Hill Climbing Algorithm Implementation

September 13, 2023

In optimization problems, we aim to find the best solution from a set of possible solutions. In this exercise, we are tasked with optimizing the placement of hospitals in a state space to minimize the total distance from houses to the nearest hospital. You will be implementing a hill climbing algorithm, a local search algorithm which iteratively explores neighboring states, moving to the neighbor that improves the objective function (in this case, reduces the total distance).

1. First, create a class called `Space`. This class represents the state space and will contain methods to implement the hill-climbing algorithm.
2. In the `__init__` method of the `Space` class:
   - Initialize the state space with the given dimensions `height` and `width`, and the number of hospitals `num_hospitals`.
   - Initialize empty sets for houses and hospitals to keep track of their positions.
3. Implement the `add_house` method:
   - This method should take in a `row` and `col` and add a house at that location in the state space. Use the set data structure to store house positions.
4. Implement the `available_spaces` method:
   - This method should return all cells that are not currently occupied by a house or hospital.
   - Initialize a set `candidates` to contain all possible cells.
   - Use nested loops to iterate through rows and columns to populate `candidates`.
   - Remove cells that are occupied by houses and hospitals from `candidates`.
5. Implement the `hill_climb` method:
   - This method will implement the hill-climbing algorithm to find an optimal solution.

- Start by initializing hospitals randomly by adding them to the `hospitals` set.
- Set up a loop that runs until either a maximum number of iterations is reached or the algorithm converges (you have found a solution).
- Inside the loop, calculate the cost of the current state using the `get_cost` method.
- Find the best neighboring state by iterating through hospitals and their neighbors. Use the `get_neighbors` method to find neighbors.
- Decide whether to move to the best neighbor or not based on its cost. If the neighbor's cost is better, update the current state.
- If the cost of the best neighbor is not better than the current state, return the current state, as there are no better neighbors.

6. Implement the `random_restart` method:
   - This method repeats the hill-climbing algorithm multiple times to potentially find better solutions.
   - Initialize variables to store the best hospitals and their cost.
   - Use a loop to run the hill-climbing algorithm multiple times.
   - Compare the cost of the current solution with the best solution found so far and update them if necessary.
   - Return the best solution found.
7. Implement the `get_cost` method:
   - This method calculates the cost of a state by summing the distances from houses to the nearest hospital.
   - Iterate through each house, find the nearest hospital, and add the distance to the cost.
8. Implement the `get_neighbors` method:
   - This method returns neighboring cells that are not already occupied by a house or hospital.
   - Define a list of candidate neighbors by specifying their relative positions.
   - Iterate through candidates, filter out occupied cells, and return the valid neighbors.
9. Finally, create an instance of the `Space` class and add houses randomly to the state space. Then, use the `hill_climb` and `random_restart` method to determine hospital placement. What do you notice? What method provides the best results?

Make sure to fill in the gaps in the template code with the appropriate code based on the instructions provided above.

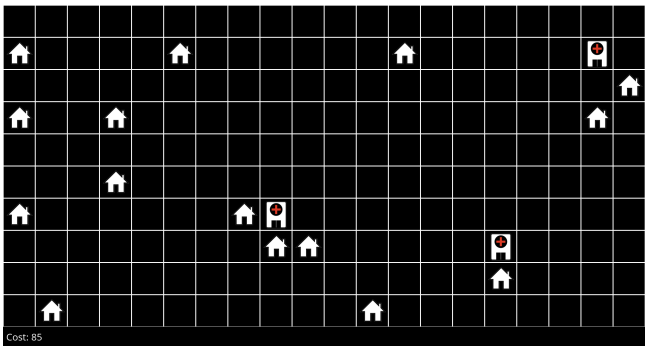

Figure 1: Example of image output: a possible solution to the hospital allocation problem.