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 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. Further, we propose an application case such that, given a moving sensor network, whithin $\mathbb{R}^2$ space, we want to be able to infer successive Isomap instances by understanding node

motion. Specifically, we want to be able to infer how distances between nodes change, assuming that the velocity is drawn from a common distribution (shared by the nodes). In this sense, the goal is to directly infer Isomap instances so that the predictor outputs a $\mathbb{R}^{n\times k}$ matrix, where n is the number of nodes in the system and k is the Isomap dimensionality.

2 Methodology

Our project begins with the generation of biconnected graphs, on which the six variants of completion algorithms are executed to obtain a fully connected graph, whose distance matrix is ($\mathbb{R}^{n\times n}$) dimensional. The common procedure for obtaining an Isomap is described in the 1 algorithm. The six graph completion algorithms, which correspond to the first step of the algorithm 1, are described in detail in the subsection 2.1.

Algorithm 1 ISOMAP Pseudocode

Require: Simple connected weighted graph $G = (V, E, d)$

1: Complete graph G
2: Compute the distance matrix $\tilde{D} \in \mathbb{R}^{n\times n}$, where $\tilde{D}_{ij} = d_{ij}$, denoting the weight from node i to node j
3: Find the approcimate Gram matrix $\tilde{B}$ of $\tilde{D}$
4: Conduct Principal Component Analysis (PCA)
5: **return** $X \in \mathbb{R}^{n\times K}$

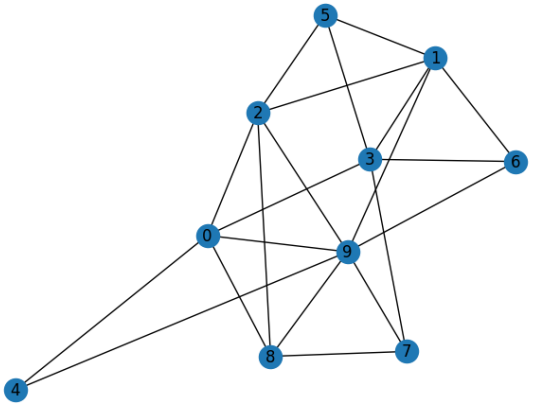
2.1 Heuristics

We can do the completion of the graph with algorithms based on the distances by browsing into the graph, but we also can do it by solving an EDMCP (Euclidiant distance matrix completion problem). Here is the SDP description of the EDMCP proposed by [2]:

$$\forall\{i,j\} \in E \quad X_{ii} + X_{jj} - 2X_{ij} = d_{ij}^2 \quad \text{with} \quad X \succeq 0. \quad (1)$$

To show graphically our Isomap output, we will use graph 4 as input.

Figure 1: Graph G(V $\in \mathbb{R}^{10}, E, d$)



2.1.1 Floyd Warshall

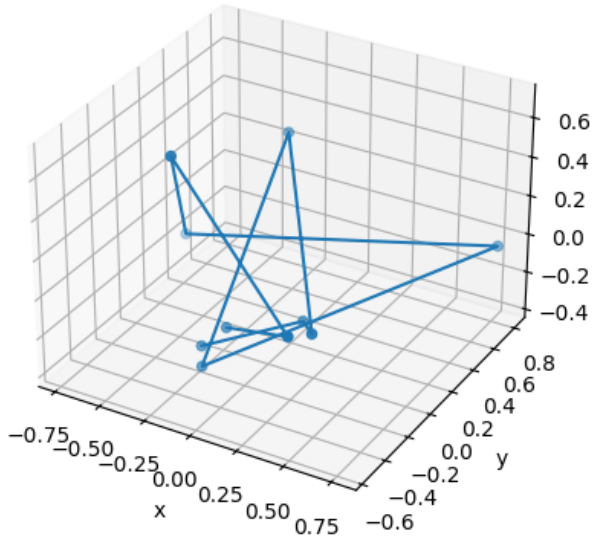
The Floyd Warshall algorithm computes the shortest path between all pairs of vertices in a weighted graph. The algorithm's procedure is outlined in Section 2. Its time complexity is $O(n^3)$. Once the distance matrix D is obtained using the Floyd Warshall algorithm, further analyses such as Principal Component Analysis (PCA) can be performed to obtain the Isomap representation.

Algorithm 2 Floyd Warshall Algorithm

Require: Simple connected weighted graph $G = (V \in \mathbb{R}^n, E, d)$

```
convert G to matrix D
for each i ∈ V do
    for each j ∈ V do
        for each k ∈ V do
            if i ≠ k and j ≠ k and i ≠ j then
                if D[i][j] > D[i][k] + D[k][j] then
                    D[i][j] ← D[i][k] + D[k][j]
                end if
            end if
        end for
    end for
end for
return D
```

Figure 2: Isomap obtained with Floyd Warshall completion algorithm



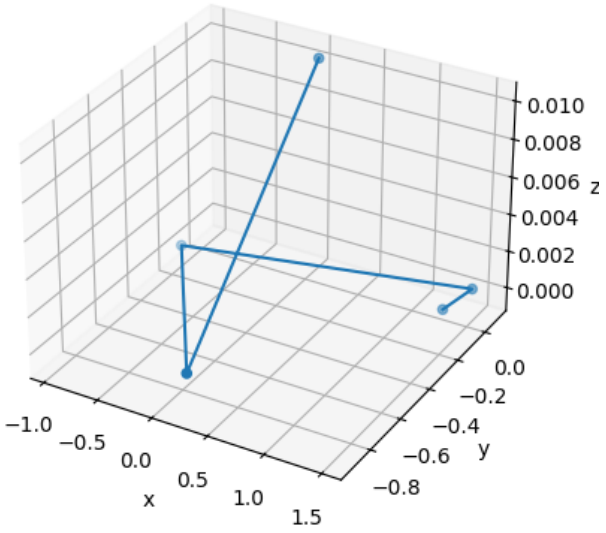
2.1.2 Push and Pull Formulation

Another approach to compute $\tilde{D}$ is by solving the following Semi-Definite Program (SDP):

$$\max_{P} \sum_{\{u,v\} \in E} \|x_u - x_v\|^2 \quad \text{subject to} \quad \forall \{u,v\} \in E \quad \|x_u - x_v\|^2 \leq d_{uv}^2$$

This method involves maximizing the sum of edge distances (pushing) while adhering to constraints that limit the maximum value they can take (pulling). We implemented this method in AMPL and tested it with numerous graphs generated by our random-graph-generator. Upon obtaining the results, we directly pass the matrix X obtained to step 3 of 1 by setting $\tilde{B} = X$.

Figure 3: Isomap obtained with Push and Pull SDP with an inital graph of 5 vertices



2.1.3 Breadth-First Search (BFS)

A modified BFS algorithm is described in the 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 a Euclidean distance matrix $D \in \mathbb{R}^{n \times n}$.

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 algorithm 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: A graph $G(V, E)$ with weights on edges $w : E \to \mathbb{R}^+$
Ensure: Positions of nodes $P : V \to \mathbb{R}^K$, distance matrix $D \in \mathbb{R}^{n \times n}$, 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 not empty **do**
    $v \leftarrow Q$.dequeue()
    $x_v \leftarrow P[v]$
    for each $w \in$ neighbors of v **do**
        if $w \notin Z$ **then**
            $Z \leftarrow Z \cup \{w\}$
            $d_{vw} \leftarrow w(v, w)$ ▷ Weight of edge (v, w)
            $x_w \leftarrow$ sample from sphere surface $\mathbb{S}^{K-1}(x_v, d_{vw})$.
            $P[w] \leftarrow x_w$
            Q.enqueue(w)
        end if
    end for
end while
for each pair $(u, v) \in V \times V$ **do**
    if $(u, v) \notin E$ and $u \neq v$ **then**
        $d_{uv} \leftarrow ||P[u] - P[v]||_2$ ▷ Euclidean distance
        Add edge (u, v) to E with weight d_{uv}
    end if
end for
$D \leftarrow$ Euclidian distances on P
return D

2.1.4 Slack/Surplus Variables (SSV)

Another technique for optimizing the objective function involves introducing slack/surplus variables. This is achieved by minimizing the expression:

$$\min_{P} \sum_{\{u,v\} \in E} s_{uv}^2 \quad \text{subject to} \quad \forall \{u, v\} \in E \quad \|x_u - x_v\|^2 = d_{uv}^2 + s_{uv}$$

In this approach, slack/surplus 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/surplus variables while ensuring that the computed edge distances match the target distances. We have implemented this method in AMPl an tested it through various graphs. Upon obtaining the results, we directly pass the matrix X obtained to step 3 of 1 by setting $\tilde{B} = X$.

2.1.5 K-means completion

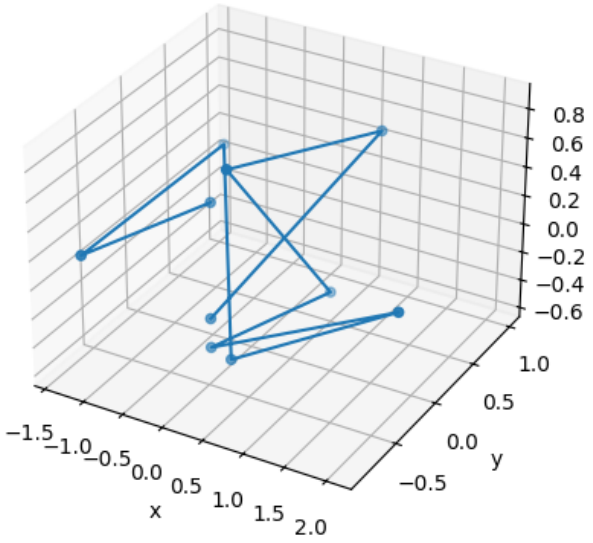
The K-means completion algorithm selects the first m most connected vertices, which can reach any other vertex, to serve as centroids. It then computes the distances between these centroids and establishes a distance matrix D,see algorithm 4. Its time complexity is $O(n \times m)$. Once the distance matrix D is obtained, further analyses such as Principal Component Analysis (PCA) can be performed to obtain the Isomap representation

Algorithm 4 K-means

Require: Simple connected weighted graph $G = (V \in \mathbb{R}^n, E, d)$

Convert graph G to distance matrix D
Initialize *vertices_attained* as an empty list ▷ List to track vertices attained
Initialize *centroids_list* as an empty list ▷ List to store centroids
while |vertices_attained| $< n$ **do** ▷ Until all vertices are attained
    $r \leftarrow \underset{v \in V}{\operatorname{argmax}} \deg(v) \setminus$ centroids_list ▷ Node with highest degree not in centroids list
    centroids_list $\leftarrow$ append r ▷ Add centroid to centroids list
    for neighbour $\in$ neighbours(r) $\setminus$ vertices_attained **do** ▷ For unattained neighbours of centroid
        vertices_attained $\leftarrow$ neighbour ▷ Mark neighbour as attained
    end for
end while
for (a, b) in V **do**
    $D_{a,b} = D(a, \text{centroid}(a)) + D(\text{centroid}(a), \text{centroid}(b)) + D(\text{centroid}(b), b)$
end for
return D

Figure 4: Isomap obtained with k-means completion algorithm



2.1.6 Random

The random algorithm selects a random vertex as the starting point, then it traverses the graph and computes the weight of the path followed to each reachable vertex. Its time complexity depends on the random choice of the first vertex and the structure of the graph see algorithm 5. The worst-case time complexity can be high if the graph is large and densely connected, as it may take a significant amount of time to find the connection with other vertex. Once the distance matrix D is obtained, further analyses such as Principal Component Analysis (PCA) can be performed to obtain the Isomap representation.

Algorithm 5 Random Path

Require: Simple connected weighted graph $G = (V, E, d)$
Convert graph G to distance matrix D
Choose a random vertex u as the starting point
Initialize *seen* as an empty list ▷ List to track visited vertices
seen.append(u) ▷ Add starting vertex to seen list
while len(*seen*) < size of D **do**
    $v \leftarrow 0$ ▷ Initialize vertex v
    $weight \leftarrow 0$ ▷ Initialize weight
    $Q \leftarrow$ List of all vertices
    Q.remove(u) ▷ Remove current vertex from Q
    for each vertex k in Q **do**
        if $D[u][k] \neq 0$ **then**
            Q.remove(k) ▷ Remove reachable vertices from Q
            $v \leftarrow k$ ▷ Update vertex v
            $weight \leftarrow D[u][v]$ ▷ Update weight
        end if
    end for
    while len(Q) > 0 **do**
        for each vertex k in Q **do**
            if $D[u][k] == 0$ and $D[v][k] \neq 0$ and $u \neq k$ **then**
                Q.remove(k) ▷ Remove reachable vertices from Q
                $v \leftarrow k$ ▷ Update vertex v
                $D[u][k] \leftarrow weight + D[v][k]$ ▷ Update distance
                $D[k][u] \leftarrow weight + D[v][k]$ ▷ Update symetrical distance
                $weight \leftarrow D[u][k]$ ▷ Update weight
            end if
        end for
    end while
    $u \leftarrow$ random vertex not in *seen* ▷ Choose a random unvisited vertex
    seen.append(u) ▷ Add the chosen vertex to seen list
end while
return D

In this algorithm, the choice of the first vertex and the exploration of random paths contribute significantly to its time complexity. Additionally, the efficiency of the algorithm can be influenced by factors such as the size and connectivity of the graph.

2.2 Real Case Application

On the application side, we designed a neural network, called `ISOPredictor` for the purpose of this section, that aims to understand the dynamics of graph movement, such as a moving sensor network, to map it to a real case scenario. The `ISOPredictor` is a neural network model designed to predict the changes in the geometric configuration of a graph as it undergoes translations in a given space. This task is critical for scenarios where the spatial arrangement of graph nodes is subject to variation over time, possibly due to external forces such as different imposed accelerations. The predictive model is trained on sequences of graph configurations represented by Isomaps, which encapsulate the distance relationships between nodes, thus facilitating the anticipation of future states of the graph based on learned motion patterns.

To achieve this, we first simulate the movement of a network graph, where the position of each node is updated according to stochastic translation. This simulation generates a series of graph states, which are then converted into isomaps. These isomaps serve as input to `ISOPredictor`, which preserves the dimensionality

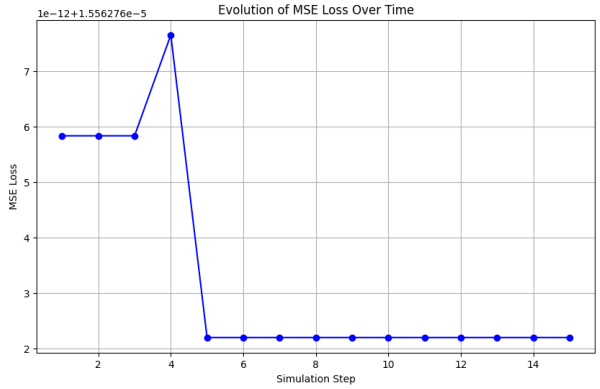
of the isomaps and outputs a predicted next state. An MSE loss is computed on the predicted and actual next states to enable network training.

Table 1: Neural Network Architecture of the `ISOPredictor`

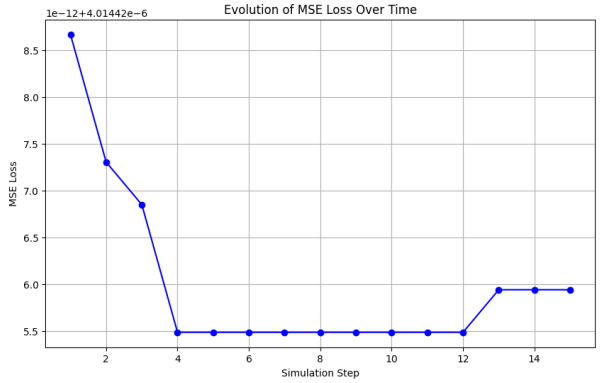
Layer	Output Dimensions	Number of Parameters
Input Layer	K	-
Hidden Layer	64	$64 \times (K+1)$
Leaky ReLU	64	-
Output Layer	K	$65 \times K$

Table 1 outlines the structure of the `ISOPredictor` model. The model begins with an input layer that matches the dimensionality of the Isomaps (K), followed by a series of transformations through a hidden layer, a non-linear activation function (Leaky ReLU), and another hidden layer that projects back to the original Isomap dimensionality.

3 Experimental Results



(a) MSE with constant normal movement $\mathbb{N}(\mu, \sigma)$ per step for all the network



(b) MSE with constant uniform movement $\mathbb{U}(-5, 5)$ per step for all the network

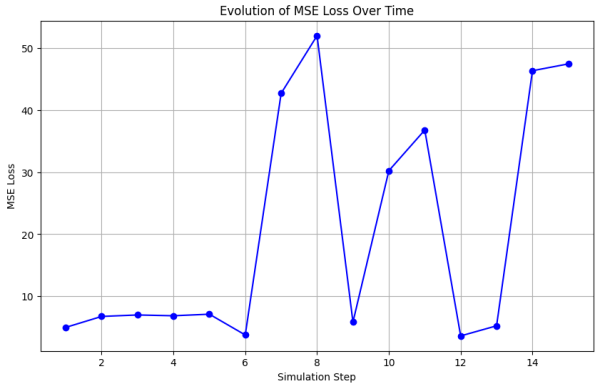
Figure 5: Comparison of MSE with constant movements

Considering Figures 5a, 5a, 6aand 6b, we can observe the capability of `ISOPredictor` in understanding network dynamics . Initially, `ISOPredictor` exhibits excellent performance in a straightforward scenario where all nodes move at the same speed per step. Each node's position $x_u \in \mathbb{R}^K, \forall u \in V$ undergoes a translation β such that β_i follows either a uniform distribution in the range $[-5, 5]$ $\mathbb{U}(-5, 5)$ or a normal distribution $\mathcal{N}(\mu = 2, \sigma = 2.5)$, $\forall i = 1, \ldots, K$. In this scenario, the Mean Squared Error (MSE) loss is approximately 10^{-5}, indicating excellent accuracy.

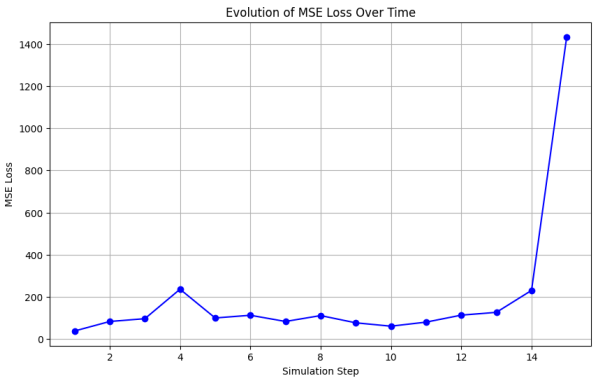
Increasing the complexity to a scenario where $\beta_{u,i} \sim \mathbb{U}(-5, 5), \forall i = 1, \ldots, K, \forall u \in V$ introduces heightened stochasticity in the network dynamics, with each node translating according to its independently sampled vector. In 6a, we observe that the MSE stays acceptable until the sixth simulation step, where it starts to diverge. This suggests that the network is able to adapt to moderate levels of randomness in node movements. However, as the randomness increases, the predictive accuracy diminishes.

Increasing the complexity further to a scenario where $\beta_{u,i} \sim \mathcal{N}(\mu = 2, \sigma = 2.5), \forall i = 1, \ldots, K, \forall u \in V$ introduces even greater stochasticity in the network dynamics. However, in this scenario, `ISOPredictor`

exhibits poor performance. This could be attributed to our neural network's lack of robustness in handling the increased randomness in node movements, which leads to a loss of predictive accuracy.



(a) MSE with random uniform movement $\mathbb{U}(-5, 5)$ per step for each element



(b) MSE with random normal movement $\mathbb{N}(\mu, \sigma)$ per step for each element

Figure 6: Comparison of MSE with random movements

4 Conclusion

This project was concerned with proposing six variants of completion algorithms for Isomap-based heuristics to solve the Euclidean Distance Geometry Problem. It extends the theory with an Isomap-based neural network predictor, which aims to learn the dynamics of a moving network by inferring time-consecutive Isomap instances. Further extensions could include directional constraints or simulating node failure to evaluate the flexibility of the predictor. Thus, testing on a large scale moving graph network would add another layer to this study. Finally, counter-testing on real-world applications such as logistics dynamics, e.g. truck coordination, would be an interesting extension.

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.

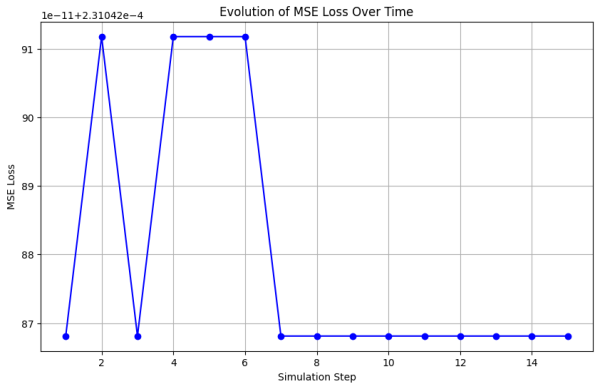
[2] L. Liberti and C. d'Ambrosio, "The isomap algorithm in distance geometry," in *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

5 Appendix

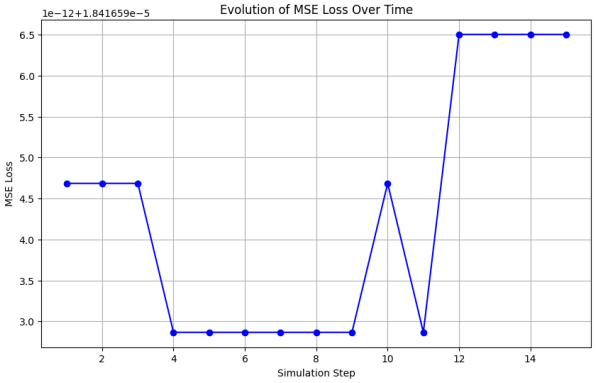
Here are the results obtained by the ISOPredictor when using different heuristics. Remarkably, regardless of the heuristic employed, the conclusions remain consistent. The predictive performance is commendable when the nodes undergo a constant movement at each step, yielding low Mean Squared Error (MSE) values. Conversely, when the movement becomes stochastic, with random translations applied to each element of the network at every step, the predictive capabilities diminish significantly, leading to substantially higher MSE values. This trend underscores the importance of considering the stability and predictability of node movements when employing predictive models like the ISOPredictor in real-world scenarios.

Algorithm	MSE Score with Constant Movement Uniform Per Step	MSE Score with Constant Movement Normal Per Step	MSE Score with Random Movement Uniform Per Element	MSE Score with Random Movement Normal Per Element
Floyd-Warshall	10^{-5}	10^{-6}	10^{1}	10^{3}
K-means	10^{-4}	10^{-5}	10^{2}	10^{3}
Random	10^{-6}	10^{-6}	10^{1}	10^{3}
BFS	10^{-1}	10^{-1}	10^{2}	10^{3}

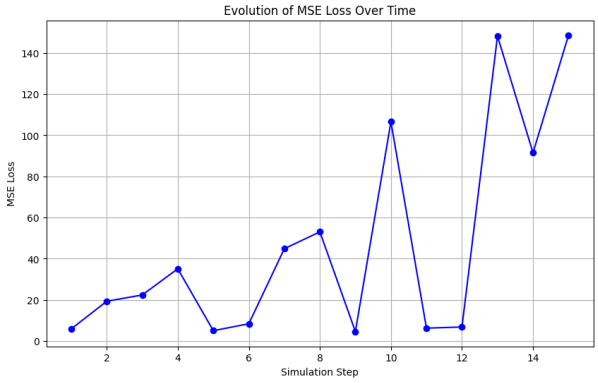
Table 2: MSE Scores for Different Algorithms per order of magnitude



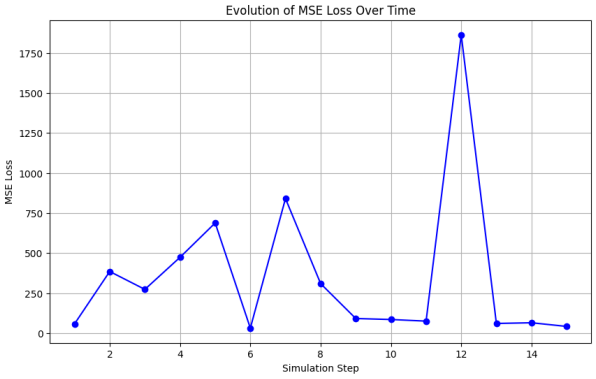
(a) MSE with constant normal movement $\mathbb{N}(\mu, \sigma)$ per step for all the network



(b) MSE with constant uniform movement $\mathbb{U}(-5, 5)$ per step for all the network



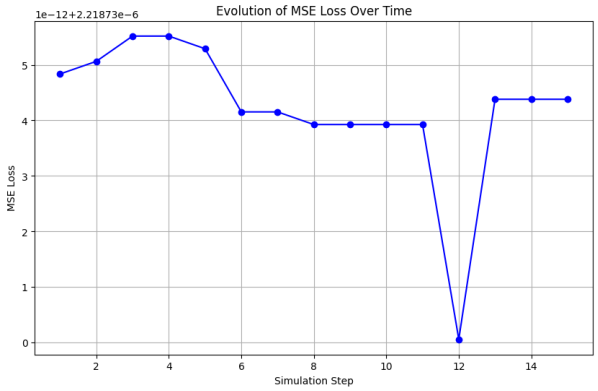
(c) MSE with random uniform movement $\mathbb{U}(-5, 5)$ per step for each element



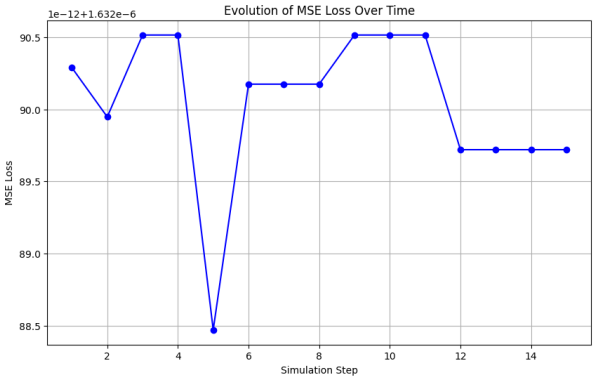
(d) MSE with random normal movement $\mathbb{N}(\mu, \sigma)$ per step for each element

Figure 7: Comparison of MSE with k-means completion algorithm

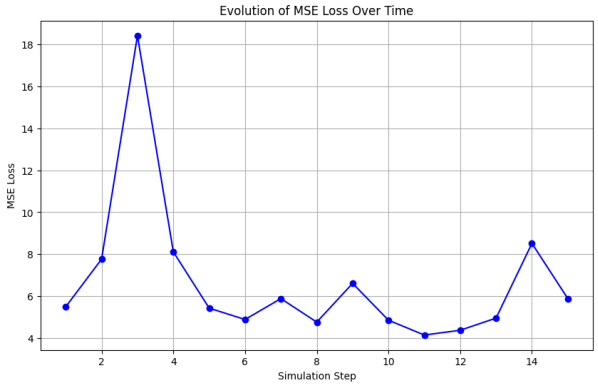
l



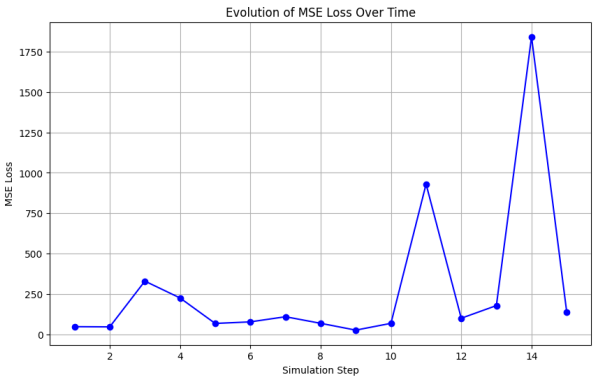
(a) MSE with constant normal movement $\mathbb{N}(\mu, \sigma)$ per step for all the network



(b) MSE with constant uniform movement $\mathbb{U}(-5, 5)$ per step for all the network

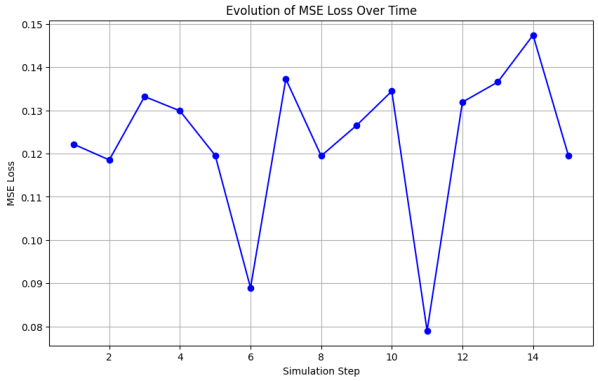


(c) MSE with random uniform movement $\mathbb{U}(-5, 5)$ per step for each element

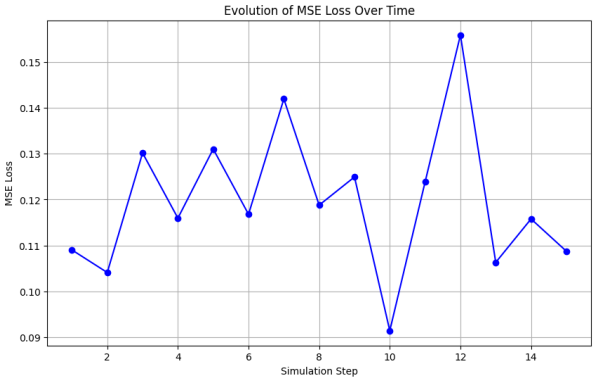


(d) MSE with random normal movement $\mathbb{N}(\mu, \sigma)$ per step for each element

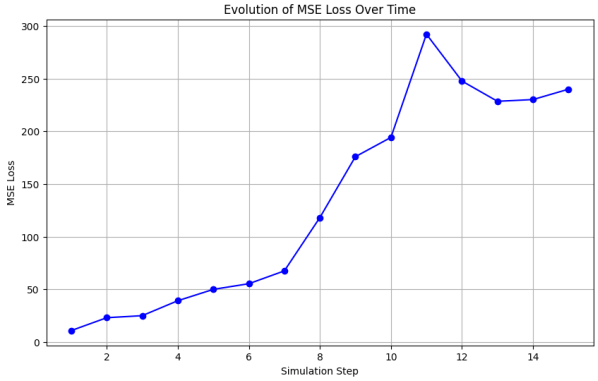
Figure 8: Comparison of MSE with the random completion algorithm



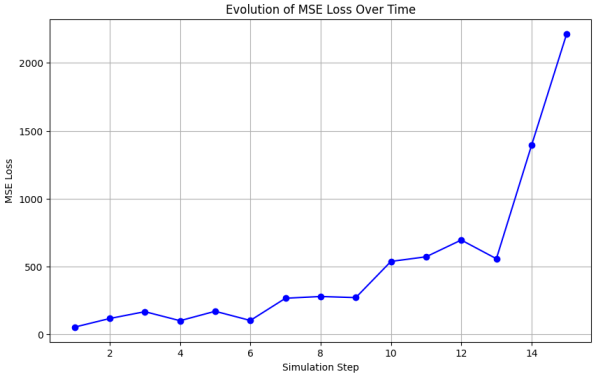
(a) MSE with constant normal movement $\mathbb{N}(\mu, \sigma)$ per step for all the network



(b) MSE with constant uniform movement $\mathbb{U}(-5, 5)$ per step for all the network



(c) MSE with random uniform movement $\mathbb{U}(-5, 5)$ per step for each element



(d) MSE with random normal movement $\mathbb{N}(\mu, \sigma)$ per step for each element

Figure 9: Comparison of MSE with the BFS completion algorithm