Heuristics Based Isomap
A study

Emanuele Mengoli, Samir Fernando Florido Poka

March 2024

Contents

1 Introduction

This project embarks on an exploratory journey into the field of dimensionality reduction, focusing in particular on the applications of the Isomap algorithm to Euclidean Distance Geometry Problems (EDGP). The origins of this project are rooted in the work of Tenenbaum et al. [1], where the Isomap algorithm was introduced as a novel method for nonlinear dimensionality reduction, adept at preserving the global geometry of data and revealing the intrinsic low-dimensional structure embedded in high-dimensional spaces. Extending this conceptual framework, Liberti and D'Ambrosio [2] further explored the potential of Isomap in the context of n Distance Geometry Problem (DGP), specifically EDGP, demonstrating its applicability in the efficient approximation of graph topologies. An endeavour that finds a significant degree of applicability, ranging from structural proteomics to wireless sensor networks and more.
This is the premise of our project. Specifically, it takes inspiration from [2], as we aim to present six variants of completion algorithms for a given distance matrix, designed for the application of Isomap to EDGP. This endeavour aims to analyse heuristics and mathematical programming variants that cope with nonlinear dimensionality reduction, in order to transform high-dimensional distance data into actionable, lower-dimensional insights.

2 Methodology

Our project begins with the generation of simple connected graphs, on which the six variants of completion algorithms are executed to obtain a complete distance matrix, which is $\mathbb{R}^{n\times n}$ dimensional. The Isomap

procedure is described in algorithm 1. The six graph completion algorithms correspond to a variant of step 3 of pseudocode 1, taken from [2].

Algorithm 1 ISOMAP Algorithm

Require: A set of points $X \subset \mathbb{R}^n$

1: Compute all pairwise distances for X to obtain distance matrix D
2: Select a subset of distances to form a simple connected weighted graph $G = (V, E, d)$ where $V \in \mathbb{R}^n$, $d : E \to \mathbb{R}_+$
3: Compute all shortest paths in G to produce the approximate distance matrix $\widetilde{D}$, where $\widetilde{D}_{ij}$ equals the shortest path distance from i to j
4: Derive the approximate Gram matrix $\widetilde{B} = -\frac{1}{2}J\widetilde{D}^2J$, where $J = I_n - \frac{1}{n}11^T$
5: Perform eigen decomposition on $\widetilde{B}$ to find Λ and P, such that $\widetilde{B} = P\Lambda P^T$
6: Replace any negative eigenvalues in Λ with zeros to obtain a positive semi-definite (PSD) matrix
7: If there are more than K positive eigenvalues, keep only the largest K ones to form $\widehat{\Lambda}$
8: get the K-dimensional realization $X = P^T\sqrt{\widehat{\Lambda}}$
9: **return** the lower-dimensional representation $X \subset \mathbb{R}^K$

2.1 Heuristics

As solving EDGP is hard, [2] proposed to solve Euclidean Distance Matrix Completion Problem (EDMCP), who is proved to be solved in a polynomial number of steps (Th.1 of [2]) on a Real RAM machine. Therefore, the proposed algorithms aim to solve EDMCPs.

2.1.1 Floyd Warshall

The Floyd Warshall algorithm computes the shortest path between all pairs of vertices in a weighted graph. The algorithm's procedure is outlined in pseudocode 2. Its time complexity is $O(|V|^3)$.

Algorithm 2 Floyd Warshall Algorithm

Require: $G = (V, E, d)$, $\tilde{D}$ initialized as $D \in \mathbb{R}^{n\times n}$ partial matrix
Ensure: completed distance matrix $\tilde{D} \in \mathbb{R}^{n\times n}$

for each $k \in V$ **do**
  for each $i \in V$ **do**
    for each $j \in V$ **do**
      if $i \neq k$ and $j \neq k$ and $i \neq j$ **then**
        if $\tilde{d}[i][j] > \tilde{d}[i][k] + \tilde{d}[k][j]$ **then**
          $\tilde{d}[i][j] \leftarrow \tilde{d}[i][k] + \tilde{d}[k][j]$
        end if
      end if
    end for
  end for
end for
return $\tilde{D}$

2.1.2 Push and Pull (PaP)

The PaP formulation allows to compute a realisation $x' \in \mathbb{R}^n$, on which $\tilde{D}$ can be determined via pairwise Euclidean distances. The PaP is a semi-definite programming (SDP) formulation:

$$\max_{x} \sum_{\{u,v\}\in E} \|x_u - x_v\|^2$$
$$s.t.$$
$$\|x_u - x_v\|^2 \leq d_{uv}^2 \quad \forall\{u,v\} \in E$$

This method involves maximizing the sum of edge distances (pushing) while adhering to constraints that limit the maximum value they can take (pulling).

2.1.3 Breadth-First Search (BFS)

A modified BFS algorithm is described in pseudocode 3. The goal of this algorithm is to generate a spatial embedding of the nodes of the graph in $\mathbb{R}^K$, thus allowing the computation of the all pairwise Euclidean distances to get the matrix $\tilde{D} \in \mathbb{R}^{n\times n}$. Its time complexity is $O(|V|)$, since each vertex enters once in the *nodes explored (Z)* set.

A critical step in this algorithm is sampling from the surface of a K-dimensional sphere, $\mathbb{S}^{K-1}(x_v, d_{vw})$, as suggested in [2]. Specifically, when a new vertex w is to be placed relative to an existing vertex v, the algorithm samples a point from the surface of a sphere centred at the position $P[v]$ with a radius equal to the weight of the edge connecting v and w (d_{vw}), see pseudocode 3.

The sampling operation is computed by scaling a normalised Gaussian generated vector by the radius distance and summing it with the realisation of the parent vertex to obtain the new realisation.

Algorithm 3 Modified Breadth-First Search, realization in $\mathbb{R}^K$

Require: $G(V, E, d)$
Ensure: Positions of nodes $P : V \to \mathbb{R}^K$, Z explored set
$r \leftarrow \underset{v\in V}{\operatorname{argmax}} \deg(v)$ ▷ Node with the highest degree
$P[r] \leftarrow 0$ ▷ Position r at origin in K-space
$Z \leftarrow \{r\}$
$Q \leftarrow$ FIFO queue initialized with r
while $|Q| > 0$ **do**
    $v \leftarrow Q$.dequeue()
    $x_v \leftarrow P[v]$
    for each $w \in I_v$ **do** ▷ I_v is the set of neighbours of v
        if $w \notin Z$ **then**
            $Z \leftarrow Z \cup \{w\}$
            $x_w \leftarrow$ sample from $\mathbb{S}^{K-1}(x_v, d_{vw})$ ▷ surface of a K-dimensional sphere
            $P[w] \leftarrow x_w$
            Q.enqueue(w)
        end if
    end for
end while

2.1.4 Slack/Surplus Variables (SSV)

An alternative DGP formulation involves introducing slack/surplus variables. This is achieved by minimizing the expression:

$$\begin{aligned} &\min_{x,s} \sum_{\{u,v\}\in E} s_{uv}^2 \\ &\quad s.t. \\ &\|x_u - x_v\|^2 = d_{uv}^2 + s_{uv} \quad \forall \{u,v\} \in E \end{aligned}$$

In this approach, slack variables are introduced to the objective function. These variables represent the difference between the computed edge distances and the target distances. The objective is to minimize the sum of squared slack variables while ensuring that the computed edge distances match the target distances.

2.1.5 Centers tracker

The Centre Tracker's completion algorithm identifies the first m vertices with the highest connectivity, designating them as centres. These centres are interconnected, thus ensuring maximal connectivity within distinct neighbourhoods. For non-centre end nodes belonging to different neighbourhoods, their distances are computed by summing the distances between each end node and its nearest centre, along with the distances between the centres themselves. This process generates the distance matrix, $\tilde{D}$, as outlined in algorithm 4. The algorithm operates with a time complexity of $O(|V| \times m)$, where $|V|$ represents the number of vertices.

Algorithm 4 Centers tracker

Require: $G = (V, E, d)$
Ensure: completed distance matrix $\tilde{D} \in \mathbb{R}^{n \times n}$
Initialize $\Gamma = [\,]$ ▷ List to track vertices attained
Initialize $C = [\,]$ ▷ List to store centers
while $|\Gamma| < |V|$ **do** ▷ Until all vertices are attained
  $r \leftarrow \underset{v \in V}{\operatorname{argmax}} \deg(v) \quad s.t \quad v \notin C$ ▷ Node with highest degree not in centers list
  C $\leftarrow C \cup \{r\}$ ▷ Append center to centers list
  for w $\in \{I_r \setminus \Gamma\}$ **do** ▷ For unattained neighbours (I_r) of the center
    $\Gamma \leftarrow \Gamma \cup \{w\}$ ▷ Mark neighbour as attained
  end for
end while
for all $\{i, j\} \in C \times C$ **do**
  if $d_{i,j}$ is not known **then**
    $d_{i,j} \leftarrow$ Dijkstra(i, j) ▷ Calculate centers distance using Dijkstra's algorithm
  end if
end for
for a, b in $V \times V$ **do**
  $\tilde{d}_{a,b} = d(a, \text{center}(a)) + d(\text{center}(a), \text{center}(b)) + d(\text{center}(b), b)$
end for
return $\tilde{D}$

2.1.6 Random walk

The random walk algorithm begins by arbitrarily selecting a vertex as the initial node. It then progresses through the graph, assessing the path distance to each reachable node from this starting point. The choice of the initial vertex is pivotal, given that it sets the trajectory for the subsequent path exploration. The algorithm's efficiency is sensitive to the graph's attributes, such as its size and degree of interconnectedness.

In densely woven graphs, for example, the algorithm might labor intensively to identify a non-redundant connection to another node, which can escalate the worst-case time complexity significantly. See algorithm 5 for implementation details.

Algorithm 5 Random walk

Require: $G = (V, E, d)$, its partial distance matrix $D \in \mathbb{R}^{n \times n}$
Ensure: A completed distance matrix $\tilde{D} \in \mathbb{R}^{n \times n}$

Initialize $C = [\,]$ ▷ List to track visited vertices
Choose a random vertex u as the starting point
$C \leftarrow C \cup \{u\}$ ▷ Append starting vertex to *seen* list
while $|C| < n$ **do**
    $v \leftarrow 0$
    $weight \leftarrow 0$
    $Q \leftarrow \{0, 1, \ldots, n-1\} \setminus \{u\}$ ▷ Set of all vertices excluding u
    for all $k \in Q$ **do**
        if $\tilde{D}[u, k] \neq 0$ **then**
            $Q \leftarrow Q \setminus \{k\}$ ▷ Remove reachable vertices from Q
            $v \leftarrow k$
        end if
    end for
    $weight \leftarrow \tilde{D}[u, v]$
    Initialize Z as an empty list ▷ *passed* list
    while $|Q| > 0$ **do**
        $v' \leftarrow v$
        for all $k \in Q$ **do**
            if $\tilde{D}[u, k] = 0$ and $\tilde{D}[v, k] \neq 0$ and $u \neq k$ and $k \notin Z$ **then**
                $Z \leftarrow [\,]$ ▷ Reset *passed* list
                $Q \leftarrow Q \setminus \{k\}$
                $v \leftarrow k$
                $\tilde{D}[u, k] \leftarrow weight + \tilde{D}[v, k]$
                $\tilde{D}[k, u] \leftarrow D[u, k]$
                $weight \leftarrow \tilde{D}[u, k]$
            end if
        end for
        if $v = v'$ **then**
            for all $k \in \{0, 1, \ldots, n-1\}$ **do**
                if $\tilde{D}[u, k] \neq 0$ and $k \notin Z$ **then**
                    $Z \leftarrow Z \cup \{k\}$
                    $v \leftarrow k$
                    $weight \leftarrow \tilde{D}[u, k]$
                    Break
                end if
            end for
        end if
    end while
    Select a new $u \notin C$ randomly ▷ Ensure u is not in *seen* list
    $C \leftarrow C \cup \{u\}$
end while
return $\tilde{D}$

3 Experimental Results

Our simulations construct graph configurations with $|V| = 15$. We use three distinct graph topologies to test the robustness of the algorithms in different scenarios.

The first topology is an Erdős-Rényi graph, known for its random nature, in which each edge is included with a probability of p. This is followed by the Barabási–Albert model, that generate a random topology in which new nodes are attached with ν edges, through the preferentially attachment mechanism (i.e. existing nodes with the highest degree are preferred). Finally, we consider a regular graph structure in which each vertex has the same degree d; this uniformity provides a predictable and stable framework for our simulation.

Type of Graph	**Parameters Value**
Erdős-Rényi	$p = 0.5$
Barabási–Albert	$\nu = 6$
Regular Graph	$d = 6$

Table 1: Graph Configuration Simulation Parameters

We evaluate the quality of the obtained realizations according to mean (MDE), largest distance error (LDE) and root mean squared deviation (RMSD), defined as follows:

$$\text{MDE}(x, G) = \frac{1}{|E|} \sum_{(i,j)\in E} |\|x_i - x_j\|_2 - d_{ij}| \tag{1}$$

$$\text{LDE}(x, G) = \max_{(i,j)\in E} |\|x_i - x_j\|_2 - d_{ij}| \tag{2}$$

$$\text{RMSD}(x, G) = \sqrt{\frac{1}{|E|} \sum_{(i,j)\in E} \left(\|x_i - x_j\|_2 - d_{ij}\right)^2} \tag{3}$$

We also considered CPU computation time (in seconds) as an additional dimension of comparison.

We ran 50 simulations for each type of graph topology, running all 6 proposed completion algorithms, thus collecting average scores for each of the above metrics. Results are presented in tables: 2,3,4.

Mesure	Floyd Warshall	Centers tracker	Random path	BFS	Push and Pull	SSV
MDE	9.2964e-02	4.8227e-01	1.5096e-01	6.1542e-01	2.7027e-01	4.9461e-01
LDE	4.1406e-01	1.3468e+00	5.3254e-01	1.7650e+00	9.1143e-01	1.6366e+00
RMSD	1.3050e-01	5.8962e-01	1.9522e-01	7.4056e-01	3.6633e-01	6.3820e-01
CPU Time (s)	3.0566e-03	1.2310e-03	1.1783e-03	1.4362e-03	1.8954e+01	2.4819e+01

Table 2: Results obtained with the Erdos Renyi graph generator

Mesure	Floyd Warshall	Centers tracker	Random path	BFS	Push and Pull	SSV
MDE	7.9072e-02	2.8877e-01	1.3562e-01	5.4168e-01	2.6711e-01	4.5196e-01
LSE	3.6684e-01	9.7308e-01	4.8669e-01	1.6521e+00	9.0719e-01	1.5129e+00
RMSD	1.1121e-01	3.7104e-01	1.7710e-01	6.8466e-01	3.6470e-01	5.9447e-01
CPU Time (s)	2.8084e-03	1.0771e-03	1.1072e-03	1.4786e-03	1.8056e+01	2.6427e+01

Table 3: Results obtained with the Barabasi Albert graph generator

Mesure	Floyd Warshall	Centers tracker	Random path	BFS	Push and Pull	SSV
MDE	3.9891e-01	1.1549e+00	3.2377e-01	6.5284e-01	3.6035e-01	5.7982e-01
LDE	1.1583e+00	3.1226e+00	8.6157e-01	1.8815e+00	9.4409e-01	1.6029e+00
RMSD	5.2543e-01	1.3895e+00	3.9362e-01	7.8684e-01	4.2411e-01	6.8544e-01
CPU Time (s)	3.0887e-03	1.1820e-03	1.1405e-03	1.5191e-03	2.1287e+01	2.4238e+01

Table 4: Results obtained with the random regular graph generator

From the results, it's evident that for both the Erdos Renyi and Barabasi Albert graph generators, the Floyd Warshall heuristic achieves the lowest scores for MDE, LDE, and RMSD. This indicates superior performance in preserving distances within the graph. However, it ranks fourth in terms of CPU time duration, suggesting higher computational complexity compared to the Centers Tracker heuristic or Random Path. Nevertheless, it's important to note that while the order of magnitude for MDE, LDE, and RMSD is similar across all heuristics, significant differences exist in CPU time. For example, Random Path and Push and Pull show minimal differences in distance scores but exhibit considerable differences in timing. This could be attributed to the use of a solver that incurs higher computational costs.

We also observe that for both graph generators, BFS and SSV have the highest scores for LDE. This could be attributed to the nature of these algorithms, which may not be optimized for preserving the largest distance error.

For the random regular graph generator, we also note that it is not Floyd Warshall that achieves the lowest scores, but rather Random Path and Push and Pull. This suggests that for this particular type of graph, Random Path and Push and Pull algorithms are more effective in preserving distances compared to Floyd Warshall. Additionally, we observe that the CPU time comparison between these two heuristics remains consistent: Push and Pull requires significantly more CPU time than Random Path. This substantial difference in CPU time usage could be attributed to the computational complexity of the Push and Pull algorithm, which might involve more intricate calculations or iterations compared to the Random Path algorithm.

A visual example of a generated graph is shown in Figure 1. Figure 2 provides a visual representation of a realisation obtained with the Isomap algorithm, exploiting the PaP formulation.

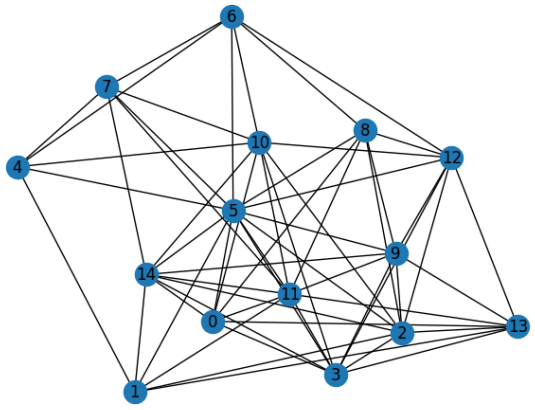

Figure 1: Graph obtained through Erdős-Rényi, $p = 0.5$

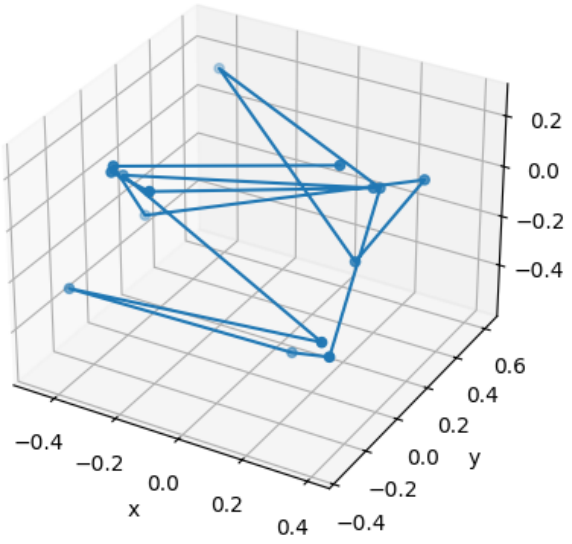

Figure 2: Isomap realization obtained with PaP with an inital graph of 15 vertices

4 Conclusion

This project was concerned with proposing six variants of completion algorithms for Isomap-based heuristics to solve EDGPs. Our results, emphasize the importance of considering both accuracy and computational efficiency when selecting heuristics for graph analysis, as well as the need for tailored approaches for different graph topologies. Further optimization and exploration of alternative algorithms may provide valuable avenues for improving performance in specific scenarios.

References

[1] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[2] L. Liberti and C. d'Ambrosio, "The isomap algorithm in distance geometry," in *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.