Protein Folding With Cubes

James Gillespie

Idea

Previously, I had developed an algorithm based on the 1996 paper on simplified protein folding. The idea was simple: represent the position of acids in a 2D grid, and compute the energy level of a given configuration by looking at pairwise adjacency's of acids. The pairwise energy was determined by simple rules that dictated the energy between hydrophobic-hydrophobic, hydrophobic-hydrophilic, and hydrophilic-hydrophilic pairs. I also implemented a simple "Water Penalty", which penalized a hydrophobic-water contact by increasing the energy by a set amount.

During our presentation, a classmate asked why I chose to make the grid 2D. I responded that 3D would be computationally infeasible, or at least, very time consuming. However, I later recalled that the paper did indeed have a 3D version of the model; and if they could do it in 1996, I certainly could do it in 2023. Furthermore, I believed I could do it in an efficient manner, if only I implemented it carefully.

Towards an Efficient Algorithm

Throughout the algorithm, we often need to get positions adjacent to a given position - both when computing energy, and when looking for where to place the next acid. Rather than computing these positions each time they are needed, we can pre-compute them, and store them in a 3D data structure. This allows adjacencies to be obtained in O(1) time - to find the adjacencies of some position (x, y, z), just get the list at position (x, y, z) in the database.

I also decided that I should be able to go further than the 1996 algorithm, and incorporate diagonal adjacencies. This calls for a classification of adjancencies to a position $P = (x, y, z)$:

- Directly Adjacent: 1 coordinate differs by ± 1 (distance = 1)
- C1 Adjacent: 2 coordinates differ by ± 1 (distance = $\sqrt{2}$)

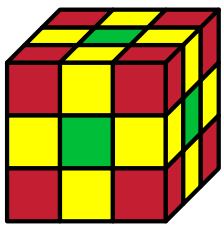

Figure 1: Adjacencies

Figure 2: Edge Types 1 and 2

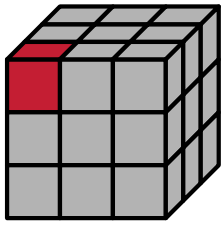

Figure 3: Edge Type 3

- C2 Adjacent: 3 coordinates differ by ± 1 (distance = $\sqrt{3}$)

In Figure 1, Green, Yellow, and Red represent Direct, C1, and C2 adjanencies, respectively. The energy between two acids is scaled by the inverse of the square of the distance - so Direct, C1, and C2 adjanencies are scaled by 1, 1/2, and 1/3, respectively.

A significant complication to the algorithm is considering water molecules "outside" of the grid, whose positions are not listed in the 3D adjacency data structure for obvious reasons. Thus, we introduce a secondary 3D data structure, with *precomputed* penalty multipliers based on position type. We classify position types (referred to as "edge types" in the algorithm) as follows.

- 0 - Position is total enclosed within the grid
- 1 - Position is exposed on one face
- 2 - Position is exposed on two faces
- 3 - Position is exposed on 3 faces

Figures 2 and 3 show the different edge types on a 3x3 grid. A position of Edge Type 0 would be entirely enclosed.

Our new 3D structure contains the edge type for each position. A second structure contains pre-computed values for the water penalty caused by molecules outside of the grid based on edge type. We can thus access these values in O(1) time as well.

Performance and Results

The algorithm has the capability to consider or ignore C1 diagonals, C2 diagonals, and the Water Penalty. Code is written in C and compiled using -O2 option in gcc.

Mode 1 - All Options Off

C1 = OFF, C2 = OFF, Water Penalty = OFF. Time to run: 53 seconds.
Results:

```
Num Recursive Calls: 440459565
Valid configurations tested: 142296000
Num new mins: 18
Min energy: -40.000000
Optimal configuration:
Slice: z = 0
SER  GLN  NULL
PRO  TRP  ASP
PRO  LEU  LYS
Slice: z = 1
ASN  ILE  NULL
LEU  TYR  GLY
PRO  PRO  GLY
Slice: z = 2
NULL NULL NULL
GLY  SER  NULL
ARG  SER  NULL
```

Mode 2 - Include diagonals

C1 = ON, C2 = ON, Water Penalty = OFF. Time to run: 2 mins 54 seconds.
Results:

```
Num Recursive Calls: 440459565
Valid configurations tested: 142296000
Num new mins: 32
Min energy: -75.350000
Optimal configuration:
Slice: z = 0
NULL NULL NULL
NULL GLY  GLY
NULL LYS  ASP
Slice: z = 1
NULL PRO  SER
SER  PRO  TYR
SER  LEU  ILE
Slice: z = 2
NULL PRO  ASN
ARG  PRO  LEU
GLY  TRP  GLN
```

Mode 2 - All Options On

C1 = ON, C2 = ON, Water Penalty = ON, Water Penalty Amount: 2.5. Time to run: 4 mins 15 seconds.
Results:

```
Num Recursive Calls: 440459565
Valid configurations tested: 142296000
Num new mins: 34
Min energy: 68.783333
Optimal configuration:
Slice: z = 0
NULL NULL NULL
ASN  LEU  GLY
LYS  ASP  GLY
Slice: z = 1
NULL NULL SER
TRP  TYR  PRO
LEU  PRO  PRO
Slice: z = 2
NULL NULL SER
GLN  ILE  GLY
SER  PRO  ARG
```

Improving performance with Multithreading

Luckily, this problem lends itself very well to parallel processing. I tweaked the algorithm to spawn a thread for each starting position - for a 3x3 cube, we have 8 starting positions (a 2x2x2 cube of starting positions will obtain all possible structures and eliminate symmetrical structures generated by a 3x3x3 cube of starting positions - indeed, the 2x2x2 cube will still produce many symmetrical structures). Each thread keeps track of its own individual results; once all threads have finished, the results are combined.

A Correction

Previously, the model included energies between molecules that appear consecutively in the sequence. This has been rectified.