Roadmap HP model in 2D and 3D: Model implementation and research

In this part of the project, we shall focus on implementing and testing the 2D and 3D Hydrophobic-Polar (HP) model. A functional 2D model (available in the shared GitHub repository) computes energies and generates random 2D protein structures. Afterwards, we will leverage the existing 3D Self-Avoiding Walk (SAW) code (also in the repository) to implement energy calculations for non-bonded hydrophobic monomers in 3D, similar to the 2D case, and potentially explore additional folding functions. The following outlines the steps to be completed by April for the report submission.

1. Step_1:

Objective:
Implementing the simulated annealing method in the 2d_HP_model.py code.

Indications:

The main idea is that you implement within this code the following function (this is a pseudo-code to give you an idea of what is needed)

```
function simulated_annealing(seq, initial_struct, e, max_iterations, initial_temp,
cooling_rate):
    """
    Performs simulated annealing to find the minimum energy protein structure.

    Args:
        seq: The protein sequence.
        initial_struct: The initial protein structure.
        e: The energy contribution per H-H bond.
        max_iterations: The maximum number of iterations.
        initial_temp: The initial temperature.
        cooling_rate: The rate at which the temperature decreases.

    Returns:
        The protein structure with the lowest energy found.
    """

    current_struct = initial_struct
    current_energy = calculate_energy(e, seq, current_struct)
    best_struct = current_struct
    best_energy = current_energy
    temperature = initial_temp

    for iteration in range(max_iterations):
        # Generate a new structure by applying a random transformation
        new_struct = tail_fold(current_struct.copy())
        new_energy = calculate_energy(e, seq, new_struct)

        # Calculate the energy difference
```

```
        energy_diff = new_energy - current_energy

        # Accept the new structure if it has lower energy or with a probability
        # based on the Boltzmann distribution
        if energy_diff < 0 or random.random() < math.exp(-energy_diff / temperature):
            current_struct = new_struct
            current_energy = new_energy

            # Update the best structure if necessary
            if current_energy < best_energy:
                best_struct = current_struct
                best_energy = current_energy

        # Cool down the temperature
        temperature *= cooling_rate

    return best_struct
```

2. **Step_2:** Organise the code using classes and possible a utils.py for functions that we can call outside (this is like import utils at the beginning of the python code) and possibly we need to optimise afterwards (I believe the function *count_hh_non_bonded_neighbors* is one of the candidates). I would like to have a code that is modular and very easy to use for any user.

3. **Step_3:** Implementing the BPSO approach to this problem.

The main idea to implement the BPSO is to build a function like:

```
function binary_pso(seq, e, num_particles, max_iterations, w, c1, c2):
    """
    Performs binary Particle Swarm Optimization to find the minimum energy protein
structure.

    Args:
        seq: The protein sequence.
        e: The energy contribution per H-H bond.
        num_particles: The number of particles in the swarm.
        max_iterations: The maximum number of iterations.
        w: The inertia weight.
        c1: The cognitive coefficient.
        c2: The social coefficient.

    Returns:
        The protein structure with the lowest energy found.
    """

    # Initialize particles
```

```
    particles = []
    for i in range(num_particles):
        structure = generate_random_valid_structure(seq) # Use existing functions to create
this
        velocity = generate_random_binary_velocity(len(seq))
        particles[i].position = structure
        particles[i].velocity = velocity
        particles[i].best_position = structure
        particles[i].best_energy = calculate_energy(e, seq, structure)

    # Find initial global best
    global_best_position = find_best_structure(particles) # Function to find the best
structure in the swarm
    global_best_energy = calculate_energy(e, seq, global_best_position)

    # Main PSO loop
    for iteration in range(max_iterations):
        for particle in particles:
            # Update velocity
            for i in range(len(seq)):
                r1 = random.random()
                r2 = random.random()
                cognitive_component = c1 * r1 * (particle.best_position[i][0] - particle.position[i]
[0])
                social_component = c2 * r2 * (global_best_position[i][0] - particle.position[i][0])
                particle.velocity[i] = w * particle.velocity[i] + cognitive_component +
social_component

                # Apply sigmoid function to get probability
                probability = 1 / (1 + exp(-particle.velocity[i]))

                # Update position
                if random.random() < probability:
                    temp_struct = particle.position.copy()
                    temp_struct[i] = apply_random_transformation(temp_struct, i) # Apply
transformation starting at i-th monomer
                    if valid_move(temp_struct):
                        particle.position = temp_struct

            # Update personal best
            current_energy = calculate_energy(e, seq, particle.position)
            if current_energy < particle.best_energy:
                particle.best_position = particle.position
                particle.best_energy = current_energy

                # Update global best
```

```
            if current_energy < global_best_energy:
                global_best_position = particle.position
                global_best_energy = current_energy

    return global_best_position
```

4. **Step_4: Performance Optimization and Analysis**

This step focuses on optimizing the performance of the HP model implementation. We will:

- **Transfer Function Evaluation:** Explore various transfer function implementations and assess their accuracy across different protein lengths. These functions will be developed as a separate module and integrated into the main codebase.
- **Performance Profiling:** Analyze the code's efficiency and scalability using profiling tools like `scalene` or `cProfile`. Identify potential bottlenecks and areas for optimization, including opportunities for parallelization.
- **Optimization Strategies:** Investigate and implement optimization techniques using libraries like `numba` or `cython` to improve code execution speed and scaling.

Objective:

- Gain proficiency in code organization and profiling.
- Learn to apply optimization libraries effectively.
- Evaluate the performance impact of different transfer functions and optimization strategies.

Deliverables:

- A comprehensive analysis of the code's performance, including profiling results and optimization strategies.
- A dedicated chapter in the final report summarizing the performance evaluation, highlighting the advantages and disadvantages of different implementations, and discussing the effectiveness of the optimization techniques employed.

Enhancement and Implementation of the 3D Hydrophobic-Polar (HP) Model

This document outlines the proposed enhancements and implementations for the 3D Hydrophobic-Polar (HP) model. The current version of the 3D Self-Avoiding Walk (SAW) code is available in the shared GitHub repository.

Proposed Enhancements:

1. **Protein Sequence Integration and 3D Structure Generation:**
    - Modify the existing 3D-SAW code to accept protein sequences as input.
    - Generate 3D linear protein structures from the input sequences, analogous to the 2D case.
    - Implement functionality to generate diverse random protein configurations from the initial linear structure using existing 3D-SAW functions.

- Refactor the code, organizing functions into a class for a user-friendly interface. Comprehensive documentation should be included.

2. **Energy Function Calculation:**
    - Develop a `count_hh_non_bonded` function to calculate the number of non-bonded hydrophobic (H) neighbors for any given protein configuration in the 3D-SAW. This function will directly provide the energy of a conformation.

3. **Advanced Optimization Algorithms (Optional):**
    - If time permits, explore implementing simulated annealing and a Binary-Based Particle Swarm Optimization (BPSO) algorithm for the 3D HP model. This step is considered ambitious and may be subject to time constraints.