Application of Dynamic FastMap on Visualizing Real World Dynamic Social Networks

Ruijie Rao
Department of Computer Science
University of Southern California
ruijiera@usc.edu

Abstract

Visualizing data not only creates fascinating pieces of art but also greatly supports human-in-the-loop decision-making. Shapes and colors from static graphs as well as even motions from dynamic graphs can help the audiences to understand the dimensions within seconds. However, it becomes extremely harder for people to summarize the essence when it comes to a dynamic network, which involves tens and thousands of nodes and even more edges. With the help of Dynamic FastMap, which efficiently embeds dynamic graphs in Euclidean space, we are able to visualize the change over time in large-scale community networks. This will provide insights into community evolvement patterns and also open up space for geometric and analytical methods.

Introduction

In recent times, data visualization has grown to play a bigger and bigger part in business decision-making. This is because visualization brings advantages by increasing the amount of information delivered and decreasing the cognitive and intellectual burden to interpret information for decision-making (Park et al. 2021). Our brains are very capable of capturing important information from the shape, color, and motions of static and dynamic graphs, which greatly supports our interpretation of data. However, when it comes to dynamic networks, which are collections of ever-changing nodes and edges, we can no longer derive a meaningful interpretation of them by simply observing. Moreover, embedding these networks in a human interpretable space (like the Euclidian space) also allows us to solve graph-theoretic problems using geometric and analytical methods. Some examples of dynamic networks include social networks like Facebook friendship networks which could provide huge insights on relation recommendations and fraud detection with the right method.

In this paper, I will be exploring the possibility of applying Dynamic FastMap (Thakoor and Kumar 2022) on visualizing dynamic social networks. This algorithm is built based on a near-linear-time graph embedding algorithm called FastMap (Cohen et al. 2018; Li et al. 2019). FastMap embeds the vertices of a given undirected graph in Euclidean space while approximately preserving the shortest-path distances as Euclidean distances for all pairs of vertices. While FastMap itself is only applicable to static graphs, Dynamic FastMap is designed to embed dynamic graphs in Euclidean space with one dimension representing time and the other dimension representing pairwise distances between vertices of the graph at any timestep.

In order to meet the real-world context of social networks, we will also be using another derivation of FastMap: FastMap Block Modeling (Li et al. 2022). It achieves the purpose of cluster detection on graphs with high efficiency and solution quality by combining FastMap algorithm with Gaussian Mixture Models (GMMs). Every temporal cut of a dynamic social network will be embedded in Euclidean space using FMBM prior to Dynamic FastMap, and the cluster memberships provided will be used to generate real-world scenarios for evaluation purposes.

Related Works

This section will briefly explain how FastMap, FMBM, and Dynamic FastMap work and how they support the application process.

FastMap

The original FastMap (Faloutsos and Lin 1995) embeds a collection of abstract objects in an artificially created Euclidean space to enable geometric interpretations, algebraic manipulations, and downstream Machine Learning algorithms. It gets as input a collection of abstract objects O, where $D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in O$. A Euclidean embedding assigns a κ-dimensional point $p_i \in \mathbb{R}^{\kappa}$ to each object O_i . A good Euclidean embedding is one in which the Euclidean distance

χ_{ij} between any two points pi and pj closely approximates $D(Oi, Oj)$. For $p_i = ([p_i]_1, [p_i]_2 \dots [p_i]_\kappa)$ and $p_j = ([p_j]_1, [p_j]_2 \dots [p_j]_\kappa)$, $\chi_{ij} = \sqrt{\sum_{\{r=1\}}^{\kappa}([p_j]_r - [p_i]_r)}$. FastMap creates a κ-dimensional Euclidean embedding for a user-specified value of κ. In the very first iteration, FastMap heuristically identifies the farthest pair of objects O_a and O_b in linear time. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$ and $d_{ib} = D(O_i, O_b)$, as shown in Figure 1 (top panel). The sides of the triangle define its entire geometry, and the projection of O_i onto the line $O_a O_b$ is given by

$$x_i = \frac{d_{ai}^2 + d_{ab}^2 - d_{ib}^2}{2d_{ab}} .$$

FastMap sets the first coordinate of p_i, the embedding of O_i , to x_i. In the subsequent $\kappa - 1$ iterations, the same procedure is followed for computing the remaining $\kappa - 1$ coordinates of each object. However, the distance function is adapted for different iterations. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab}, respectively. Because these coordinates fully explain the true domain-specific distance between these two objects, from the second iteration onward, the rest of p_a and p_b's coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line $O_a O_b$. Although the hyperplane is never constructed explicitly, its conceptualization implies that the distance function for the second iteration should be changed for all i and j in the following way:

$$D_{new}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2 .$$

Here, O'_i , O'_j are the projections of O_i , O_j , respectively, onto this hyperplane, and $D_{new}(O_i, O_j)$ is the new distance function. FastMap can also be used to embed the vertices of a given graph $G = (V, E)$ in a Euclidean space so as to preserve the pairwise shortest-path distances between them. As such, the Data Mining FastMap algorithm cannot be directly used for generating a graph embedding in linear time, since it assumes that the distance d_{ij} between two objects O_i and O_j can be computed in "constant time", i.e., independent of the number of objects in the problem domain, whereas, computing the shortest-path distance between two vertices depends on the size of the graph. This challenge is tackled in FMBM to build a graph-based version of FastMap that runs in near-linear time. There, the key idea is to root shortest-path trees at vertices O_a and O_b, in each iteration, to yield all necessary distances d_{ai} and d_{ib} in one shot, achieving near-constant amortized time complexity.

FastMap with Block Modeling

A *block model* decomposes $G = (V, E)$ into a set of k vertex partitions representing the blocks (groups), for a given value of k. The partitions are represented by the membership matrix $C \in \{0,1\}^{n \times k}$, where $C_{ij} = 0$ and $C_{ij} = 1$ represent vertex v_i being absent from and being present in partition j, respectively. An image matrix is a matrix $M \in [0,1]^{k \times k}$, where M_{ij} represents the likelihood of an edge between a vertex in partition i and a vertex in partition j. The block model decomposition of G tries to approximate A by CMC^T with the best choice for C and M. In other words, the objective is:

$$\min_{C,M} \|A - CMC^T\|_F^2 ,$$

As mentioned before, FMBM works in two phases. In the first phase, it essentially implements FastMap as described in the previous subsection but calls the Single-Source Probabilistically-Amplified Shortest-Path Distance (SS-PASPD) instead of the regular single-source shortest-path distance function, which fails in relatively simple cases like bipartite graphs. SS-PASPD is thus created for pairs of vertices based on a few intuitive guidelines: (a) the smaller the shortest-path distance between v_i and v_j , the smaller the distance $D(v_i, v_j)$ should be; (b) the more paths exist between v_i and v_j , the smaller the distance $D(v_i, v_j)$ should be; and (c) the complement graph $\bar{G}$ of the given graph G should yield the same distance function as G. The new distance function $D_p(v_i, v_j)$ is defined as follows:

$$\sum_{\mathbb{G} \in G_{set}} d_{\mathbb{G}}(v_i, v_j)$$

Here, $d_{\mathbb{G}}(v_i, v_j)$ represents the shortest-path distance between vi and vj in an undirected graph G. G_{set} represents a collection of undirected graphs derived from the given graph G or its complement $\bar{G}$. In particular, each graph in G_{set} is an edge-induced subgraph of either G or $\bar{G}$. The

Algorithm 1: FastMap with Block Modeling

```
Input: $G = (V, E)$ and $k$
Parameters: $L$, $F$, $T$, $K$, $Q$, and $\epsilon$
Output: $c_i$ for each $v_i \in V$
$MinObj \leftarrow +\infty$ and $BestC \leftarrow \emptyset$.
for $t = 1, 2 \dots T$ do
    for $r = 1, 2 \dots K$ do
        Choose $v_a \in V$ randomly and let $v_b \leftarrow v_a$.
        for $q = 1, 2 \dots Q$ do
            $\{d_{ai} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_a$).
            $v_c \leftarrow \text{argmax}_{v_i}\{d_{ai}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_i]_j)^2\}$.
            if $v_c == v_b$ then
                Break.
            else
                $v_b \leftarrow v_a$ and $v_a \leftarrow v_c$.
            end if
        end for
        $\{d_{ai} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_a$).
        $\{d_{ib} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_b$).
        $d'_{ab} \leftarrow d_{ab}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_b]_j)^2$.
        if $d'_{ab} < \epsilon$ then
            Break.
        end if
        for each $v_i \in V$ do
            $d'_{ai} \leftarrow d_{ai}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_i]_j)^2$.
            $d'_{ib} \leftarrow d_{ib}^2 - \sum_{j=1}^{r-1}([\vec{p}_i]_j - [\vec{p}_b]_j)^2$.
            $[\vec{p}_i]_r \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$.
        end for
    end for
    $P \leftarrow [\vec{p}_1, \vec{p}_2 \dots \vec{p}_{|V|}]$.
    $C \leftarrow$ GMM($P$, $k$).
    $Obj \leftarrow$ GetObjectiveValue($G$, $C$).
    if $Obj \leq MinObj$ then
        $MinObj \leftarrow Obj$.
        $BestC \leftarrow C$.
    end if
end for
return $c_i$ for each $v_i \in V$ according to $BestC$.
```

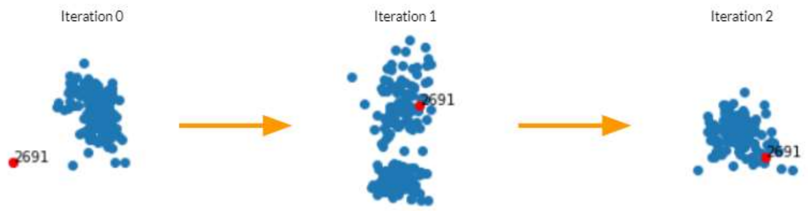

Figure 1: A FMBM embedding modified using patching in Scenario 1. Node 2691 is added at the beginning of the process with 2 edges, and it is further connected at each iteration.

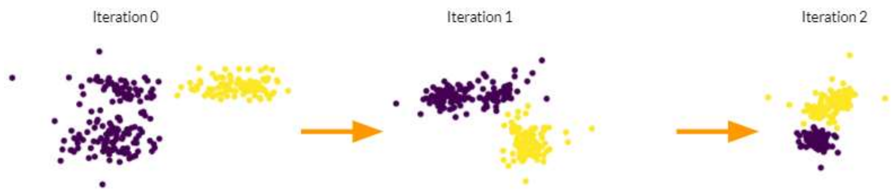

Figure 2: A FMBM embedding modified using patching in Scenario 2.

edge-induced subgraphs are created by probabilistically dropping edges from G or $\bar{G}$.

Dynamic FastMap

FastMap is very efficient on embedding most static graphs, but not dynamic graphs. Developed by Omkar Thakoor, Dynamic FastMap aims to make FastMap viable for dynamic graphs. It takes as input snapshots of a dynamic graph on N vertices. Each snapshot G_t , for $t = 0, 1, \ldots, T$, describes the graph at timestep t. Additionally, the algorithm takes as input κ, the number of spatial dimensions for the embedding, and ϵ, a threshold parameter required for invoking FastMap as a subroutine. The algorithm outputs the sequence of embeddings $Z_t = [z_1^t, z_2^t, \ldots, z_N^t]$, where each $z_i^t \in \mathbb{R}^\kappa$ denotes the κ-dimensional point assigned to vertex i at timestep t, for $i = 1, 2, \ldots, N$ and $t = 0, 1, \ldots, T$.

Iterating through the timesteps $t = 0, 1, \ldots, T$, each graph G_t is embedded as Z_t through 2 passes: FastMap and patching. In the first pass, it invokes FastMap on G_t to obtain an embedding $X_t = [x_1^t, x_2^t, \ldots, x_N^t]$. In the second pass, it modifies X_t to Z_t by patching X_t relative to the previous embedding Z_{t-1}.

Patching is a process that intends to obtain a smooth transition of the points between the embeddings Z_{t-1} and Z_t by preserving all the pairwise distances between its points and minimizing the difference between them. This is accomplished by finding the optimized isometric operation which is represented by $A^t x_i^t + b^t$, where A^t is the orthogonal transformation and b^t is the translation. The optimization problem is as follows:

$$\min_{A^t, b^t} \sum_{i=1}^{N} \left\| A^t x_i^t + b^t - z_i^{t-1} \right\|^2$$
$$s.t \|A_i^t\|^2 = 1 \; \forall 1 \leq i \leq \kappa$$
$$A_i^t \cdot A_j^t = 0 \; \forall 1 \leq i \leq j \leq \kappa$$

This can be solved by using a state-of-the-art optimizer such as Gurobi.

Application on Dynamic FastMap

In this application project, I will be using a real-world Facebook social network from 2012 found in Stanford Network Analysis Project (SNAP) provided by McAuley and Leskovec in their paper. The entire undirected network includes 4039 nodes and 88034 edges. For the purpose of efficient testing, I will be randomly slicing out a 200-node-large subnet from the original network as test network.

The entire application process is separated into 3 phases: create real-world scenarios, embed test networks using FMBM, and modify the networks using Dynamic FastMap. Every person in a social network act with their own intentions. Basic social actions include connecting and disconnecting with another person. More complex social intentions are composed of basic social actions with specific targets and action frequencies. For example, a normal Facebook user may be adding people at a regular but slow pace, but a fraud account may add a specific community (like elderlies) by a large amount at a very frequent pace. As a result, I claim that the application of Dynamic FastMap on real-world social networks would be successful if observers can detect social actions and intentions by observing its generated temporal graphs through specifically designed scenarios.

Scenario 1: New Facebook Account

In this scenario, there will be a new account that just joined Facebook. As a new user, the person will be eagerly adding all his/her related people at a fast pace. Thus, I will be inserting a new node into the network. The new node will have

2 random edges attached. Then, for every iteration, the node will be connected to one-third of its unconnected neighbors. Each iteration t generates a graph G_t, embedded using FMBM as X_t, then patched using Dynamic FastMap as Z_t, which is shown in figure 1. It is observable that the node is isolated from the community in iteration 0. In iteration 1, the node has joined with part of the community and has joined the whole community in iteration 2. It is proper to conclude that through this 3-iteration scenario, the designed social intention is observable.

Scenario 2:

In this scenario, there will be two community groups trying to merge with each other. Since community merging happens very frequently among groups of different sizes, I believe this is a very good case to study. The original network at iteration 0 is generated by linking 2 randomly sliced networks of size 100. FMBM successfully identified the nodes' community membership with an accuracy of 97.5%. At every iteration, I added 30 edges between the two communities. In Figure 2, we can clearly detect 2 separated community groups. It is also observable that as iterations proceed, two communities become closer to each other. It is clear for us to detect a community merging action.

Applying FMBM

With the original code provided by Ang Li, FMBM is able to be successfully ran on the test network using hyperparameters: $L = 4, F = 10, T = 10, K = 3, Q = 10$, and $\epsilon = 10^{-4}$. The number of clusters k is set to be 2 at default and adjusted according to the graph's Euclidean embedding. There have been frequent occasions when the algorithm terminate itself early with reduced dimensions due to new $d_{ab} \leq \epsilon$. Meanwhile, the efficiency of embedding can be further improved.

Applying Dynamic FastMap

The original FastMap embedding process in Dynamic FastMap is replaced by FMBM as mentioned above. Every X_t other than Z_0 is modified with the "patching" process. A^t and b^t are found using Gurobi optimizer in python for every X_t.

Further Work

First of all, more work needs to be done on finding the optimal hyperparameters for applying FMBM on community networks. This is because currently there have been inconsistencies during the embedding and the efficiency can be further improved. Meanwhile, I would like to try the weighted version of this application and further apply other graph analytical tools on the embedded results.

References

Park, S.; Bekemeier, B.; Flaxman, A. and Schultz, M 2022. Impact of data visualization on decision-making and its implications for public health practice: a systematic literature review, *Informatics for Health and Social Care*, 47:2, 175-
193, DOI: 10.1080/17538157.2021.1982949

Thakoor, O. and Kumar, T. K. S. 2022. Dynamic FastMap: An Efficient Algorithm for Spatiotemporal Embedding of Dynamic Graphs

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. K. S. 2018. The FastMap algorithm for shortest path computations. In Proceedings of the 27th International Joint Conference on Artificial Intelligence

Ang Li, Peter Stuckey, Sven Koenig, and T. K. Satish Kumar. 2022. A FastMap-Based Algorithm for Block Modeling. *In Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 19th International Conference,* CPAIOR 2022, Los Angeles, CA, USA, June 20-23, 2022, Proceedings. Springer-Verlag, Berlin, Heidelberg, 232–248. https://doi.org/10.1007/978-3-031-08011-1_1

J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.