Ben Gurion University of the Negev
Gilad Shwartzman, Itay Vagen, Yonatan Browner, Shahar Davidovici, Inbal Melamed
Computer Science department

Stochastic optimization

14th March 2024

OVERVIEW

In our exploration of two-dimensional space, we are presented with a collection of points, each uniquely defined by its x and y coordinates and assigned an initial weight of one. While identifying the longest path of progression through these points—where each point along the path strictly increases in both coordinates from its predecessor—forms a part of our challenge, our central focus shifts towards a more nuanced objective. The core of our endeavor is to increase the weights of all points, enhancing their individual significance within the dataset, while ensuring that the longest path, denoted as W, retains its status as the heaviest among all paths.

GOALS

The project is driven by two main objectives:

1. **Identification of the Longest Path:** To determine the longest or heaviest path within the collection of points, ensuring that each step along this path adheres to a strict mathematical increase in both dimensions.
2. **Optimization of Point Weights:** Once the longest path is established, the goal shifts towards maximizing the weights of all points in the input set. This optimization seeks to elevate the significance of each point under the constraint that the longest path identified retains its status as the most significant sequence, thereby enriching the dataset's overall depth and complexity.

SPECIFICATIONS

The process unfolds through a designed algorithm that adheres to the following specifications:

- ☐ **Strict Increase Requirement:** For a path to be considered, it must exclusively consist of points where each x, y pair is greater than the x', y' pair of the preceding point, establishing a clear direction of progression.

- [ ] **Dynamic Path Length Calculation:** The algorithm dynamically calculates the maximum length of a path that any given point x, y is part of. This calculation is pivotal in understanding each point's role within the network of paths and sets the stage for subsequent weight optimization.
- [ ] **Iterative Weight Recalculation:** Once the maximum path length L through each point is determined, the algorithm iteratively recalculates the weight of each point using the formula W/L, where W is the weight of the heaviest path. This formula ensures that the new weight is inversely proportional to the point's participation in the longest paths, allowing for a nuanced enhancement of point weights across the dataset.
- [ ] **Optimization Protocol:** The final phase of the algorithm when recalculating the weights of each point is ensuring that every point is as significant as possible while upholding the identified longest path as the pinnacle of the set's structural hierarchy.

DYNAMIC PATH LENGTH CALCULATION

The logic step by step:

1. **Sorting the Points:**

   The points are first sorted by their x-values and then by y-values. This is done to ensure that when iterating over the points to form chains, each point considered later in the sequence has a higher or equal x-value (and y-value if x-values are equal) compared to the points considered earlier.

2. **Calculating Longest Chains (Forward Direction):**
   - A list ‘*lengths’* of the same length as ‘*sorted_points’* is initialized, with each element set to 1. This represents the longest chain ending at each point, starting with the assumption that each point can be a chain of length 1 by itself.
   - The nested loops compare each point i with all previous points j to find chains. If *‘sorted_points[i]’* is "greater" than *‘sorted_points[j]’* (as determined by the *‘is_greater’* function), and if the chain length ending at i can be extended by including j (i.e. ,$length[i] < length[j] + 1$), the length of the chain ending at i is updated.
3. **Calculating Longest Chains (Backward Direction):**

   A similar process is repeated in reverse order with a new list *‘lengths1’*. This time, for each point i, the function looks ahead to all points j that come after i and updates the chain lengths in reverse. This accounts for chains that might form in the opposite direction when considering the points in sorted order.

4. **Combining Results:**

   Finally, the function '*add_arrays(lengths1, lengths)*' combines the two lists of chain lengths calculated in forward and backward directions. Adding corresponding elements from each list to get the total length of chains that can be formed passing through each point.

OPTIMIZATION PROTOCOL

The optimization process in our algorithm involves recalculating the weight of each point after identifying the maximum path length, L, that goes through it. This is achieved by setting the new weight of a point as W/L, where W is the weight of the heaviest path found in the dataset. This method works because it proportionally increases the weight of each point relative to its involvement in the longest paths, thereby enhancing the overall structure without disturbing the hierarchical significance of W. By adjusting weights in this manner, points that contribute to shorter paths receive a smaller boost in weight compared to those on or near the heaviest path.

PROOF OF THE OPTIMIZATION PROTOCOL

To Prove:

That recalculating the weight of each point in a dataset as W/L, where W is the weight of the heaviest path and L is the maximum path length through a particular point, does not result in the creation of a new path that is heavier than W.

Proof:

Considering the nature of the problem, there are effectively two relevant scenarios for the paths in relation to W: paths shorter than W and paths exactly W. A "path longer than W" scenario does not exist within the constraints of our problem since W represents the maximum length of any path by definition.

1. Paths Shorter Than W:

   Paths shorter than W consist of points for which $L < W$, thus receiving a recalculated weight greater than 1 due to the W/L formula. While this increases individual point weights, the total recalculated weight of such paths remains constrained. This is because the increase in weight for each point is proportional to its relative distance from achieving W, ensuring the cumulative weight of these paths cannot exceed W. The distribution of recalculated weights respects the dataset's original hierarchy, preventing any shorter path from surpassing W's total weight.

2. Paths Exactly W:

   For paths exactly W, each point along the path has $W = L$, making their recalculated weight precisely 1. Consequently, the total recalculated weight of these paths equals W, affirming their position as the heaviest paths. The recalibration process does not augment the weight of these paths, ensuring their preeminence within the dataset is preserved.

Conclusion:

By recalculating point weights as W/L, the algorithm effectively enhances the dataset's granularity without altering its structural hierarchy. There is no scenario in which a path can become "longer than W" within the context of this problem, as W encapsulates the maximum achievable path weight by definition.

TIME COMPLEXITY: $O(n^2)$

The overall time complexity of the main algorithm, primarily driven by the longest_chain_lengths function, is $O(n^2)$. This quadratic time complexity is due to the nested loop structure used for calculating the longest chains. The sorting operation and other array manipulations contribute lower-order terms when considering the algorithm's overall time complexity.