Evolutionary computation

Tasks 2024/2025

2 Optimization Algorithm: Hill Climbing

2.1 Hill Climbing Algorithm with Steepest-Ascent Strategy

In the second part of the task, you will implement the Hill Climbing algorithm with the Steepest-Ascent strategy to find the optimal solution for test problems from the first task. Implement the class HillClimbing that inherits from the Algorithm class. The algorithm should have the parameter stepSize, which defines the step size for each dimension when generating new solutions. The algorithm should generate 2 * d neighbors in each step, where d is the number of dimensions of the problem. Among all neighbors, the algorithm selects the one with the best fitness if it improves the current solution.

Pseudocode for the Hill Climbing Algorithm:

Algorithm 1 Pseudocode for Hill Climbing with Steepest-Ascent

Input: $Problem$: contains the fitness function to optimize
Input: $stepSize$: size of each step for generating neighbors
Input: $maxFes$: maximum number of evaluations
Output: $bestSolution$: the best solution found
Initialize $bestSolution \leftarrow$ generateRandomSolution from $Problem$
Initialize $fes \leftarrow 1$ {Counter for fitness evaluations}
while $fes <= maxFes$ **do**
  Initialize $bestNeighbor \leftarrow bestSolution$
  Initialize $improved \leftarrow$ false
  Generate $2 * D$ neighbors, where D is the number of dimensions
  Evaluate each neighbor
  $fes \leftarrow fes + 2 * D$
  if A $neighbor$ is better than $bestNeighbor$ **then**
    $bestNeighbor \leftarrow neighbor$
    $improved \leftarrow$ true
  end if
  if $improved$ **then**
    $bestSolution \leftarrow bestNeighbor$
  else
    break {Stop if no improvement found}
  end if
end while
return $bestSolution$

To analyze the algorithm's performance, you need to perform statistical analysis of the results. Create a separate class StatisticsUtility containing methods to find the best solution, calculate the average, and calculate the standard deviation. Run the HillClimbing algorithm 100 times on each test problem from the first task, recording the best solution for each run, with maxFes set to **3000 * d**. If a problem allows for multiple dimensions, run the algorithm for dimensions 2, 5, and 10. Using StatisticsUtility, output for each problem the best solution (minimum value), average, and standard deviation for each test problem and dimension.

Example output for results of the Sphere problem with dimension 2:

```
Problem: Sphere, Dimensions: 2 Min: 5.08e-57, Average: 0.045, Std: 0.012
```

* Deadline: **November 28, 2024**.
* The task is **mandatory** and worth **10 points**.

2.2 Improving the Hill Climbing Algorithm

In this task, you will enhance the basic Hill Climbing algorithm. Your task is to select and implement one or more techniques to improve the algorithm's performance. Possible strategies for improving the algorithm include:

- **Dynamic Step Size Adjustment (stepSize)**: Instead of a fixed step size, try adjusting the step size based on the algorithm's performance or the current iteration.
- **Increasing the Number of Generated Neighbors**: Experiment with generating more or fewer neighbors in each step and observe the impact on the algorithm's performance.
- **Multiple Runs with Different Initial Solutions**: Run the algorithm multiple times from different starting points and combine the best solutions to increase the chances of finding a global optimum.
- **Hybridization with Simple Random Search (Random Search)**: Use random search in combination with the Hill Climbing algorithm for better exploration of the solution space.
- . . .

Task Requirements

Choose one or more strategies that you believe will improve the efficiency of the Hill Climbing algorithm. Implement the improved version in a new class (e.g., ImprovedHillClimbing). Perform a statistical analysis of the results for each improvement and compare the performance of the basic and improved versions of the algorithm. Similar to the basic version, run the improved algorithm 100 times on each test problem from the first task, with maxFes set to **3000 * d**, and run the algorithm for dimensions 2, 5, and 10 where applicable. Use the StatisticsUtility class to calculate the average, standard deviation, and best solution. Print the results in the console in the same format as the basic algorithm version. Briefly interpret the results, including whether the improved algorithm found better solutions on average compared to the basic version. Also, mention any potential drawbacks or limitations of your improved version. Prepare a table clearly comparing the basic and improved versions of the algorithm for each test problem and dimension. The table should contain columns for the average, standard deviation, and best solution for both algorithm versions.

Example comparison output:

```
Problem: Sphere, Dimensions: 2
Original - Min: 5.08e-57, Average: 0.045, Std: 0.012
Improved - Min: 3.00e-57, Average: 0.035, Std: 0.009
```

* Deadline: **November 28, 2024**.
* The task is **optional** and worth **5 points**.