CS235 Projects

Subhash Suri

January 17, 2024

1 Red Blue Point Separation

Input will be a set of n red and m blue points. The goal is to compute a *low complexity* separator. This is a fundamental "classification" problem that arises in many applications including machine learning, but for this project we will limit our study to just two dimensional data.

The simplest form of separator one can imagine is a line (linear separator) but in many cases such a simple separation is not possible, and so one resorts to more complex shapes, such as a circle, or convex polygon, or a polyline with a few segments. Your task for this project is to explore efficient algorithms, and test them on a variety of data and separator models.

Project Goals

1. Explore a small number of interesting geometric separator models?
2. Explore a small number of interesting input models? I.e. how do you plan to generate the red and blue points? What makes them natural or interesting?
3. How do you propose to estimate the complexity of the optimal separation?
4. Explore the effects of "outliers," that is, does the separator complexity improve significantly if we allow a few points to be misclassified.
5. Are there simple ML tools that can serve as a benchmark?

2 Stable Geometric Roommates

This projects asks you to explore the geometric version of the classical "stable matching" problem, defined as follows. The input is a set of $3n$ points in the plane. We want to split the points into n groups of 3 each. We can think of each group as a set of roommates, and the underlying motivation is that roommates should be compatible.

We measure compatibility by the Euclidean distance between two points–the smaller the distance, the more compatible the pair, and larger the distance, the less compatible the pair. We measure the compatibility of three points p, q, r by a scoring function $score(p, q, r)$. Some example scores are: (1) sum of distances $|pq| + |qr| + |pr|$; (2) the maximum of the three distances; (3) the sum of the *squared* distances, etc.

Suppose we have an assignment A (a collection of matched triples). We say that (p, q, r) is a *blocking triple* if

$$score(p, q, r) \ < \ \min\{score(A(p)), score(A(q)), score(A(r))\},$$

where $A(p), A(q), A(r)$ are the groups to which p, q, r are assigned in A. In other words, the triple p, q, r will be happier together as roommates than they are with their current assignments in A.

An assignment A is called *stable* if there is no blocking triple. The goal of this project is to explore algorithms for computing stable roommates.

The following paper has some partial results and background: Geometric Stable Roommates, by Arkin, Bae, Efrat, Okamoto, Mitchell, and Polishchuk, in Information Processing Letter, 2009.

Project Goals

1. Design and implement an efficient algorithm that can *check* whether a given assignment is stable. Explore ideas for beating the naive time bound for this step.

2. Explore cases where the stable assignment problem is easy, and also where the stable assignment problem is hard.

3. Explore some greedy strategies for solving the problem *approximately*: the algorithm can remove some of the input points so that the rest can be stably assigned. Empirically evaluate the performance of your strategies.

4. Explore a few different scoring functions, and report their relative merits or complexity.

5. Any assignment is just a collection of triangles. One can, therefore, also explore if these triangles have any nice structure, or share properties with some of the triangulations studied in the course.

6. Explore the k-roommate generalization, where each group has k points instead of 3.

3 Embedding k-Nearest Neighbor Graphs

Suppose we are given a set of points P in some d-dim space. Its k-nearest neighbor graph, $kNN(P)$, is a graph whose vertices represent the points of P and each vertex v has k outgoing (directed) edges to the k points of P that are closest to v by Euclidean distance.

In this project, the goal is to explore the *inverse* problem: we are given a directed but unlabeled graph G, each of whose vertices has out-degree k, and we wish to compute a set of points P in d-dimensional space whose k-NN graph is G.

Project Goals

1. Explore this problem in small dimensions ($d = 2$ or even $d = 1$).
2. The 1-dimensional problem (where all the points must lie on the line) is not NP-hard but is not straightforward. So your task is to design and empirically evaluate algorithms that perform well in practice.
3. The problem is known to be NP-hard in $d = 2$ dimensions; it is hard to decide if a given graph can be realized as k-NN graph in two dimensions, in fact, even if $k = 1$. Can you find an algorithm/heuristic that works well for embedding a 1-NN graph in the plane?
4. Next generalize your heuristic for embedding k-NN graphs with $k > 1$ in two dimensions.
5. Explore necessary and/or sufficient conditions for which G is easy to embed. Or, which k-NN graphs are hard to embed.
6. If the 1-NN graph is not embeddable in 2D, we can attempt to find an embedding in 3 or higher dimensions. Propose and explore algorithms for finding the smallest dimension where you can always embed such a graph.

4 Stable Connectivity For Moving Points

The general goal in this project is to explore the problem of finding a single best geometric structure for moving point data. To make the goals concrete, we will focus on the *minimum spanning tree* problem, using the Euclidean distances.

The input will be a set of n points $p_1, p_2, \ldots, p_n$ in 2 (or higher) dimension. Each of the points is under linear motion—constant speed and direction. The speed and direction of different points can be different but we know all the information at the start. We can normalize the motion period to unit time interval $[0, 1]$, so the $t = 0$ gives the initial positions of the points and $t = 1$ gives the final positions.

Our goal is to construct a spanning tree for these points, using Euclidean distances as edge costs, which for example might serve as a communication network if we think of points as mobile robots. Unfortunately, a spanning tree that has small cost at $t = 0$ (initial positions) may become quite costly after the points have moved to different positions. Indeed, the set of edges (point pairs) that form the MST may need to be continuously updated with the motion.

To avoid this complication, the entity managing the mobile robots would prefer a build *one spanning tree* at the start and use it throughout the entire period of the motion. However,

there are many possible (exponentially many) spanning trees, and the question is to choose the best one. The first thing we should decide is what best means.

Our goal is to choose a spanning tree $MMST$, which we call Minimum Moving Spanning Tree, so that its *maximum cost during the motion is minimized.* More specifically, suppose we have a spanning tree T of the points. As the points move, the length of each of its edges will change, so at time t, its cost will be

$$cost(T,t) = \sum_{e \in T} len(e(t)),$$

namely the sum of its edge lengths at time instant t, where $e(t)$ is the distance between the endpoints of e at time t. Our goal is to find the tree T^* that minimizes

$$max_{t \in [0,1]} cost(T,t),$$

namely the largest cost the tree will ever achieve during the motion.

The following two papers give some background for this problem.

- "Spanning Tree, Matching, and TSP for Moving Points: Complexity and Regret," by Wachholz and Suri, Canadian Conference on Computational Geometry, 2023.
- "The minimum moving spanning tree problem," by Akitaya, Biniaz, Bose, Carufel, Maheshwari, da Silveira, and Smid. J. Graph Algorithms Applications, 2023.

In particular, these papers prove that the problem if NP-hard, even for points moving on a line (one dimensional case). However, it is possible to efficiently compute a 2-approximation.

Project Goals

1. Propose and implement good algorithms (or heuristics) and empirically evaluate their performance.
2. Implement the 2-approximation algorithm, and compare its results (both tree quality and runtime) with your own algorithms.
3. Explore the performance and tree quality over different motion models and various dimensions, starting with $d = 1$ and $d = 2$ but possibly also $d \geq 3$.
4. Explore other related connectivity problems such as clustering (k-center clusters) or matching.
5. In all cases, it is understood that you should also explore how your algorithm's runtime scales with the input size.