Mini Project In Stochastic Optimization
Project Report

Task

Given n points in a plane, the algorithms selectively increase the weights of the points while ensuring that the weight of the heaviest increasing subsequence remains within a specified maximum limit M = the length of the longest initiating path (it's the same as the weight at the initial).
Additionally, we aim to maximize the sum of the first moment and also the sum of the second moment of the weights of the points while adhering to this constraint.
The algorithms should efficiently handle large datasets of points.

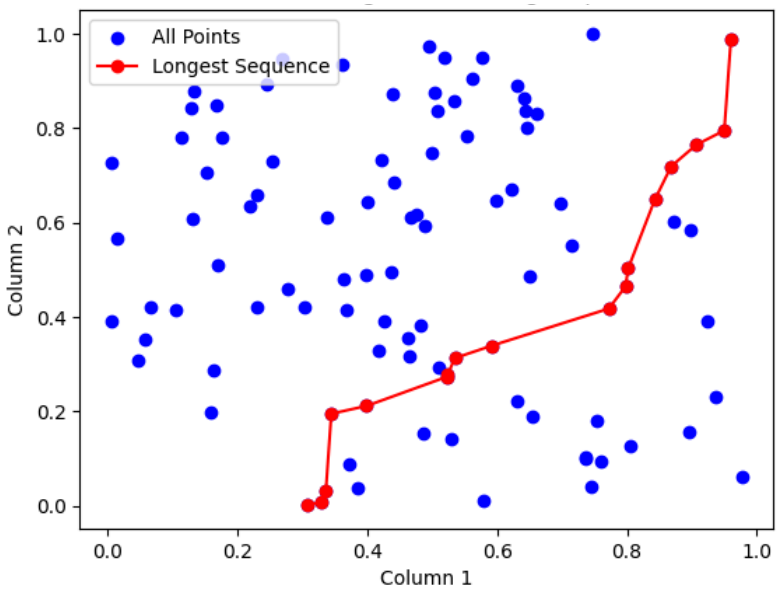
Our Idea is to approach the points that no other point is greater than them, which means - if $\alpha = (\alpha_x, \alpha_y)$ is a point that there's no $\beta = (\beta_x, \beta_y)$ for which $\alpha_x < \beta_x$ and also $\alpha_y < \beta_y$.
Since all of the increasing routes end in those points (if not- it's not a maximal route), our idea is to put most of the weights on these points, making them the heaviest they can get.
This will maximize the sum of the first moment and also the second moment, because if we want to maximize the second moment we want to increase the weight of the minimal amount of points so the square will be greater so we want the separation of the weight will be closer.
The maximization of the second moment will also maximize the first moment since the weights will be distributed between fewer points, but the maximization will still work.

So we decided that the increasing algorithm will be good for both problems.

In the algorithm we will calculate the heaviest path to each α - we'll call it A.
We'll change the weight of α to be- $(M - A + 1)$.
This will make the weight of the heaviest path to α up to M, so the constraint at the graph will be maintained.
The purpose is to save a lot of time going through all of the given points- since it can be a very big number.
This idea might not lead to the highest sum of all the point's weight, but it will get us a very good sum in a reasonable running time for both problems.

Example of our idea

In the pictures below we can see the heaviest route in the initial state.

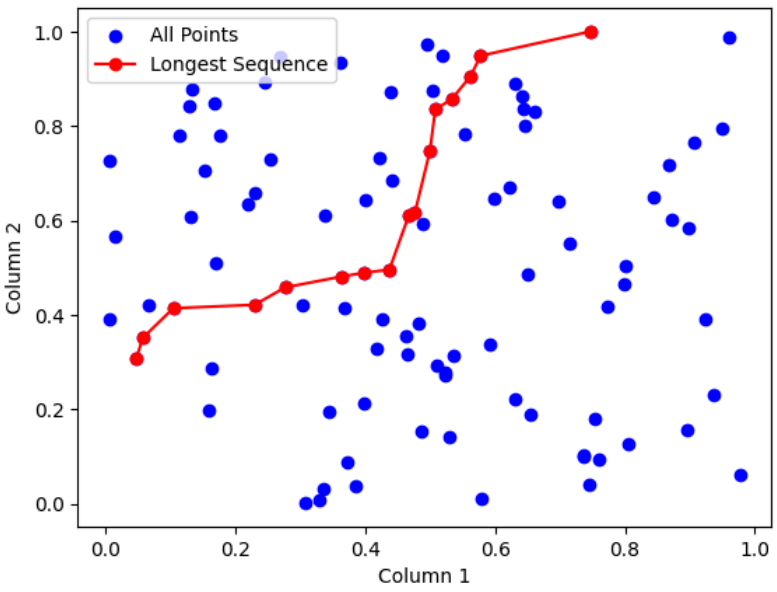


Subsequence with Maximal Weight:

	Column 1	Column 2	w
25	0.308138	0.001913	1
26	0.329269	0.007098	2
27	0.336554	0.031205	3
29	0.344952	0.194672	4
36	0.399367	0.211249	5
57	0.523413	0.273207	6
58	0.524061	0.277485	7
61	0.535125	0.312510	8
66	0.591281	0.338487	9
85	0.772148	0.417363	10
86	0.797800	0.464424	11
87	0.801887	0.502510	12
89	0.842943	0.648385	13
90	0.868175	0.718817	14
94	0.906783	0.764351	15
97	0.949808	0.794737	16
98	0.960033	0.988169	17

Maximal Weight: 17

In the pictures below we can see the heaviest route after running our algorithm.
We can see that the weight is still 17, but the number of points in this route is now 16, meaning that the constraint is still maintained.

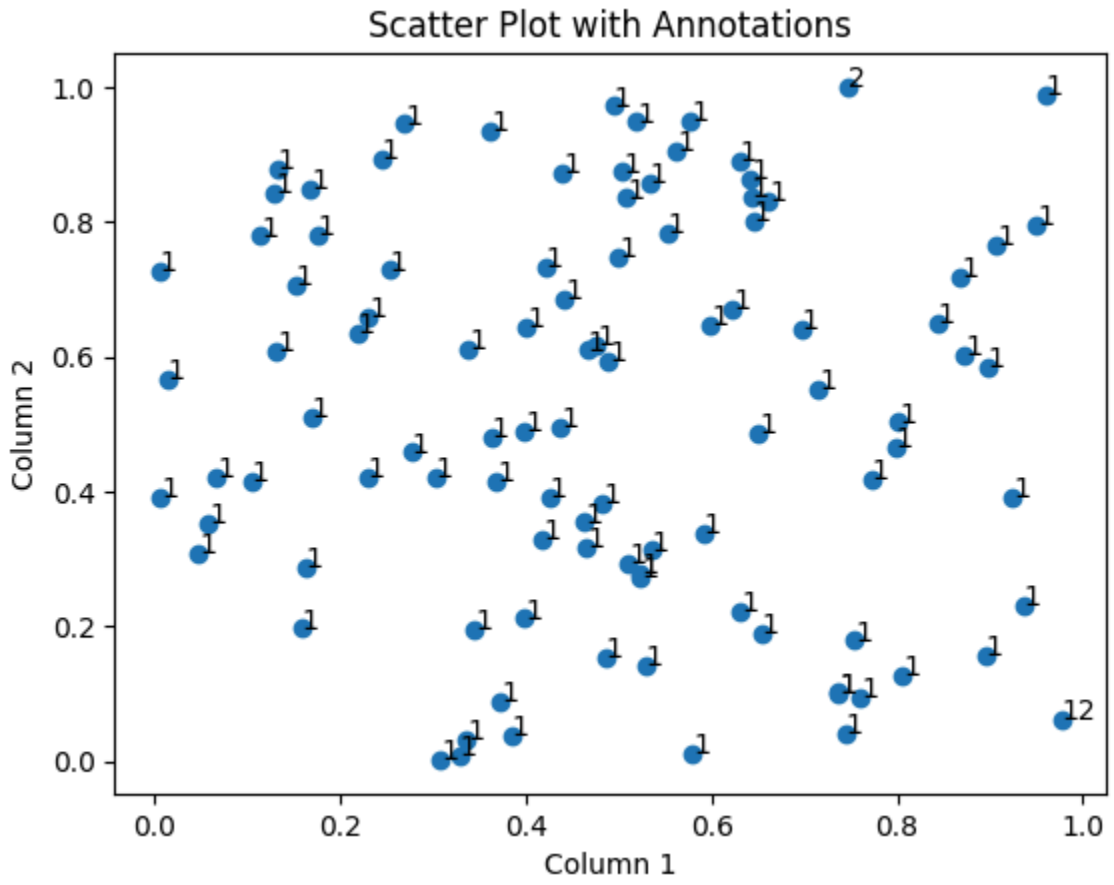


Subsequence with Maximal Weight:

	Column 1	Column 2	w
3	0.047886	0.308545	1
4	0.058239	0.351714	2
6	0.104889	0.413932	3
18	0.230124	0.421267	4
23	0.276752	0.457917	5
31	0.363121	0.480918	6
35	0.398991	0.489089	7
41	0.437373	0.495767	8
46	0.467040	0.609652	9
47	0.476587	0.615840	10
52	0.499836	0.746949	11
54	0.508087	0.834949	12
60	0.533437	0.856891	13
63	0.562488	0.905779	14
64	0.577708	0.948737	15
82	0.746164	0.999939	17

Maximal Weight: 17

In the picture below we can see how we maximize the weights.
All the weight we increased is in the “highest” points of the graph, or as we called them before-
α points.



The Algorithm

The algorithm begins by processing the input data,where the initial weight of each point is 1. Subsequently, the points are sorted based on their coordinates, facilitating efficient identification of higher points for weight adjustment.

Utilizing dynamic programming, the algorithm computes the heaviest increasing subsequence among the sorted points, maintaining constraints on its weight throughout the process.

Through strategic weight adjustments, prioritizing higher points, the algorithm maximizes the sum of weights while ensuring adherence to the predefined constraints.

Finally, the algorithm outputs the subsequence with the maximal weight and its corresponding weight, providing insights into the optimized weight distribution while preserving the properties of the longest increasing subsequence.

Overall, by integrating sorting, dynamic programming, and strategic weight adjustments, the algorithm effectively addresses the task, offering a practical solution to weight maximization in increasing subsequences.

We'll explain what each function does:

maximal_weight_increasing_subsequence(file_path) Function:

This function plays a crucial role in the algorithm by dynamically updating the weights of points to maximize the weight of the longest increasing subsequence while ensuring adherence to the specified constraints. It enables the algorithm to efficiently identify and optimize the subsequence with the maximal weight, contributing to the overall objective of weight maximization in increasing subsequences.

- Parameters:
  The function takes a file path as input, which specifies the location of the input file containing the coordinates of points in the plane.

- Output:
  It returns the subsequence with the maximal weight and its corresponding weight.

- Flow:
  The function reads the data from the specified Excel file, which contains the coordinates of points in the plane along with their initial weights.
  It sorts the data points based on their x and y coordinates.
  Sorting the points is essential for subsequent operations, such as finding the longest

increasing subsequence and weight maximization.
The function utilizes dynamic programming to iteratively update the weights of points to maximize the weight of the longest increasing subsequence while adhering to the given Constraints.
The function iterates through each point and considers its predecessors to determine the maximum weight achievable while maintaining the LIS property.
It updates the weights of points dynamically based on the calculated maximal weight.
After determining the maximal weight, the function reconstructs the subsequence with the maximal weight by backtracking through predecessors.
Finally, the function returns the subsequence with the maximal weight and its corresponding weight.

increase_values (path, x) Function:

The function adjusts the weights of points based on their positions relative to each other.
It ensures that the maximum weight specified is achieved while maintaining certain conditions regarding the order of points.

- Parameters:
  - path: The path to the Excel file containing the data.
  - x: The maximum weight that needs to be achieved.

- Flow:
  The function gets the points from the path, then sorts the data based on the point's x and y axis.
  After that the function initialize a dynamic programming list to store cumulative weight values and iterate through each row of the DataFrame.
  In the iteration we calculate the cumulative weight for the current row, considering preceding points that satisfy certain conditions.
  Then we update the cumulative weight for the current row based on the maximum weight achieved.

  At last we adjust the weights of points that do not have any following points with larger x and y values to ensure that the maximum weight specified (x parameter) is attained.
  We save the weights.

Data Handling and Preprocessing

Sorting Points:

In the algorithm we sorted the points based on their coordinates to facilitate subsequent operations.

Sorting the points is crucial as it enables efficient processing of points and facilitates the identification of higher points, which are candidates for weight increase, since all routes ends in these higher points.

Our Idea was to address these points when we increase the weights since all routes ends there, so we will put most of the weights on them, instead of iterating all n points which might take a while.

Results:

The algorithm successfully achieves the task objectives by effectively increasing the weights of points while adhering to the specified constraints.

Sorting the points plays a crucial role in identifying higher points, which are prioritized for weight increase.

It maximizes the sum of weights of points while ensuring that the weight of the longest increasing subsequence does not exceed the predefined maximum limit.

The algorithm is capable of handling large datasets efficiently, making it suitable for practical applications.

Conclusion:

In conclusion, the developed algorithm provides an effective solution to the problem of weight maximization in increasing subsequences. By employing a combination of dynamic programming techniques and strategic weight adjustments, the algorithm achieves the desired objectives while maintaining computational efficiency. The results obtained demonstrate the algorithm's capability to handle real-world datasets and its relevance to the specified task requirements.