Topics in Stochastic Optimization

Mini Project

By Ido Hassidim (209703222)

and Harel Meshulam (206649683)

Introduction

The problem involves analyzing a set of points in a plane, where each point has an initial weight of 1. Points are compared based on their coordinates, with one point said to "win" against another if it is greater in both X and Y dimensions. A chain is a series of points where each point wins against its predecessor, and the goal is to maximize the weight of chains while adhering to specific constraint - no chain's total weight exceeds the original maximum chain length (W).

The goal is to reach a final state where no more points can be increased to weight 2, while minimizing the number of points that end up with weight 2.

The Process

A brief explanation for each stage of the solution:

1. Data Preparation: Firstly, reading points from Excel file and sorting them by X, then Y coordinates. Then, creating data structures, for example, tracking valid predecessors for each point, or storing the sheets of the Excel file.

2. Initial W Calculation: Uses dynamic programming approach.

For each point i:

- Finds valid predecessors (points that can come before i in a chain).
- Updates $dp\_arr[i]$ with maximum chain length ending at point i.
- Stores predecessor information in $prev$ array.

3. Point Selection Strategy: Selects points with highest dp_arr value less than W. This strategy tries to pick points that are "almost" in a maximum chain but not quite. Therefore, there are more points banned per iteration in the next step, which results in fewer total iterations needed.

4. Chain Recalculation and Point Banning: Recalculate chains affected by the weight change. Update the set of banned points (points that can't be increased to 2). Update available points for future selections. This ensures maintaining the W constraint. While there are available points, select the next point to increase weight.

Chain Weight Calculation

Next, we will provide a more detailed explanation. We will begin by discussing how W is calculated, starting with the determination of the initial value of W. We will then explain how the weights of points are adjusted without exceeding W.

Initial W

This is like finding the longest possible path through our points, where each step must move "up and right" (beating the previous point).
Here's how it works: For each point, we ask: "What's the longest chain we can make ending at this point?"
First, we find all points that could come before it (valid predecessors). This is easy, since we sorted the points.
Then we look at the longest chains ending at each of these predecessors.
Then we add 1 (the current point) to the longest predecessor chain. This gives us the longest chain ending at our current point.
As we go through each point, we keep track of: How long the chain is up to that point (in dp_arr) and which points came before in the best chain (in $prev$ array).
After checking all points, the largest value we found in dp_arr becomes W, and we recursively ban all $prev$ points, which are part of the a chain with weight W (meaning, we can't increase their weight).
All of this happens inside the function $find\_longest\_chains()$.

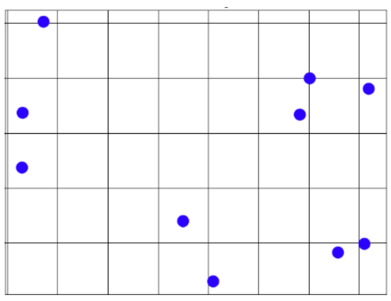
After Increasing Point Weight

We use the same dynamic programming technique as we did for the initial W in $find\_longest\_chains$.
The difference is we will use fix_dp instead. This function uses the fact that changing a point's weight does not affect previous cells in dp_arr. This is because chains that end with points before the current one won't be effect by the increase of weight to the current point.
We will rebuild dp_arr, starting from the index of the chosen point. Doing so means at the worst case dp_arr will be rebuilt from the smallest point, and in the general case only part of it will be rebuilt.
This approach reduces unnecessary calculations, and makes the algorithm run much faster.

Smart Way of Choosing a Point

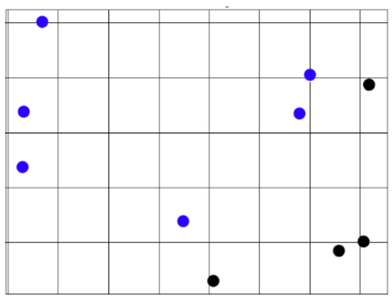
So far we chose a point, increased its weight and took care of the effect it had. If there are any available points, we select another point, until there are no longer points that their weight can be increased.
The obvious question is what is the best way of choosing a point?
We want each point we chose, will ban more points.
Therefore, we will have less iterations in total which will help with reducing the run time and we will complete the task objective - minimizing the number of points that end up with weight 2.
We have concluded that the best way to achieve this, is selecting the end point of a chain its weight is as close to W. This choice will lead to more banned points every iteration, hence achieving our goal.

Demonstration



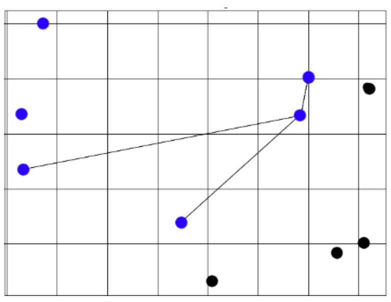
Initial W calculation. $W = 4$ and the black points are now banned.

$\longrightarrow$



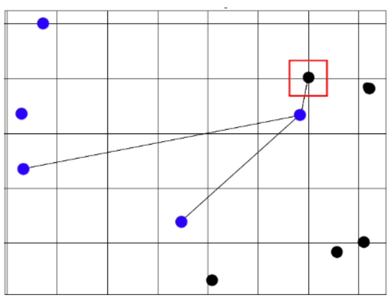
The next heaviest chains have w=3 as marked in the picture.

$\longrightarrow$



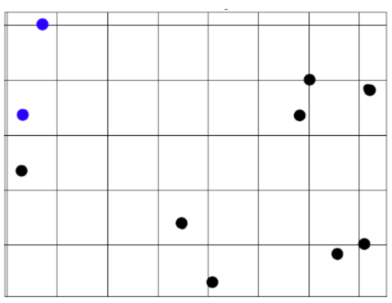
This point has been selected to have its weight increased, as it is an end point that has the largest value in dp_arr.

$\longrightarrow$



Since increasing its weight causes the chains to have weight of 4, all the points in the chains are now banned

$\longrightarrow$



And so on until there are no more available points.

$\longrightarrow$

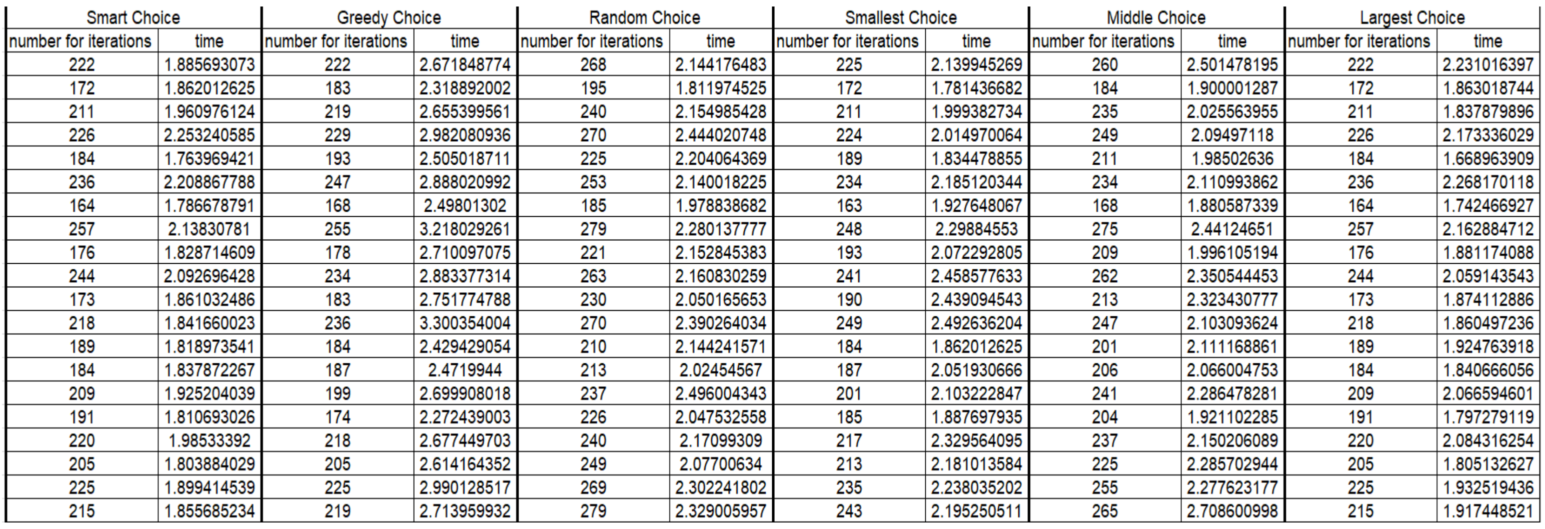
More Ways of Choosing a Point

We have implemented a few more ways to the point selection strategy.

- Random Choice: From all the available points, choose a random point to increase its weight.
- Picky Choice: From all the available points, choose a point with the smallest/middle/largest coordinates to increase its weight.
- Greedy Choice: For each available point, calculate its appearance in more potential chains. This is the “score” of a point. Choose the point with the largest score.

Comparing The Ways of Choosing a Point

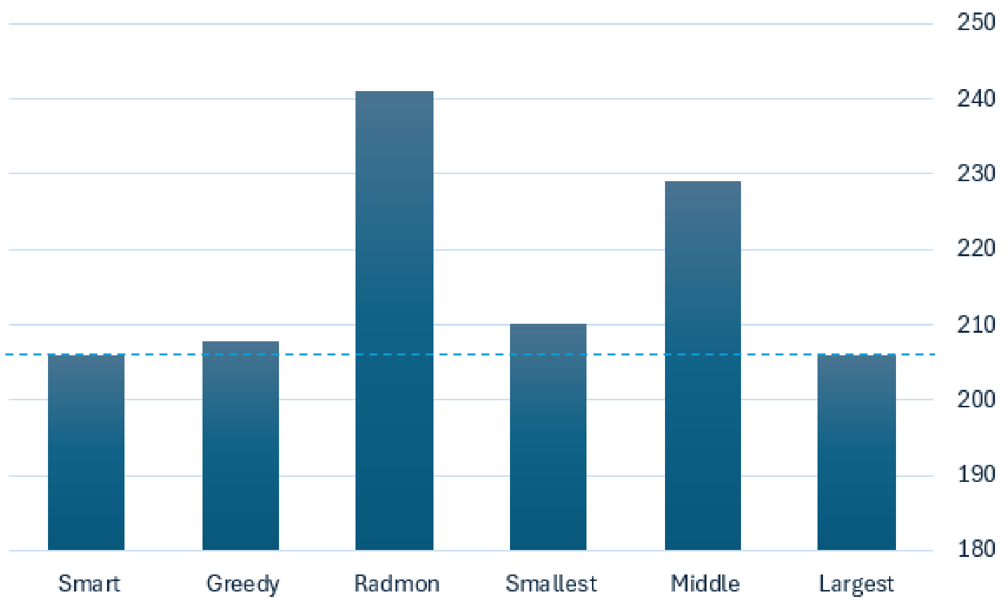
We ran the different algorithms with 20 unique files, each including 500 random points. Here are the results:



Smart Choice		Greedy Choice		Random Choice		Smallest Choice		Middle Choice		Largest Choice	
number for iterations	time	number for iterations	time	number for iterations	time	number for iterations	time	number for iterations	time	number for iterations	time
222	1.885693073	222	2.671848774	268	2.144176483	225	2.139945269	260	2.501478195	222	2.231016397
172	1.862012625	183	2.318892002	195	1.811974525	172	1.781436682	184	1.900001287	172	1.863018744
211	1.960976124	219	2.655399561	240	2.154985428	211	1.999382734	235	2.025563955	211	1.837879896
226	2.253240585	229	2.982080936	270	2.444020748	224	2.014970064	249	2.09497118	226	2.173336029
184	1.763969421	193	2.505018711	225	2.204064369	189	1.834478855	211	1.98502636	184	1.668963909
236	2.208867788	247	2.888020992	253	2.140018225	234	2.185120344	234	2.110993862	236	2.268170118
164	1.786678791	168	2.49801302	185	1.978838682	163	1.927648067	168	1.880587339	164	1.742466927
257	2.13830781	255	3.218029261	279	2.280137777	248	2.29884553	275	2.44124651	257	2.162884712
176	1.828714609	178	2.710097075	221	2.152845383	193	2.072292805	209	1.996105194	176	1.881174088
244	2.092696428	234	2.883377314	263	2.160830259	241	2.458577633	262	2.350544453	244	2.059143543
173	1.861032486	183	2.751774788	230	2.050165653	190	2.439094543	213	2.323430777	173	1.874112886
218	1.841660023	236	3.300354004	270	2.390264034	249	2.492636204	247	2.103093624	218	1.860497236
189	1.818973541	184	2.429429054	210	2.144241571	184	1.862012625	201	2.111168861	189	1.924763918
184	1.837872267	187	2.4719944	213	2.02454567	187	2.051930666	206	2.066004753	184	1.840666056
209	1.925204039	199	2.699908018	237	2.496004343	201	2.103222847	241	2.286478281	209	2.066594601
191	1.810693026	174	2.272439003	226	2.047532558	185	1.887697935	204	1.921102285	191	1.797279119
220	1.98533392	218	2.677449703	240	2.17099309	217	2.329564095	237	2.150206089	220	2.084316254
205	1.803884029	205	2.614164352	249	2.07700634	213	2.181013584	225	2.285702944	205	1.805132627
225	1.899414539	225	2.990128517	269	2.302241802	235	2.238035202	255	2.277623177	225	1.932519436
215	1.855685234	219	2.713959932	279	2.329005957	243	2.195250511	265	2.708600998	215	1.917448521

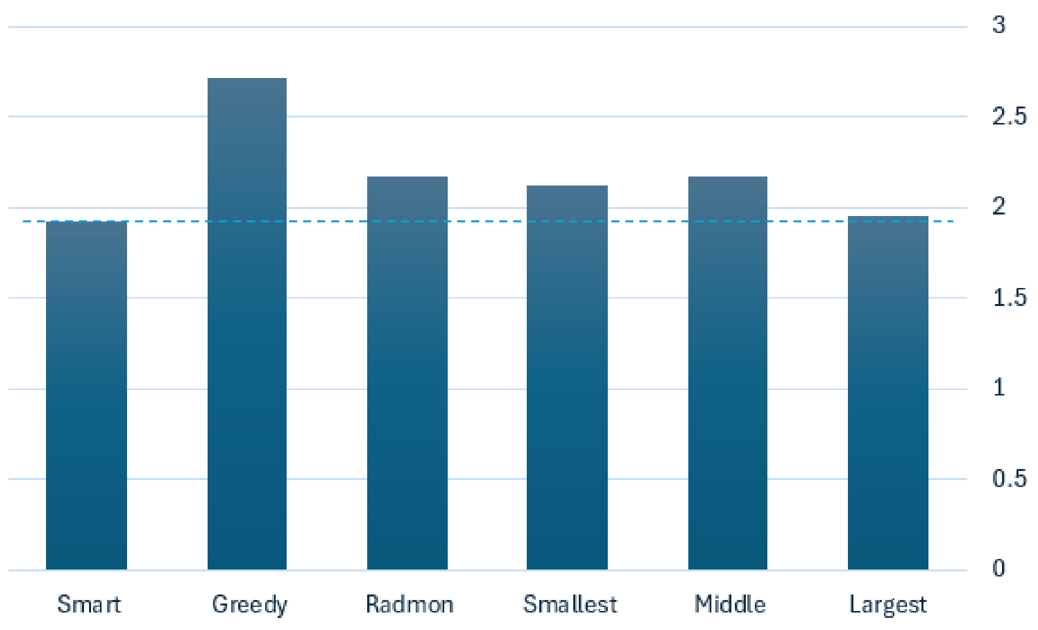
Comparing The Average Number of Iterations

It is important to notice that because of the way we implemented the task, the number of iterations equals the number of points that have weight 2 at the end.



As we can see, the smart way of choosing a point is the most efficient way, together with the largest way. We will see on the next chart the difference between the two.
The greedy way is a close second, showing it does its job well.
Smallest method is third. The middle and the random way are bad due to the fact that they are choosing a point with no idea behind it.

Comparing The Average Run Time (In Seconds)



As we can see, the smart way of choosing a point is the most efficient way, beating the largest by 0.028523733 seconds. Even though it may seem negligible, we should remember these are times for 500 points. Meaning, for 10,000 the time difference will be much more significant.

Another interesting thing to notice is that although the greedy way was efficient in the number of iterations, it is doing the worst on running time. It can be related to the fact that we need to do heavy calculations on each iteration.

How To Run the Project

Make sure to $pip\ install\ pandas$, $numpy$ and $openpyxl$ before running the project.

Make sure the excel file is in the same directory as the project file.

In the function: $if\ _name_ == $ "$\_main\_$": make sure to have $input\_file$ as your file name. If you want to overwrite the input file with the output file, meaning the result will be in the same file as the input, make sure to name your output file the same as the input file.