Quick Hull Algorithm Overview

October 28, 2024

Quick hull algorithm based on divide and conquer numerical probelm

The Quick Hull algorithm is a classic divide-and-conquer approach to find the convex hull of a set of points. The convex hull is the smallest convex polygon that can contain all points. Quick Hull works similarly to Quick Sort by recursively partitioning the points, leading to an efficient solution.

Here's a step-by-step breakdown:

Steps for the Quick Hull Algorithm

1. **Find Extremes**:
    - Start by finding the leftmost and rightmost points in the dataset, which will always be part of the convex hull.
2. **Partition Points**:
    - Use these two points to form a line. Partition the remaining points into two subsets:
        - Points that are above the line.
        - Points that are below the line.
3. **Recursive Partitioning**:
    - For each subset:
        - Find the farthest point from the line. This point, by definition, will be part of the convex hull.
        - Split the subset further into two, creating two new lines:
            - One from the leftmost point to the farthest point.
            - Another from the farthest point to the rightmost point.
        - Recursively apply the same steps on each of the resulting subsets until there are no more points left in the subsets.
4. **Combine the Results**:
    - The points that were identified during the recursion form the convex hull of the original set.

Implementation of Quick Hull Algorithm (Python)

Here is a Python implementation of the Quick Hull algorithm:

```python
def find_side(p1, p2, point):
    """Finds the position of a point relative to a line segment (p1, p2)."""
    val = (point[1] - p1[1]) * (p2[0] - p1[0]) - (p2[1] - p1[1]) * (point[0] - p1[0])
    return 1 if val > 0 else -1 if val < 0 else 0

def line_distance(p1, p2, point):
    """Calculates the distance of a point from a line segment (p1, p2)."""
    return abs((point[1] - p1[1]) * (p2[0] - p1[0]) - (p2[1] - p1[1]) * (point[0] - p1
[0]))

def quick_hull(points, p1, p2, side, hull_points):
    """Recursive function for finding points in the convex hull."""
    max_point = None
    max_distance = 0

    for point in points:
        dist = line_distance(p1, p2, point)
        if find_side(p1, p2, point) == side and dist > max_distance:
            max_point = point
            max_distance = dist

    if max_point:
        # Recursive call to partition the points
        quick_hull(points, p1, max_point, -find_side(p1, max_point, p2), hull_points)
        quick_hull(points, max_point, p2, -find_side(max_point, p2, p1), hull_points)
    else:
        # Add the line's endpoints to the hull if no further points exist
        if p1 not in hull_points:
            hull_points.append(p1)
        if p2 not in hull_points:
            hull_points.append(p2)

def get_convex_hull(points):
    """Main function to initiate the Quick Hull process."""
    if len(points) < 3:
        return points

    min_x = min(points, key=lambda point: point[0])
    max_x = max(points, key=lambda point: point[0])

    hull_points = []
    quick_hull(points, min_x, max_x, 1, hull_points)
    quick_hull(points, min_x, max_x, -1, hull_points)

    return hull_points

# Example usage
points = [(0, 3), (1, 1), (2, 2), (4, 4), (0, 0), (1, 2), (3, 1), (3, 3)]
convex_hull = get_convex_hull(points)
print("Convex Hull:", convex_hull)
```

Explanation

- `find_side` checks the relative position of a point to a line formed by two other points.
- `line_distance` computes the perpendicular distance from a point to a line.
- `quick_hull` is the recursive function that applies the divide-and-conquer strategy, finding the farthest point from a line and further partitioning the set.
- `get_convex_hull` is the main function that initiates the process by finding the points with minimum and maximum x-coordinates and calling `quick_hull` on each side.

Complexity

- **Time Complexity**: $O(n \log n)$ in average cases, due to the divide-and-conquer nature, but can degrade to $O(n^2)$ in the worst case for certain point configurations.
- **Space Complexity**: $O(h)$, where h is the number of points in the convex hull.

This algorithm is efficient for many practical applications and can handle a wide range of input sizes and distributions.

Numerical problem

Here's a numerical example to illustrate the Quick Hull algorithm on a set of points.

Example

Given points:

$$(0,3), (1,1), (2,2), (4,4), (0,0), (1,2), (3,1), (3,3)$$

Step-by-Step Solution

1. **Find Extremes**:
   - The leftmost point is $(0,3)$ and the rightmost point is $(4,4)$.
   - These points form a line that will be part of the convex hull.

2. **Partition Points**:
   - Divide the points into two subsets based on which side of the line $(0,3) - (4,4)$ they lie on:
     - **Above**: $(2,2), (0,0), (1,1)$
     - **Below**: $(3,1)$

3. **Recursive Partitioning on Each Subset**:
   - **Above Line**:
     - Find the farthest point from the line $(0,3) - (4,4)$. This is $(0,0)$.
     - Split this subset further using two lines:
       - $(0,3) - (0,0)$ with points $(1,1)$ and $(2,2)$ above it.
       - $(0,0) - (4,4)$ with no points above it.
     - Repeat the process:
       - For $(0,3) - (0,0)$, the farthest point is $(1,1)$, which is part of the hull.
       - For $(0,0) - (4,4)$, add endpoints to the hull since there are no points left.
   - **Below Line**:
     - Only point $(3,1)$ lies below the initial line, so it is part of the hull.

4. **Combine Results**:
   - Points in the convex hull are: $(0,3), (0,0), (1,1), (3,1), (4,4)$.

Convex Hull Solution

The convex hull for this set of points is:

$$(0, 3), (0, 0), (1, 1), (3, 1), (4, 4)$$

This forms the smallest convex polygon that can contain all the given points.

Calculations for the farthest point

To find the farthest point from a line segment (p_1, p_2), we need to calculate the perpendicular distance of each point from the line. This distance can be found using the line equation and is given by:

$$\text{Distance} = \frac{|(y_2 - y_1) \cdot x - (x_2 - x_1) \cdot y + x_2 \cdot y_1 - y_2 \cdot x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

Where:

- (x_1, y_1) and (x_2, y_2) are the coordinates of points p_1 and p_2 forming the line.
- (x, y) is the point from which we're calculating the distance to the line.

Step-by-Step Example

Let's use the example points from earlier:

$$(0, 3), (1, 1), (2, 2), (4, 4), (0, 0), (1, 2), (3, 1), (3, 3)$$

We have:

- $p_1 = (0, 3)$ (leftmost point)
- $p_2 = (4, 4)$ (rightmost point)

Let's calculate the perpendicular distance of each point from the line segment $(0, 3) - (4, 4)$:

1. Point $(1, 1)$:

$$\text{Distance} = \frac{|(4 - 3) \cdot 1 - (4 - 0) \cdot 1 + 4 \cdot 3 - 4 \cdot 0|}{\sqrt{(4 - 3)^2 + (4 - 0)^2}}$$

$$= \frac{|1 \cdot 1 - 4 \cdot 1 + 12|}{\sqrt{1^2 + 4^2}}$$

$$= \frac{|1 - 4 + 12|}{\sqrt{1 + 16}} = \frac{9}{\sqrt{17}} \approx 2.18$$

2. Point $(2, 2)$:

$$\text{Distance} = \frac{|(4-3)\cdot 2 - (4-0)\cdot 2 + 4\cdot 3 - 4\cdot 0|}{\sqrt{(4-3)^2 + (4-0)^2}}$$
$$= \frac{|2 - 8 + 12|}{\sqrt{17}} = \frac{6}{\sqrt{17}} \approx 1.46$$

3. Point $(0, 0)$:

$$\text{Distance} = \frac{|(4-3)\cdot 0 - (4-0)\cdot 0 + 4\cdot 3 - 4\cdot 0|}{\sqrt{17}}$$
$$= \frac{|0 - 0 + 12|}{\sqrt{17}} = \frac{12}{\sqrt{17}} \approx 2.91$$

Summary of Distances:

- $(1, 1)$: 2.18
- $(2, 2)$: 1.46
- $(0, 0)$: 2.91

Result

The farthest point from the line $(0, 3) - (4, 4)$ is $(0, 0)$ with a distance of approximately 2.91.

This process is repeated recursively to identify additional points that belong to the convex hull.