nucleotides, entropy values of sub-sequences are compared
            1. A typical aptamer consists of between 10 and 40 nucleotides, and it is not possible to know beforehand how many nucleotides will be present in the ideal aptamer
            2. Therefore, entropy values from previous iterations (iterations 10-40) are compared, and the information about the iteration with the minimum entropy is saved as a separate variable
                a. Example: if a final nucleotide sequence starting from a given position is: ATTAGCAGCTAGCCTGGATTTGACCCTTAGGGTACGATCG, the entropy of the conformations with the first 10, 11, 12, … 40 nucleotides in the sequence are compared. If the entropy is minimized when there are 15 nucleotides in the sequence, ATTAGCAGCTAGCCT is saved as a potential aptamer. This process is repeated for all potential aptamer starting locations.
            3. There is now one possible aptamer for each starting location. The entropies of these are then compared and the nucleotide sequence with the minimum entropy when bound to the protein is chosen as the aptamer.