1 Problem Background

Note: In this paper, the terms ***amino acid*** *and* ***residue*** *are used interchangeably, ignoring biological nuances.*

Protein folding is the physical process by which a linear chain of amino acids ***(primary structure)*** spontaneously collapses into a functional three-dimensional conformation ***(tertiary structure)***. This folded structure determines the protein's biological activity and chemical interactions.

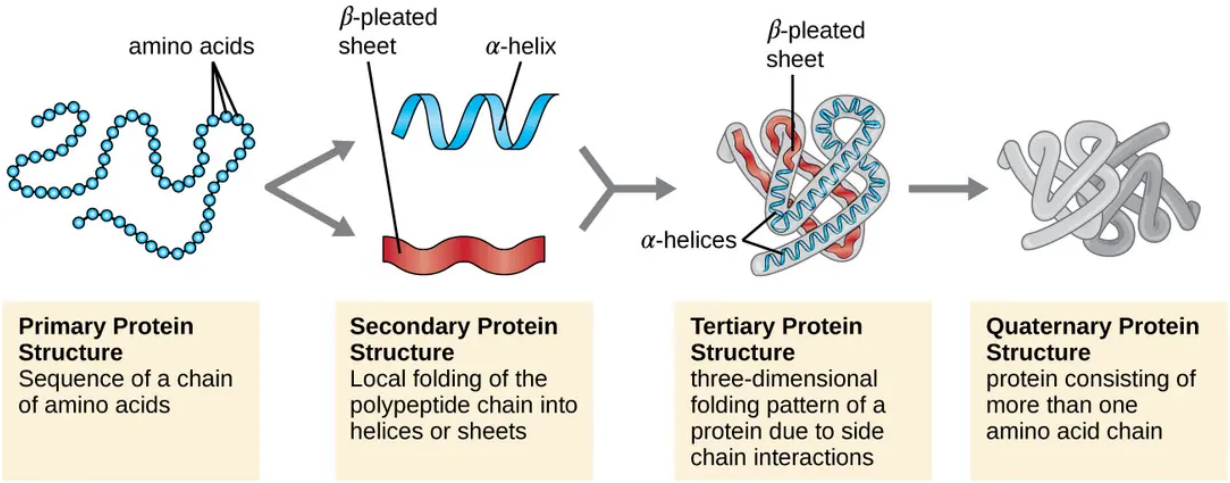


Figure 1: Protein Folding Process

Despite having the complete amino acid sequence, predicting the final folded structure remains an **NP-Hard** problem due to the astronomical number of possible configurations and the complex interplay of forces such as hydrogen bonding, hydrophobic interactions, and van der Waals forces. As a result, only around 200,000 protein structures have been experimentally solved, out of billions of possible sequences.

Accurate protein folding prediction is biologically critical because a protein's structure directly determines its function. Misfolding or inaccurate predictions can result in proteins with unintended chemical properties or loss of function, potentially contributing to disease.

1.1 Current Solutions: HP 3D Lattice model

Due to the complexity of the protein folding problem, researchers have proposed an approximate model known as the **HP 3D Lattice model**, a simplified formulation where:

- **Protein Structure**: 3D cubic lattice
- **Amino Acid Classification**: where the amino acid sequence (eg.`G,A,S,L`) is extracted and simplified into (eg.`P,H,P,H`), into 2 distinct types:
  - **Hydrophobic (H)**: A, C, F, I, L, M, V, W, Y
  - **Polar (P)**: D, E, H, K, N, Q, R, S, T, G, P

  Hence the name HP 3D Lattice Model.

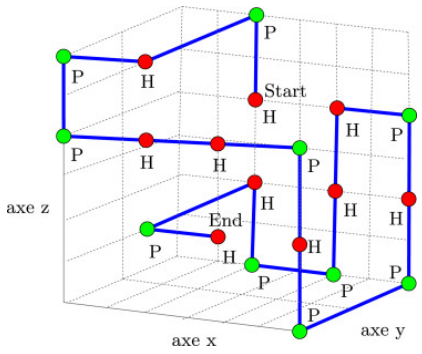


Figure 2: 3D HP lattice structure, taken from [1].

- **Energy Calculation**: Intramolecular energies are reduced to two types:
  - Non-sequential **H-H** contact adjacency will have interaction energy of -1
  - Non-sequential **P-P** or **H-P** contact adjacency will have neutral energy of 0

-1 energy value corresponds to a favourable interaction (i.e., negative enthalpy), implying that such contacts make the protein structure more stable and spontaneous.

Thus, optimisation & metaheuristic algorithms can be used for the HP lattice model, with objective function to **minimise the enthalpy energy** or **maximise the H-H contacts**, modelled with space constraints.

1.2 Bond Adjacency vs Contact Adjacency

Note: It is important to distinguish between:

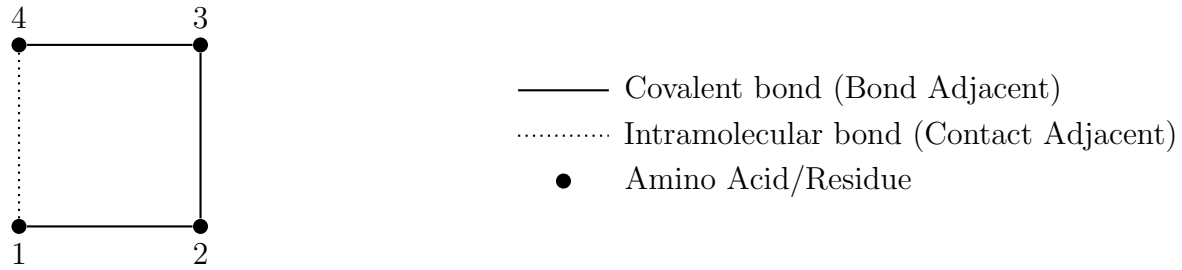


Figure 3: A Simple Folded 4 Chain Length Protein

- **Covalent bond adjacency**:*represented by solid line* — sequential residues in the protein chain that are connected via covalent peptide bonds (i.e., residues 1 & 2, 2 & 3, 3 & 4)
- **Non-sequential contact adjacency**:*represented by dotted line* — residues that are *not* sequentially connected but are positioned adjacent to one another on the lattice. (i.e., residue 1 & 4) These are the ones that we use to calculate energy with.

1.3 Our Project Goal

While the HP model offers simplicity and computational efficiency, it lacks biochemical realism. To address this, we propose an improvement by incorporating the **Miyazawa-Jernigan (MJ) matrix** [4], a 20×20 empirical matrix that captures residue-residue interaction energies. (Refer to Figure12)

Instead of assigning a uniform -1 energy to H-H contacts, the MJ matrix enables us to:

- Assign **explicit contact energies**, including both favourable (negative) and unfavourable (positive) interactions

Our overall objectives remain the same:

1. To **minimise the enthalpy** of the folded protein in order to predict its likely structure. *(However, we must temper our expectations, as this model still assumes a discrete 3D lattice, which does not fully reflect how proteins fold in continuous real-world space.)*
2. To demonstrate that **heuristics and metaheuristics** have significance in such specific biological environments

2 Protein Folding MILP Formulation using MJ Matrix

The MILP formulation that follows is inspired by and modified from Guo et al. [2], who formulated the HP model on a 3D lattice [2]. Our formulation generalises and refines their approach to incorporate the MJ matrix and account for all amino acid intramolecular bond energies.

2.1 Decision Variables

Note: Covalent BOND adjacency is different from non-sequential CONTACT adjacency

$$\begin{aligned} x_{i,\phi} &\in \{0,1\} \quad \forall i \in [n],\, \forall \phi \in [G]^3 \\ y_{i,j} &\in \{0,1\} \quad \forall i < j,\, |i-j| \geq 2 \end{aligned}$$

where:

- $x_{i,\phi} = 1$ if amino acid i is placed at lattice coordinate $\phi = (x, y, z)$, 0 otherwise
- $y_{i,j} = 1$ if residues i and j are in non-sequential CONTACT adjacency (adjacent in lattice but not consecutive in sequence), 0 otherwise
- $[n] = \{1, 2, \ldots, n\}$ is the protein sequence length
- $[G]^3 = \{1, 2, \ldots, G\}^3$ is the 3D cubic lattice space

2.2 Objective Function

$$\min \sum_{\substack{i<j \\ |i-j|\geq 2}} c_{i,j} y_{i,j}$$

Minimises total contact energy, where $c_{i,j}$ is the pairwise energy from the MJ matrix.

2.3 Constraints

2.3.1 1. Single Position Assignment

$$\sum_{\phi \in [G]^3} x_{i,\phi} = 1 \quad \forall i \in [n]$$

Each amino acid occupies exactly one lattice point.

2.3.2 2. No Positional Clashes

$$\sum_{i=1}^{n} x_{i,\phi} \leq 1 \quad \forall \phi \in [G]^3$$

At most one amino acid per lattice point.

2.3.3 3. Bond Adjacency (Chain Connectivity)

$$x_{i,\phi} \leq \sum_{\phi' \in \mathcal{N}(\phi)} x_{i+1,\phi'} \quad \forall i \in \{1, 2, \ldots, n-1\}, \forall \phi \in [G]^3$$

where $\mathcal{N}(\phi) = \{\phi' : \|\phi' - \phi\|_1 = 1\}$ is the set of positions adjacent to ϕ.
Ensures consecutive residues are lattice-adjacent (Manhattan distance = 1). Thus, this enforces **an important assumption**: **ALL bond length = 1 unit**

2.3.4 4. Contact Adjacency

For each $i < j$ with $|i - j| \geq 2$ and each $\phi \in [G]^3$:

$$y_{i,j} \geq x_{i,\phi} + x_{j,\phi'} - 1 \quad \forall \phi' \in \mathcal{N}(\phi)$$

$$y_{i,j} \leq \sum_{\phi' \in \mathcal{N}(\phi)} x_{j,\phi'} + (1 - x_{i,\phi}) \quad \forall \phi \in [G]^3$$

These constraints ensure $y_{i,j} = 1$ iff residues i and j are:

1. **Non-consecutive** in sequence ($|i - j| \geq 2$, ie. not covalent bond adjacency)
2. Adjacent in lattice ($\|\phi_i - \phi_j\|_1 = 1$, ie. side by side)

More intuitively, contact adjacency, $y_{i,j} = 1$, is when amino acids are **not** covalently bonded but side by side in coordinates.

2.4 Energy Parameters

$$c_{i,j} = \text{MJ}(a_i, a_j) \quad \text{(from Miyazawa-Jernigan matrix)}$$

where a_i is the amino acid type at sequence position i. (Refer to Figure 12)

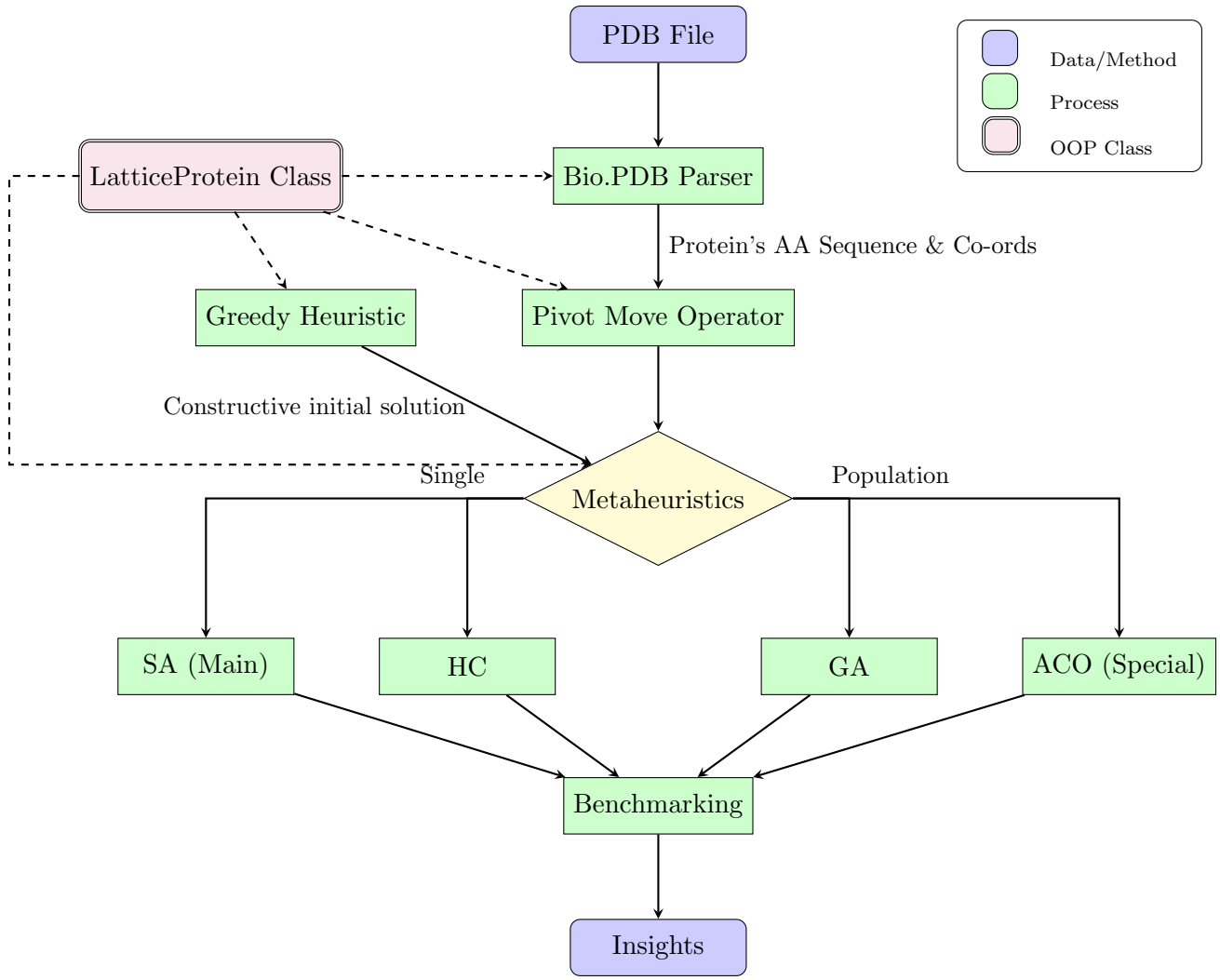
2.5 Search Space

The theoretical upper bound of 5^n possible configurations arises from:

- A *self-avoiding walk* constraint on the lattice, where no two amino acid should occupy the same coordinate.
- At each step (amino acid), 5 possible moves:
  - 6 cardinal directions in 3D cubic lattice
  - -1 forbidden move (backtracking to previous residue)

This yields approximately $4.684^n \leq 5^n$ distinct n-residue conformations, since each step in a self-avoiding walk on a cubic lattice has at most 5 non-reversing choices. Mathematically, the number of such walks c_n grows as $c_n \approx \mu^n$, where $\mu \approx 4.684$ is the *connective constant* for the simple cubic lattice [3].

3 Methodology



This is a general flow of our methodology. The later sections shall expand further on what each node means and does. Click on each to go to the specific (sub)sections.

4 Description of Data and Preprocessing Steps

Note: The full implementation with detailed line-by-line comments + markdowns is available in the Jupyter notebook (`.ipynb` file). The pseudocodes shown below is simplified.

4.1 MJ matrix data

The MJ matrix was digitised from the original paper [4] (Appendix 12) using AI and stored as a Python Pandas DataFrame for ease of referencing.

4.2 Protein Data (Bio.PDB Parser)

Our protein structural data are obtained from the Protein Data Bank (PDB) archive. We are only focusing on two main attributes of the `.pdb` file:

- **SEQRES:** the primary **amino-acid sequence** of each chain.
- **ATOM:** the three-dimensional **Cartesian coordinates** of each atom in the structure. (Only used for actual protein comparison in conclusion)

To parse these, we use the `BioPython` library & `Bio` packages:

```
from Bio.PDB import PDBParser
```

For this project, we load the protein with file `2a3d.pdb`, which is a monomeric protein with a well-defined hydrophobic core. (More about it in 6.2)We extract:

- `original_sequence`: the full amino-acid sequence from **SEQRES**
- `hp_sequence`: sequence indicating hydrophobic (H) vs. polar (P) residues from `original_sequence`

Algorithm 1 Extract Hydrophobic/Polar (H/P) Sequence from PDB File

procedure GET_HP(PDB file)
    Use `PDBParser()` to load the sequence structure from the file
    Traverse all residues in the sequence
    for each standard amino acid **do**
        record its three-letter code
    Convert the original sequence to an H/P sequence using `aa_to_hp`
    Return both the H/P sequence and the original sequence

Note: `aa_to_hp` *is a function that simply maps each amino acid to an H or P using hashmaps or list indexing, just a leetcode easy function*

4.3 Protein Preprocessing (LatticeProtein Class)

Because we repeatedly operate on the same protein data (sequence, coordinates, HP sequence, energy level, ...) in a 3D lattice stucture, we encapsulate these operations in a Python **Class**:

Algorithm 2 LatticeProtein Class

```
procedure LatticeProtein.__init__(HP sequence, original sequence)
    Attribute 1: HP sequence
    Attribute 2: original sequence
    Attribute 3: length of sequence
    Attribute 4: list of tuples storing Cartesian coordinates of each amino acid
    Attribute 5: enthalpy energy of the protein
function initial_soln                                              ▷ Greedy Heuristics
    Start positions list with first amino acid at origin
    for each subsequent amino acid in sequence do
        Generate all 5 candidate positions adjacent to current amino acid
        for each candidate position do
            Calculate energy gain from contact adjacent residues   ▷ uses calculate_energy()
            if energy gain is better (lower) than best_energy then
                Update best_position and best_energy
        if no valid candidate found then
            Choose any adjacent free position to previous amino acid
        Append best_position to positions list
    return positions list
function calculate_energy
    Initialise energy to 0
    for each pair of residues that are non-consecutive in the sequence do
        if they are contact adjacent on the lattice then           ▷ uses is_adj()
            Refer to MJ Matrix and add their energy value
    return total energy
function is_adj(coord1, coord2)
    Manhattan distance between coord1 and coord2 == 1
```

`LatticeProtein` has 5 attributes and 3 methods:

- **Attribute 1:** HP sequence
- **Attribute 2:** Original amino acid sequence
- **Attribute 3:** Length of the sequence
- **Attribute 4:** List of tuples storing Cartesian coordinates of each amino acid
- **Attribute 5:** Enthalpy energy of the protein

The class also includes the following methods:

- **Method 1:** `initial_soln()` — generates a greedy initial positions, where amino acids are laid out sequentially. For each amino acid, the algorithm explores the best position out of 5 moves based on energy evaluation. This continues for n amino acid. This provides a decent **constructive heuristics** to start our metaheuristics. (Refer to Figure 13 for simplified flow chart)
- **Method 2:** `calculate_energy()` — calculates the enthalpy energy based on residue contacts using the MJ matrix

- **Method 3:** `is_adj()` — checks if two coordinates are adjacent on the lattice

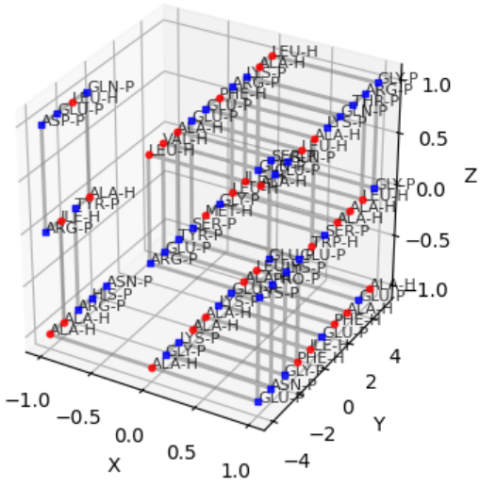
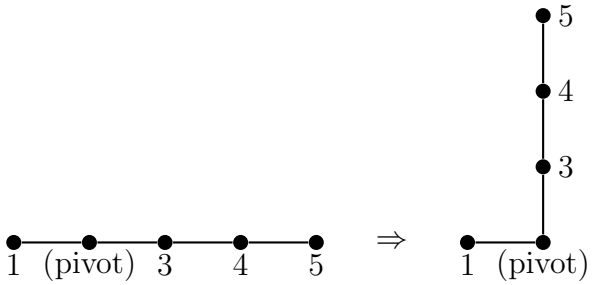


Figure 4: Greedy Heuristics of Protein 2a3d

4.4 Pivot Move Operator

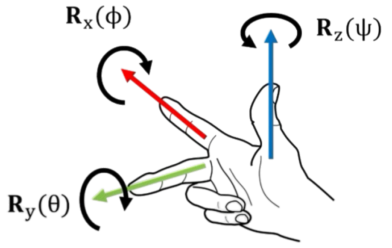
• **Concept 1: Pivot Chain Preservation**

The pivot move applies a rotation to a structure ***starting from*** a chosen pivot residue, rotating ***all subsequent amino acids***. Why must we rotate all subsequent amino acid?



Let's say we choose amino acid 2 to be the pivot. If we rotate 90°, the subsequent amino acids, 3 to the end, must follow the same rotation. This is to make sure the bond length is 1 & the chain connectivity is preserved.

• **Concept 2: Neighbourhood Size when Pivoting on One Residue, 9 or 5?**



The neighbourhood is where the subesquent amino acids turn 90°, 180°, 270° (360° is left out for obvious reasons) around the **x, y, z** axes, generating ***9*** **fundamental perturbations for each chosen pivoting amino acid**. 3 choices × 3 choices = 9. This is inspired by Game Developers where they model continuous 3D space, but in our case, we use a discrete lattice coordinate space.

Figure 5: The 9 Theoretical Pivot Moves.

However, as discussed in Section 2.5, each amino acid has ***5*** possible theoretical next move. This was even used in the Greedy Heuristics in the LatticeProtein code above. So why do we consider a neighbourhood size of ***9*** instead of ***5***?

This relates back to **Concept 1**, where we rotate *all subsequent amino acids* starting from a selected pivot. In the initial Greedy Heuristic, each amino acid only considers its 5 possible adjacent lattice positions in isolation. However, in the case of a pivot move, we are *not simply placing a single amino acid*; we are *rotating the entire tail of the protein* about a fixed pivot.

As a result, the neighbourhood is not defined by 5 local directions, but by the 9 possible global rotations. Each of these rotations generates a different global positioning of the chain.

It would therefore be short-sighted to only consider the movement of the next residue, as a **pivot move changes the geometry of all residues downstream**. Exploring all 9 possible rotations allows for a more comprehensive and accurate exploration of the protein's search space.

Algorithm 3 Pivot Move Algorithm for Protein Folding

```
1:  procedure PivotMove(protein, systematic, max_attempts)
2:      Store original energy and original positions of protein
3:      Store best_energy as original energy and best_positions as None
4:      9 possible moves: rotate around x/y/z axes by 90°, 180°, or 270°        ▷ Concept 2
5:      if systematic = True then                          ▷ Systematic approach (Concept 4)
6:          attempts ← 0
7:          found_improvement ← False
8:          while attempts < max_attempts and not found_improvement do
9:              Choose random pivot point along protein chain (not at ends)
10:             for each move in 9 possible moves do
11:                 attempts ← attempts + 1
12:                 if attempts ≥ max_attempts then                              ▷ Concept 4
13:                     break
14:                 Reset protein to original positions
15:                 Rotate everything after pivot point                          ▷ Concept 1
16:                 if IsValid(protein) then                                     ▷ Concept 3
17:                     new_energy ← CurrentEnergy(protein)
18:                     if new_energy < original_energy then
19:                         found_improvement ← True
20:                         break
21:         if found_improvement then
22:             return new_positions
23:         else
24:             Try random sampling instead
25:     else                                       ▷ Random sampling approach (Concept 4)
26:         for attempt ← 1 to max_attempts do
27:             Choose random pivot point
28:             Choose random axis and rotation angle
29:             Rotate everything after pivot point                              ▷ Concept 1
30:             if IsValid(protein) then                                         ▷ Concept 3
31:                 Accept move regardless of energy
32:                 return new_positions
33:         return original positions
```

***Note**: a simplified flow chart at 14 to understand what the pivot move pseudocode is doing*

• **Concept 3: Solution Validity Conditions**
A solution is valid if and only if both are satisfied:

1. **Non-overlapping positions**:
   - For an n-length chain, all amino acids must have n unique Cartesian coordinates
   - Mathematically: $\forall i, j \in \{1, ..., n\}, i \neq j \Rightarrow (x_i, y_i, z_i) \neq (x_j, y_j, z_j)$
2. **Proper chain connectivity**:
   - Consecutive amino acids must maintain covalent bonds with unit length
   - Manhattan distance between adjacent residues must equal 1, as mentioned in constraints
   - Mathematically: $\forall i \in \{1, ..., n-1\}, |x_{i+1} - x_i| + |y_{i+1} - y_i| + |z_{i+1} - z_i| = 1$

• **Concept 4: Systematic & Random Sampling Approach** (refer to 14)
Initially, we only used the **Random Approach**, functioning as a typical search/move operator that *randomly selects* a pivot residue, an axis, and a rotation angle.

After consulting with our professor, we introduced a more deliberate **Systematic approach**, inspired by best descent. This method fixes a random pivot point and *exhaustively explores all 9 possible rotations*, returning the valid configuration with the lowest energy.
However, due to the highly compact nature of the initial solution, many pivot moves were either invalid or resulted in worse energy. To overcome this, we introduced a `while` loop to ensure that only moves which are both valid and improve energy are accepted, but this came at the cost of significantly slower runtime per pivot move.
Key Differences:

Difference	**Systematic**	**Random**
Exhaustiveness	Tests all 9 rotations	Tests 1 random rotation
Acceptance	Energy-improving only	Any valid move
Runtime	Slower	Faster

Therefore, we decided to combine both approaches, similar to how a **Variable Neighbourhood Search** (*VNS*) would work:

1. Attempts **Systematic Approach** (capped at `max_attempts`)
2. Falls back to **Random Approach** if no improvement found (capped at `max_attempts`)
3. If both approaches fail to return valid solution, return the original positions.
4. Behaves like *simplified VNS* with two possible neighbourhoods

This algorithm avoids randomly selecting possible neighbourhoods and also prevents sacrificing excessive time searching for better energy within a neighbourhood.

Disclaimer: This `pivot_move` *operator is unconventional, as it does more than just generate a neighbouring solution like a typical move operator. We are experimenting with integrating best descent logic directly into the move operator to see if it improves performance.* ***A standard version of the move operator is also provided in another cell*** *(Random Sampling approach), so we can easily switch back to the conventional behaviour if needed.*

5 Metaheuristic Methods

5.1 Simulated Annealing(Main)

Algorithm 4 Simulated Annealing for Protein Folding

procedure SIMULATED_ANNEALING(hp_sequence, original_seq, max_iter, initial_temp)
    Initialise `LatticeProtein` from `hp_sequence`, `original_seq`
    Set `current_protein`, `best_protein`, and `best_energy` from the initial protein
    Initialise `energy_history`, `position_history`
    Set `temperature` ← `initial_temp`
    Initialise `temperature_iteration_counter` ← 0
    Initialise `iteration` ← 0
    while within time limit **do**
        Create `trial_protein` as a deep copy of `current_protein`
        Apply `pivot_move()` to `trial_protein`
        Compute `trial_energy` ← energy of `trial_protein`
        if `iteration` ≤ 1000 **then**
            if `temperature_iteration_counter` > 25 **then** ▷ **Concept 1**
                Update `temperature` using log cooling:
                $T \leftarrow \frac{\text{initial\_temp}}{\log(\text{iteration} \times 5000)}$
                Reset `temperature_iteration_counter` ← 0
        else
            Update `temperature` using exponential cooling:
            $T \leftarrow \text{temperature} \times 0.9975^{\text{iteration}}$
        Increment `temperature_iteration_counter`
        Increment `iteration`
        Compute $\Delta E \leftarrow$ `trial_energy` − `current_energy`
        Compute Metropolis probability: $P \leftarrow e^{-\Delta E/T}$
        if `trial_energy` < `current_energy` **OR**
            `random()` $< P$ **AND** $\Delta E < 30$) **then** ▷ **Concept 2**
            Accept move: `current_protein` ← `trial_protein`
            if `trial_energy` < `best_energy` **then**
                Update `best_protein`, `best_energy`
        else
            Reject move
        Append current energy to `energy_history`
        Append current positions to `position_history`
    return `best_protein`, `energy_history`, `position_history`

• **Concept 1: Cooling Rate**
We explore multiple cooling rates: linear, rational, exp and polynomial decay. Through trial and error, we found out that the algorithm tends to **perform best** when the Metropolis acceptance probability remains in the range of `0.3 - 0.6`. Thus, we employed a 2 phase cooling method.

- **Phase 1 (Exploration)**: First 1000 iterations

– **Logarithmic cooling** every 25 iterations:

$$T = \frac{T_0}{\log(\text{iter} \times 5000)}$$

– Preserves higher acceptance probabilities longer to escape local minima

- **Phase 2 (Exploitation)**: Subsequent iterations
  – **Exponential cooling** for after 1000 iterations:

$$T = T_0 \times (0.9975)^{\text{iter}}$$

  – Starts converging faster and tapers off the probability

Note: the numbers 0.3,0.6,1000 iterations etc. are all emprically found, specific to this 2a3d protein. Optimisation of these parameters will be different for different proteins

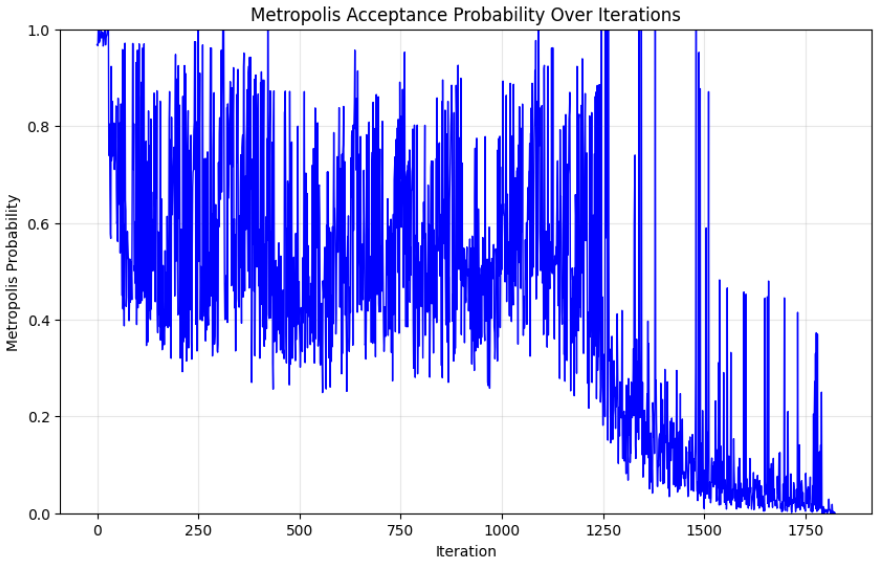


Figure 6: Metro Probability over Iterations with our parameters

• **Concept 2: Acceptance Condition**
Normal metropolis probability in Simulated Annealing occasionally accepted *extremely poor* solutions, which led to convergence in extremely suboptimal neighbourhood when temperature cooled. To avoid this, we introduced a new condition: $\Delta E \leq 30$ (empirically found, different for different proteins). Therefore, the new condition is:

$$\text{Accept if: } (E_{\text{new}} < E_{\text{current}}) \vee \left[\Delta E < 30 \wedge (P > \text{rand}(0,1)) \right]$$

This means if the metropolis accepts a worse move, it does so only **if the increase in energy is below a threshold of 30**. This threshold prevents the search from exploring neighbourhoods that lead to extremely poor solutions.

5.2 Genetic Algorithm

Genetic Algorithm is very fitting for this project, as we are literally simulating nature with algorithm to predict proteins in real life.

Algorithm 5 Genetic Algorithm for Lattice Protein Folding

```
procedure GENETIC_ALGORITHM(hp_sequence, original_seq, max_iter)
    Initialise population ← empty list
    Initialise best_energy ← ∞, best_solution ← None
    Initialise timer and iteration ← 0
    for i = 0 to 9 do                                   ▷ Generate initial population
        Initialise LatticeProtein from hp_sequence, original_seq
        Apply 10-30 random pivot_move() operations
        Add protein to population
    Add greedy solution to population                   ▷ Population size = 10
    while within time limit do
        Evaluate fitness for all in population
        Select top 40% as elites                        ▷ Elitism
        Select parents using roulette wheel selection
        Initialise offspring ← empty list
        for each parent pair do
            if random() < 0.95 then                     ▷ Crossover
                Produce child via crossover             ▷ Concept 1
            else
                Select fitter parent as child
            if random() < 0.75 then                     ▷ Mutation
                Apply pivot_move() mutation             ▷ Concept 2
            Add child to offspring
        Replace worse offspring with elites
        Update population ← offspring
        if current best energy < best_energy then
            Update best_energy, best_solution
        Increment iteration
    return best_solution, energy history, solution history
```

Most of GA does not differ that much from what we learnt in class. Only 2 new concepts introduced:

• **Concept 1: Crossover Operator**

Instead of using `pivot_move()` as our primary move operator, we use Single Point Crossover. However, due to complex spatial constraints of protein folding, many times, the offspring are **invalid**. We therefore attempt up to **50 crossovers**, and if all fail, we return the **better of the two parents** instead.

• **Concept 2: Mutation**

Mutation is implemented as a **random** `pivot_move()`. Similar to crossover, mutations often lead to invalid structures.

– We attempt up to **10 mutations**, and if all fail, we revert to the original.

– The number of pivot moves applied is randomly chosen from **1 to 3**, allowing for both small and slightly larger structural changes in each mutation.

5.3 Hill Climbing

This is a best descent algorithm, serving primarily as a **baseline for benchmarking**, to evaluate how well more advanced metaheuristics perform compared to a basic local search.

Due to simplicity of code, the pseudocode shall be in the appendix, Algorithm 7

*Important Note: This is **not a true best-descent hill climber**, because our `pivot_move()` function includes both a systematic and random strategy — effectively acting like a simplified **Variable Neighbourhood Search (VNS)**. This hybrid move operator allows us to briefly explore **2 neighbourhood structures**, not just the immediate best move, thus the results of this HC should be better than normal HC.*

5.4 Ant Colony

Out of academic academic curiosity, this algorithm is special compared to others. This algorithm does not use the pivot operator which the other algorithms use, but uses its own pivot operator where it decides the path to take using the heuristic value in the equation below in Concept 1. Therefore it is the special in the fact that it does not find the optimal neighbour through `pivot_move`, but goes through every neighbouring solution, making it much more exploratory than the other algorithms.

With this algorithm, we implemented it in a way that maximises the H-H contact while minimising the enthalpy energy at the same time. We wanted to experiment with this algorithm to see if forming the hydrophobic core adds benefits and increase the compactness of the protein formed, which is generally reflective of nature.

Algorithm 6 Ant Colony Algorithm for Protein Folding

```
procedure AntColonyOptimisation(hp_sequence, original_seq, max_iter, num_ants, alpha, beta, evaporation_rate, initial_pheromone, time_limit)
    Set the possible directions for pheromone distribution
    Initialize pheromone level in each direction to 1
    procedure CalculateEnhancedHeuristic(current_pos, direction, protein_state, step)
        Encourage H-H contacts and compactness (calculating distance from other amino acids)  ▷ Concept 1
        Penalize isolated or linear configurations
        return heuristic score
    procedure ImprovedProbabilisticSelection(protein_state, step)
        Calculate probability using pheromone and heuristic value
        Select direction to fold using the calculated probability
        return selected direction
    procedure ConstructSolutionWithBacktracking
        Let the ant backtrack when energy stagnates to explore better folding options  ▷ Concept 2
    procedure ConstructMultipleAttempts
        for each attempt do
            Build solution using ConstructSolutionWithBacktracking  ▷ Concept 3
            Update the solution if improved
            if stagnation detected then
                Reset pheromones
    return best protein structure, position history, energy history
```

- **Concept 1: Heuristic Value**

$$p_{i,j} = \frac{(\tau_{i,j})^{\alpha} * (\eta_{i,j})^{\beta}}{\sum_{\forall k} (\tau_{i,k})^{\alpha} * (\eta_{i,k})}$$

The algorithm directly calculates the probabilitic value in the `ImprovedProbabiliticSelection`. However the heuristic value in our algorithm replaces the distance value in the original Ant Colony probability equation. We tweaked this to encourage H-H contacts and compactness (calculating distance and energy from other amino acids by rewarding shorter distances.

• **Concept 2: Backtracking Ants**
Typically, in the original provided algorithm, there is no backtracking of the ants and they might increase bias for good initial solutions. With this, the bias sometimes makes it a greedy algorithm which only obtains the local minimum solution. With the backtracking of ants, it allows the ants to explore less optimal neighbourhoods, in order to maybe find a more optimal protein fold with lesser enthalpy energy.

• **Concept 3: Construct Multiple Attempts** Combined with concept 2, the construction of multiple solutions using the ants allows the algorithm to have more chances to explore less optimal neighbourhoods, allowing the Ant Colony Algorithm to be more exploratory and less exploitative. This would be good for our use case as we are trying to predict or find protein structures, which is an explorative venture.

6 Observations & Insights

6.1 Performance Benchmarking: Metaheuristics vs. Exact Solvers

To evaluate the effectiveness of our metaheuristic approaches, we conducted a comparative performance analysis against an exact solver (Gurobi). We will plot the best energy progression over time.

Experimental Setup:

- Tested on a small, 9-amino acid chain
- All methods have 60s time limit.
- Compared the four metaheuristics against Gurobi's current best solution. (Actual solution takes hours)

The results are shown below:
Note: We realise there is 1 less line than expected, that's because GA and HC interestingly had the same energy achieved over time, and the lines are set to 0.7 transparency, thus creating a brown line.

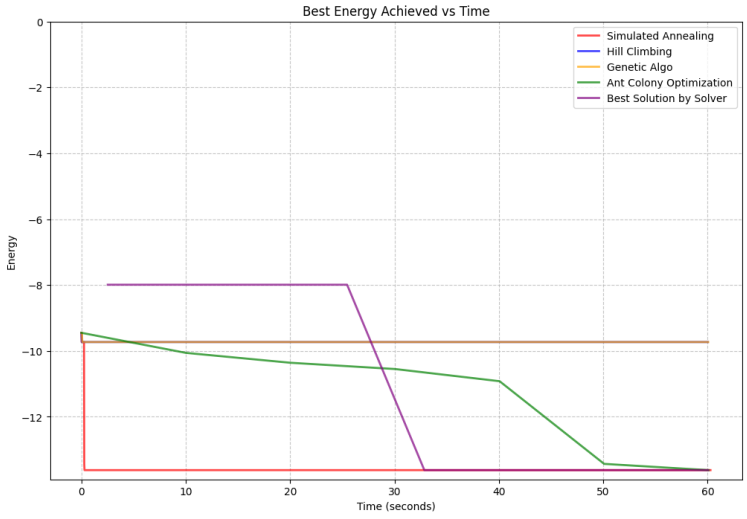


Figure 7: Energy over Time Graph on 9 Chain Protein

Initial Performance Analysis:

- Our **Constructive Greedy Heuristic** provided a strong starting point (-9.45), outperforming Gurobi's initial solution (-7.99), giving all metaheuristics a computational head start
- Gurobi solver took about 33s before arriving at best solution so far (-13.62)
- HC and GA showed minimal improvement (-9.45 to -9.73), and the identical output (resulting in the brown overlap) suggests both methods became trapped in similar local optima
- SA has the most potential, immediately solving the solution that the Gurobi provided
- ACO proves to be decreasing in enthalpy, even though we are evaluating it via H-H bond or HP-lattice model, meaning that our model works

6.2 Performance Benchmarking: `2a3d` Protein

The `2a3d` protein was selected as our primary test case:

- **Chain Length**: 75 amino acids, providing balance between computational complexity and biological relevance
- **Monometric Nature**: As a single-chain protein, folding simulation is applicable to it
- **Well-researched**: The `2a3d` structure has been extensively studied, with many available experimental data.

Experimental Setup:

- Similar to the previous graph, we test on `2a3d` protein
- All methods have 480s time limit
- Lowest Convergence and Rate of Convergence will be taken into account
- Set `seed(21)` (for some reason I can never get the same output even after fixing seed)

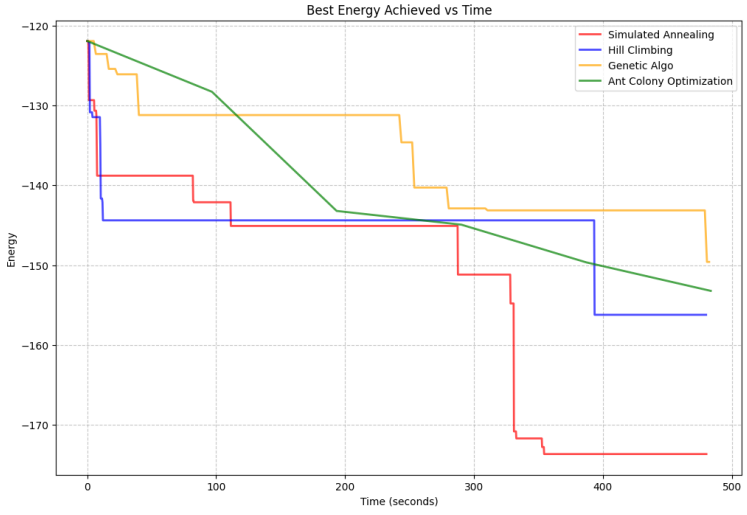


Figure 8: Best Energy over Time for Protein `2a3d` when seed(21)

Performance Analysis:

- SA reached the lowest convergence by far (-173.69)
- HC has the fastest rate and initial dip was big at the start
- GA is the worst performer, ironically (-149.59)

- ACO again has a slow but steady convergence rate

Insights:

- Both single-solution search algorithms outperformed the population-based methods.
- This is likely due to how these algorithms uses the `pivot_move()` operator. Our **pivot move function is optimised for protein folding**, which potentially helps generate *better candidate* neighbouring positions than other move operators.
- SA and HC use `pivot_move()` as their main move operator. In contrast, GA relies primarily on **crossover operators**, using pivot moves only during mutation.
- Ant Colony Optimisation (ACO) operates fundamentally differently and does not use pivot moves at all.
- It's important to note that there is no universally "best" algorithm for all problems. SA performed best in our case likely because the pivot move and SA parameters (such as cooling rate and acceptance condition) were specifically well-tuned for this problem.

6.3 Comparison Benchmarking

We shall compare the predictions of the metaheuristics with the actual `2a3d` structure. To show the final output of the metaheuristics:

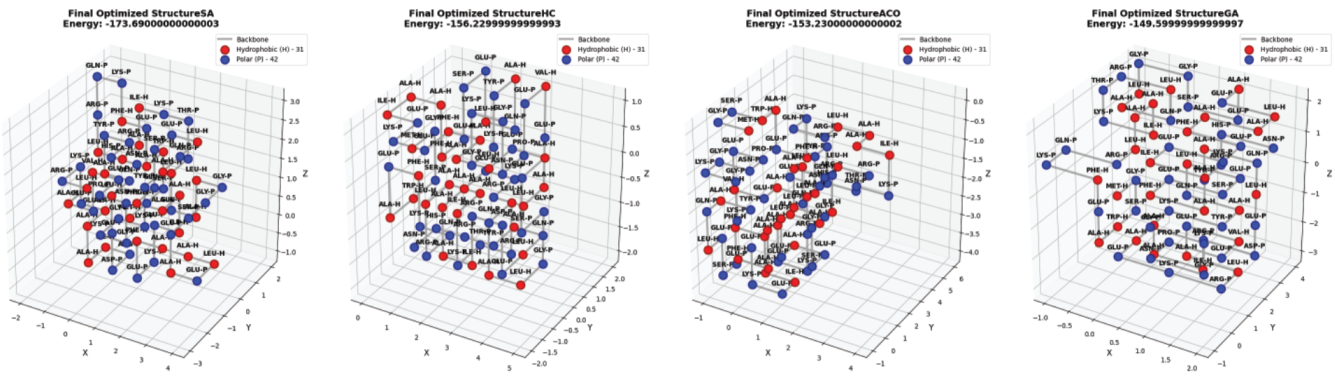


Figure 9: Outputs of metaheuristic methods when seed(21)

• Characteristic 1: Compactness

Firstly, we can observe how the solutions are in general more **compact** compared to the Greedy Solution (4), which is a key characteristic of protein folding in real life. The compactness of the solution directly correlates to the stability of the protein, as the hydrophobic core would be less exposed to interact with unstable molecules that might exist in the outside environment.

• Characteristic 2: Hydrophobic Core

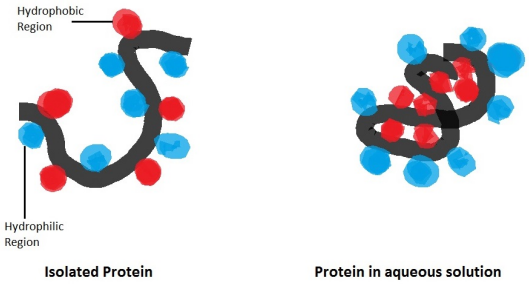


Figure 10: Visualisation of Hydrophobic Core of Folded Protein

Secondly, though not obvious in Figure 15, there were multiple times where the algorithms actually displayed a **Hydrophobic core** (the red points in the graph), where they are concentrated in the middle, which is also a key characteristic of proteins in general.

• **Comparison to Actual**

However, when we compare these structures to the actual 2a3d Protein structure:

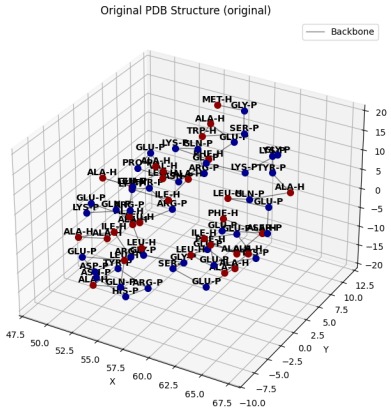


Figure 11: 2a3d Protein Structure

The difference is **huge**, which is *expected*, given the strong assumption of a discrete 3D lattice in this simplified model. In reality, amino acids can fold at much more angles, much closer together to attain lower enthalpy energies, which this model does not fully capture.

7 Conclusion & Improvements

7.1 Conclusion

In conclusion, this project uses real protein data, trying to minimise the enthalpy energy (stabilizing the shape), in an attempt to simulate nature and mimic protein folding. Researches tried to model this with the HP 3D lattice model by counting the H-H bonds, and we tried to improvise their method by referring to the MJ matrix, to calculate the theoretical energy of each bond. We therefore tried 4 different metaheuristic methods, where ACO would employ a mix of H-H bond model and the enthalpy energy model to see whether our improvisation has an impact on overall enthalpy energy.

Our 2 project goals has been reached:

1. **Minimise enthalpy** to predict folded protein structure
2. Demonstrate that **heuristics and metaheuristics** have significance in such specific biological environments

Regarding the first objective, although our model was not very successful in accurately predicting the folded protein, our methods did predict key protein structure characteristics such as the **Compactness** & **Hydrophobic Core**.

Hence, the second objective has also been reached, as there is significance in heuristics and metaheuristics in this biological space, as they have the ability to give a baseline of what possible structures of folded proteins looks like.

7.2 Further Improvements

• **Improvement 1: Error Metrics**

To understand which algorithm gave the most accurate solution compared to the real protein, we could use **Error Metrics**, such as RMSE or MSE, to quantitatively calculate the differences. In essence, this measures how far each amino acid's predicted coordinates deviate from the actual structure.

• **Improvement 2: Include Quaternary Structures**

Our current model only considers a single chain. To increase biological realism, we could extend the framework to account for quaternary structures, where **multiple folded chains** interact. This would better represent complex proteins like *haemoglobin*, which rely on inter-chain interactions for their function.

8 Appendix

Contributions:

- **Wang Zeyu:** Algorithm implementation (SA and GA), lattice constraint modelling, and final report compilation.
- **Bryan Song:** Data visualisation, 3D structure rendering, and algorithm ACO & HC.
- **Jolie Chua:** Literature review on protein folding, MJ matrix digitisation, citation

	CYS	MET	PHE	ILE	LEU	VAL	TRP	TYR	ALA	GLY	THR	SER	GLN	ASN	GLU	ASP	HIS	ARG	LYS	PRO
CYS	-5.44	-5.05	-5.63	-5.03	-5.03	-4.46	-4.76	-3.89	-3.38	-3.16	-2.88	-2.86	-2.73	-2.59	-2.08	-2.66	-3.63	-2.70	-1.54	-2.92
MET	0.70	-6.06	-6.68	-6.33	-6.01	-5.52	-6.37	-4.92	-3.99	-3.75	-3.73	-3.55	-3.17	-3.50	-3.19	-2.90	-3.31	-3.49	-3.11	-4.11
PHE	0.52	-0.22	-6.85	-6.39	-6.26	-5.75	-6.02	-4.95	-4.36	-3.72	-3.76	-3.56	-3.30	-3.55	-3.51	-3.31	-4.61	-3.54	-2.83	-3.73
ILE	0.80	-0.18	0.14	-6.22	-6.17	-5.58	-5.64	-4.63	-4.41	-3.65	-3.74	-3.43	-3.22	-2.99	-3.23	-2.91	-3.76	-3.33	-2.70	-3.47
LEU	0.59	-0.09	0.06	-0.16	-5.79	-5.38	-5.50	-4.26	-3.96	-3.43	-3.43	-3.16	-3.09	-2.99	-2.91	-2.59	-3.84	-3.15	-2.63	-3.06
VAL	0.73	-0.02	0.14	0.00	-0.01	-4.94	-5.05	-4.05	-3.62	-3.06	-2.95	-2.79	-2.67	-2.36	-2.56	-2.25	-3.38	-2.78	-1.95	-2.96
TRP	0.67	-0.63	0.12	0.19	0.11	0.13	-5.42	-4.44	-3.93	-3.37	-3.31	-2.95	-3.16	-3.11	-2.94	-2.91	-4.02	-3.56	-2.49	-3.66
TYR	0.60	-0.12	0.25	0.25	0.41	0.19	0.04	-3.55	-2.85	-2.50	-2.48	-2.30	-2.53	-2.47	-2.42	-2.25	-3.33	-2.75	-2.01	-2.80
ALA	0.59	0.29	0.31	-0.05	0.19	0.10	0.03	0.18	-2.51	-2.15	-2.15	-1.89	-1.70	-1.44	-1.51	-1.57	-2.09	-1.50	-1.10	-1.81
GLY	0.64	0.37	0.79	0.55	0.56	0.50	0.43	0.36	0.19	-2.17	-2.03	-1.70	-1.54	-1.56	-1.22	-1.62	-1.94	-1.68	-0.84	-1.72
THR	0.70	0.16	0.52	0.23	0.33	0.38	0.26	0.15	-0.04	-0.09	-1.72	-1.59	-1.59	-1.51	-1.45	-1.66	-2.31	-1.97	-1.02	-1.66
SER	0.61	0.22	0.61	0.42	0.48	0.42	0.50	0.21	0.11	0.13	0.00	-1.48	-1.37	-1.31	-1.48	-1.46	-1.94	-1.22	-0.83	-1.35
GLN	0.43	0.30	0.56	0.33	0.25	0.24	-0.01	-0.31	0.00	-0.01	-0.29	-0.18	-0.89	-1.36	-1.33	-1.26	-1.85	-1.85	-1.02	-1.73
ASN	0.93	0.33	0.67	0.91	0.70	0.91	0.39	0.10	0.61	0.32	0.14	0.23	-0.13	-1.59	-1.43	-1.33	-2.01	-1.41	-0.91	-1.43
GLU	1.23	0.43	0.50	0.47	0.58	0.49	0.36	-0.06	0.34	0.45	0.00	-0.15	-0.30	-0.04	-1.18	-1.23	-2.27	-2.07	-1.60	-1.40
ASP	0.54	0.61	0.59	0.68	0.79	0.71	0.28	0.01	0.16	-0.05	-0.32	-0.23	-0.33	-0.06	-0.16	-0.96	-2.14	-1.98	-1.32	-1.19
HIS	0.48	1.11	0.21	0.75	0.44	0.48	0.08	-0.17	0.55	0.53	-0.06	0.19	-0.02	0.18	-0.29	-0.26	-2.78	-2.12	-1.09	-2.17
ARG	0.71	0.23	0.58	0.48	0.43	0.38	-0.16	-0.28	0.44	0.10	-0.42	0.21	-0.72	0.07	-0.79	-0.80	-0.04	-1.39	-0.06	-1.85
LYS	1.11	-0.15	0.53	0.34	0.20	0.45	0.15	-0.31	0.08	0.18	-0.23	-0.15	-0.65	-0.19	-1.08	-0.90	0.24	0.57	0.13	-0.67
PRO	0.40	-0.49	0.29	0.23	0.42	0.10	-0.36	-0.43	0.03	-0.04	-0.21	-0.02	-0.69	-0.04	-0.22	-0.11	-0.19	-0.56	-0.15	-1.18

Figure 12: MJ matrix

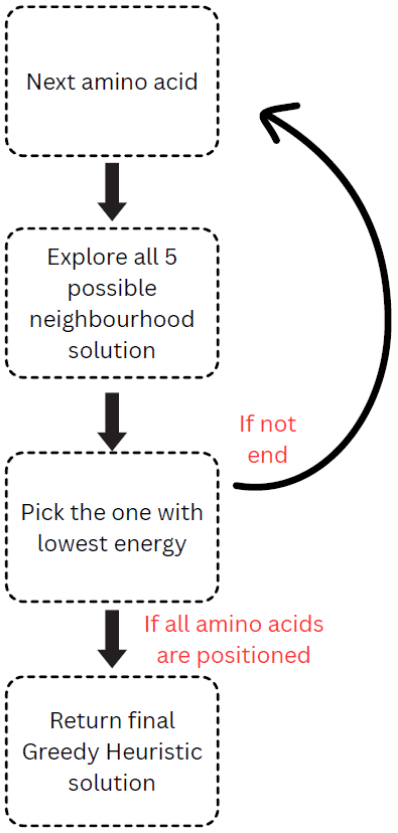


Figure 13: Greedy Heuristic Flow Chart

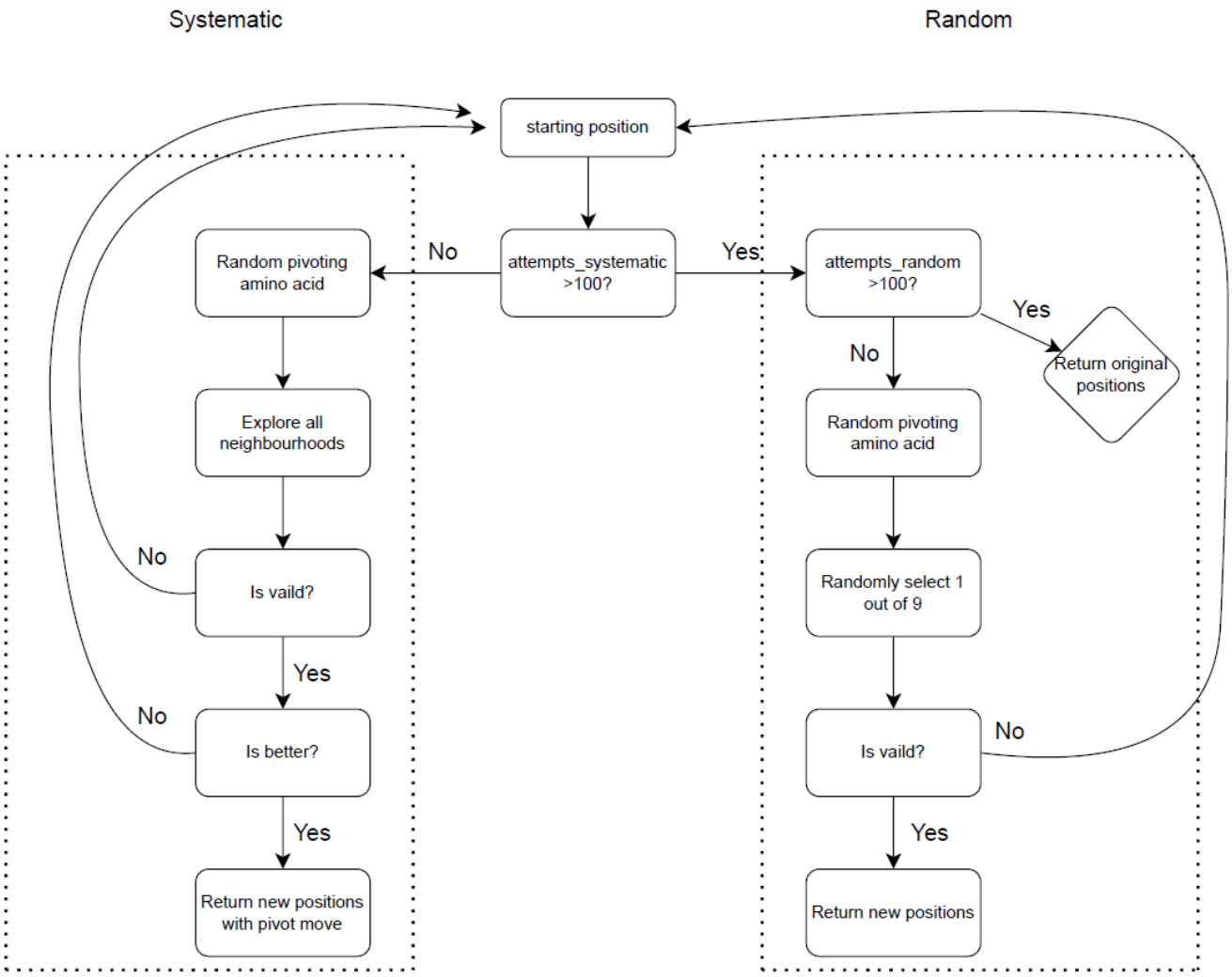


Figure 14: `pivot_move()` Flow Chart

Algorithm 7 Hill Climbing Algorithm for Protein Folding

```
procedure HILL_CLIMBING(hp_sequence, original_seq, max_iter)
    Initialise LatticeProtein from hp_sequence, original_seq
    Set current_protein, best_protein and best_energy from the initial protein
    Store energy_history and position_history
    while within time limit do
        Create trial protein as copy of current best protein
        Apply pivot move to trial protein
        Calculate energy of trial protein                ▷ Accept only improving moves
        if trial energy < best energy then
            Update best_energy ← trial energy
            Update best_protein ← trial protein
        Record current energy and best position in histories
        Record energy histories in energy_history
    return best_protein, energy_history, position_history
```

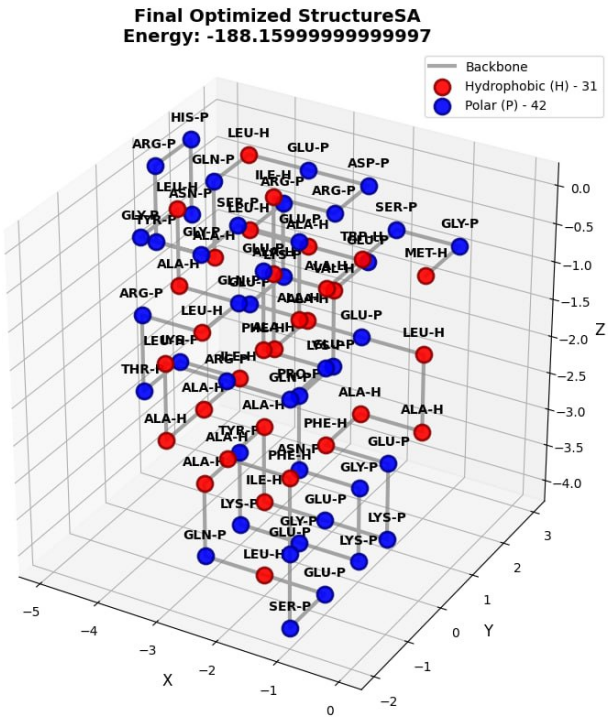


Figure 15: HOF: Lowest Enthalpy Folding using SA

References

[1] N. Boumedine, S. Bouroubi, *Protein folding in 3D lattice HP model using a combining cuckoo search with the Hill-Climbing algorithms*, Applied Soft Computing, Volume 123, 2022, 108564. https://doi.org/10.1016/j.asoc.2022.108564

[2] Y. Guo, F. Tao, Z. Wu, et al., *Hybrid method to solve HP model on 3D lattice and to probe protein stability upon amino acid mutations*. BMC Systems Biology, 11(Suppl 4), 93 (2017). https://doi.org/10.1186/s12918-017-0459-4

[3] G. Slade, *The Self-Avoiding Walk: A Brief Survey*, 2013. https://secure.math.ubc.ca/ slade/intelligencer.pdf

[4] S. Miyazawa and R. L. Jernigan, *Estimation of Effective Interresidue Contact Energies from Protein Crystal Structures: Quasi-Chemical Approximation*, Macromolecules, 18(3), 534–552 (1985). https://pubs.acs.org/doi/abs/10.1021/ma00145a039