Embedding Directed Graphs in Potential Fields Using FastMap-D

Sriram Gopalakrishnan\textsuperscript{1} and Liron Cohen\textsuperscript{2} and Sven Koenig\textsuperscript{2} and T. K. Satish Kumar\textsuperscript{2}

\textsuperscript{1}Arizona State University
\textsuperscript{2}University of Southern California

sgopal28@asu.edu, \{lironcoh, skoenig\}@usc.edu, tkskwork@gmail.com

Abstract

Embedding undirected graphs in a Euclidean space has many computational benefits. FastMap is an efficient embedding algorithm that facilitates a geometric interpretation of problems posed on undirected graphs. However, Euclidean distances are inherently symmetric and, thus, Euclidean embeddings cannot be used for directed graphs. In this paper, we present FastMap-D, an efficient generalization of FastMap to directed graphs. FastMap-D embeds vertices using a potential field to capture the asymmetry between the pairwise distances in directed graphs. FastMap-D learns a potential function to define the potential field using a machine learning module. In experiments on various kinds of directed graphs, we demonstrate the advantage of FastMap-D over other approaches.

Introduction

Graph embeddings have been studied in multiple research communities. For example, in Artificial Intelligence (AI), they are used for shortest path computations (Cohen et al. 2018) and solving multi-agent meeting problems (Li et al. 2019). In Knowledge Graphs, they are used for entity resolution (Bordes et al. 2013); and in Social Network Analysis, they are used for encoding community structures (Perozzi, Al-Rfou, and Skiena 2014). In general, graph embeddings are useful because they facilitate geometric interpretations and algebraic manipulations in vector spaces. Such manipulations in turn yield interpretable results in the original problem domain, such as in question-answering systems (Bordes, Weston, and Usunier 2014).

Despite the existence of many graph embedding techniques, there are only a few that work in linear or near-linear time\textsuperscript{1}. FastMap (Cohen et al. 2018; Li et al. 2019) is a near-linear-time algorithm that embeds undirected graphs in a Euclidean space with a user-specified number of dimensions. The efficiency of FastMap makes it applicable to very large graphs and to dynamic graphs such as traffic networks and marine environments for unmanned surface vehicles.

The resulting Euclidean embedding can be used in a variety of contexts. The $L_1$-variant of FastMap (Cohen et al. 2018) produces an embedding useful for shortest path computations. Here, the $L_1$ distances between the points corresponding to pairs of vertices are used as heuristic distances between them in the original graph; and this $L_1$ distance function is provably admissible and consistent, thereby enabling A* to produce optimal solutions without re-expansions. The $L_2$-variant of FastMap (Li et al. 2019) produces an embedding that is generally useful for geometric interpretations. In the multi-agent meeting problem, for example, the problem is first analytically solved in the Euclidean space and then projected back to the original graph using Locality Sensitive Hashing (LSH) (Datar et al. 2004).

In general, the properties of the Euclidean space can be leveraged in many ways. For example, a Euclidean space is a metric space in which the triangle inequality holds for distances. In addition, in a Euclidean space, geometric objects, like straight lines, angles and bisectors, are well defined. The ability to conceptualize these objects facilitates visual intuition and can help in the design of efficient algorithms for Euclidean interpretations of graph problems.

Despite the usefulness of Euclidean embeddings, Euclidean distances are inherently symmetric and, thus, cannot be used for directed graphs. Directed graphs arise in many real-world applications where the relations between entities are asymmetric, such as in temporal networks and social networks. In this paper, we present FastMap-D, an efficient generalization of FastMap to directed graphs. FastMap-D embeds vertices using a potential field to capture the asymmetry between the pairwise distances in directed graphs. Like the $L_2$-variant of FastMap, FastMap-D focuses on minimizing the distortion between pairwise distances in the potential field and the corresponding true distances in the directed graph. FastMap-D therefore provides physical interpretations of problems posed on directed graphs by enabling vector arithmetic in potential fields. It is a back-end algorithm meant to support a number of applications, including question-answering, machine learning and multi-agent tasks on directed graphs.\textsuperscript{2}

The difference in potential of two points in a potential

\textsuperscript{1}Linear time after ignoring logarithmic factors

\textsuperscript{2}Shortest path computation is just one more application that requires FastMap-D to also consider the properties of admissibility and consistency if the intended search framework is related to A*.
field is inherently asymmetric and therefore a good choice for capturing distances in a directed graph. FastMap-D constructs a potential function defining the potential field using a machine learning module. Through experiments conducted on various kinds of directed graphs, we demonstrate the advantage of FastMap-D over other approaches, including those that directly apply machine learning techniques to learning pairwise distances between the vertices.

Related Work

There are a variety of approaches that embed undirected graphs in a Euclidean space. While some approaches simply try to preserve the pairwise distances between vertices in the embedding, other approaches try to meet additional constraints. For example, (Linial, London, and Rabinovich 1995) surveys several methods for the low-distortion embedding of undirected graphs and their usage in algorithmic applications such as clustering. On the other hand, the Euclidean Heuristic Optimization (EHO) (Rayner, Bowling, and Sturtevant 2011) and the $L_1$-variant of FastMap (Cohen et al. 2018) meet additional constraints on the pairwise distances in the embedding. In particular, these distances satisfy admissibility and consistency, which are useful for shortest path computations with heuristic search.

Global Network Positioning (Ng and Zhang 2002) first embeds landmarks in a Euclidean space and then uses them for a frame of reference. Like EHO, this algorithm relies on solving Semi-Definite Programs (SDPs) or similar approaches that are prohibitively expensive for large graphs. Big-Bang Simulation (BBS) (Shavitt and Tankel 2004) is a different method that simulates an explosion of particles under a force field derived from the embedding error. Although it does not rely on solving SDPs, it is still prohibitively expensive for large graphs.

Existing works on embedding directed graphs, such as Node2Vec (Grover and Leskovec 2016), LINE (Tang et al. 2015) and APP (Zhou et al. 2017), focus on preserving the proximity of vertices. First-order proximity refers to the distance between two vertices; second-order proximity is related to the similarity of their one-hop neighboring vertices; third-order proximity is related to the similarity of their two-hop neighboring vertices; and so forth. These proximity-preserving embedding algorithms are based on skip-gram models originally developed in the context of Natural Language Processing (Mikolov et al. 2013). Before training, they generate samples of vertex neighborhoods via parameterized random walks. To represent asymmetric proximities, these techniques use two points for each vertex, one to represent the vertex as a source and the other to represent it as a destination. They are appropriate for link prediction, node labeling and community detection in social networks (Grover and Leskovec 2016). These algorithms differ from our approach in important ways. First, they lose the physical interpretation since they use two points for each vertex in directed graphs. Second, they are semi-supervised algorithms that require labels or data about vertex similarities while our approach is an unsupervised approach that does not require any information other than the given directed graph. Third, our approach has a plug-and-play ma-

![Figure 1](image_url)

Figure 1: (a) shows the “cosine law” projection in a triangle. (b) illustrates how coordinates are computed and recursion is carried out in FastMap.

Background

FastMap (Faloutsos and Lin 1995) was introduced in the Data Mining community for automatically generating Euclidean embeddings of abstract objects. For example, if we are given objects in the form of long DNA strings, multimedia datasets such as voice excerpts and images or medical datasets such as ECGs or MRIs, there is no geometric space in which these objects can be naturally visualized. However, there is often a well-defined distance function between each pair of objects. For example, the edit distance between two DNA strings is well defined although an individual DNA string cannot be conceptualized in geometric space. Clustering techniques, such as the $k$-means algorithm (Alpaydin 2010), are well studied in Machine Learning but cannot be applied directly to domains with abstract objects because they assume that objects are described as points in geometric space. FastMap revives their applicability by first creating a Euclidean embedding for the abstract objects that approximately preserves the pairwise distances between them.

In the Data Mining community, FastMap gets as input a complete non-negative edge-weighted undirected graph $G = (V, E, w)$. Each vertex $v_i \in V$ represents an abstract object $O_i$. Between any two vertices $v_i$ and $v_j$, there is an edge $(v_i, v_j) \in E$ with weight $D(O_i, O_j)$ that corresponds to the symmetric distance between objects $O_i$ and $O_j$. A Euclidean embedding assigns a $K$-dimensional point $p_i \in \mathbb{R}^K$

---

This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/marirsg2/FastMap-D
to each object $O_i$. A good Euclidean embedding is one in which the Euclidean distance between any two points $p_i$ and $p_j$ closely approximates $D(O_i, O_j)$.

FastMap creates a Euclidean embedding in linear time by first assuming the existence of a very high dimensional embedding and then carrying out dimensionality reduction to a user-specified number of dimensions. In principle, it works as follows: In the first iteration, it heuristically identifies the farthest pair of objects $O_a$ and $O_b$ in linear time. It does this by initially choosing a random object $O_a$ and then choosing $O_b$ to be the object farthest away from $O_a$. It then reassigns $O_b$ to be the object farthest away from $O_a$. 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 $d_{bi} = D(O_b, O_i)$. Figure 1(a) shows this triangle. The sides of the triangle define its entire geometry, and the projection of $O_i$ onto $O_aO_b$ is given by $x_i = (d_{ai}^2 + d_{ab}^2 - d_{bi}^2)/(2d_{ab})$.

FastMap sets the first coordinate of $p_i$, the embedding of object $O_i$, to $x_i$. In particular, the first coordinate of $p_a$ is 0 and of $p_b$ is $d_{ab}$. Computing the first coordinates of all objects takes only linear time since the distance between any two objects $O_i$ and $O_j$ for $i, j \not\in \{a, b\}$ is never computed.

In the subsequent $K - 1$ iterations, the same procedure is followed for computing the remaining $K - 1$ 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 distance $d_{ab}$ between them, from the second iteration onward, the remaining coordinates of $p_a$ and $p_b$ should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to $O_aO_b$.

Figure 1(b) explains this intuition. Although the hyperplane is never constructed explicitly, its conceptualization implies that the distance function for the second iteration should be changed by $D_{new}(O_i', O_j') = D(O_i, O_j)^2 - (x_i - x_j)^2$. Here, $O_i'$ and $O_j'$ are the projections of $O_i$ and $O_j$, respectively, onto this hyperplane, and $D_{new}(\cdot, \cdot)$ is the new distance function.

**FastMap-D**

In this section, we present FastMap-D, a generalization of FastMap to directed graphs. We assume that the given directed graph, $G = (V, E, w)$, is strongly connected, that is, there exists a path from any vertex $v_i \in V$ to any other vertex $v_j \in V$. While FastMap produces an embedding of the vertices in a Euclidean space for undirected graphs, FastMap-D produces an embedding of the vertices in a potential field for directed graphs. A $K$-dimensional potential field is a function $\psi : \mathbb{R}^K \rightarrow \mathbb{R}$. The potential field is used to capture asymmetric distances that are inherent in directed graphs.

Figure 2 illustrates the difference between the embeddings created by FastMap and FastMap-D. FastMap creates a $K$-dimensional point $p_i = \langle [p_i]_1, \ldots, [p_i]_K \rangle$ for each vertex $v_i$, as shown in Figure 2(a). Here, the Euclidean distance $\|p_j - p_i\|_2 = \sqrt{\sum_{k=1}^{K} ([p_j]_k - [p_i]_k)^2}$ approximates the graph-based distance $d_G(v_i, v_j)$.

FastMap-D also creates a $K$-dimensional point $p_i = \langle [p_i]_1, \ldots, [p_i]_K \rangle$ for each vertex $v_i$, as shown in Figure 2(b). However, $[p_i]_K = \psi([p_i]_1, \ldots, [p_i]_{K-1})$ for some

---

This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/marisrg2/FastMap-D
\((K - 1)\)-dimensional potential field \(\psi\). The FastMap-D distance \(p_j - p_i\|_2\), defined to be \(\sqrt{\sum_{k=1}^{K-1}([p_j]_k - [p_i]_k)^2 + \|p_j - [p_i]_K\|_2}\), approximates \(d_G(v_i, v_j)\). The first term, \(\sqrt{\sum_{k=1}^{K-1}([p_j]_k - [p_i]_k)^2}\), approximates the symmetric average distance \(d_G(v_i, v_j) = \frac{d_G(v_i, v_j) + d_G(v_j, v_i)}{2}\), and the second term, \(\|p_j - [p_i]_K\|_2\), approximates the asymmetric correction component \(d_G(v_i, v_j) - d_G(v_i, v_j)\).

**Algorithm Description**

Algorithm 1 presents FastMap-D for directed graphs. The input is a non-negative edge-weighted directed graph \(G = (V, E, w)\) along with two user-specified parameters \(K_{max}\) and \(\epsilon\). \(K_{max}\) is the maximum number of dimensions allowed in the embedding. It bounds the amount of memory needed to store the embedding of any vertex. \(\epsilon\) is the threshold that marks a point of diminishing returns when the distance between the farthest pair of vertices becomes negligible. The output is an embedding \(p_i \in \mathbb{R}^{K+1}\) (with \(K + 1 \leq K_{max}\)) for each vertex \(v_i \in V\).

FastMap-D first embeds all vertices using average distances in a \(K\)-dimensional Euclidean space (lines 2-25). It then learns a potential function that is used to determine the \((K + 1)\)th coordinate (lines 26-38), which captures asymmetric distances as mentioned above.

**Embedding Average Distances:** This phase of the algorithm (lines 2-25) is similar to the regular FastMap procedure that is applicable to undirected graphs. However, the input here is a directed graph, and the distances are asymmetric. Thus, we use the average distances \(d_G(v_i, v_j)\) as a symmetric measure derived from the directed graph. All pairwise distances or average distances are never explicitly computed since doing so would be computationally expensive. Instead, we use the function \(\text{AVERAGE-DISTANCE}\) that is invoked only \(O(K_{max})\) times.

The function \(\text{AVERAGE-DISTANCE}\) (lines 39-43) computes \(d_G(v_i, v_j)\) for a given \(v_i\) and all \(v_j \in V\). It does this efficiently by computing two shortest path trees rooted at \(v_i\). The first is computed on \(G\) to yield \(d_G(v_i, v_j)\) for all \(v_j \in V\). The second is computed on \(G_R\), which is identical to \(G\) but with all edges reversed, to yield \(d_G_R(v_i, v_j)\) for all \(v_j \in V\).

In each iteration of \(K\) (line 4), the farthest pair of vertices \((v_a, v_b)\) is heuristically chosen in near-linear time (lines 5-14). This pair of vertices is identified with respect to the residual distances for that iteration (line 9).\(^4\) The square of the residual distances in iteration \(K\), \(d_{ij}^2 = \sum_{k=1}^{K-1}([p_j]_k - [p_i]_k)^2\), is the square of the original average distances minus the square of the Euclidean distances already explained by the first \(K - 1\) coordinates created so far. This is similar to the residual distances used in the \(L_2\)-variant of FastMap (Li et al. 2019). The farthest pair of vertices, \(v_a\) and \(v_b\), are added to \(\text{pivots}\), a list of pivots, before the \(K\)th coordinate for each vertex is computed (lines 16-25) as follows: First, the \(\text{AVERAGE-DISTANCE}\) function is called on \(v_a\) and \(v_b\) to yield \(d_{a_k}\) and \(d_{b_k}\) for all \(v_i \in V\) (lines 17-18). Then, the residual distances are computed (line 19), and FastMap’s triangle projection rule is used to compute the \(K\)th coordinate (lines 23-25).

**Learning a Potential Function:** This phase of the algorithm (lines 26-38) constructs a potential function \(\psi(x_1, \ldots, x_K)\) to account for asymmetric distances. The value of \(\psi([p_i]_1, \ldots, [p_i]_K)\) is recorded in the last coordinate of the embedding \([p_i]_{K+1}\). In this phase, a sampling procedure accompanies a learning procedure to construct \(\psi(x_1, \ldots, x_K)\). As shown in the pseudocode, \(\psi(x_1, \ldots, x_K)\) can be in the form of a multi-variate polynomial of degree \(D\) on \(x_1, \ldots, x_K\). It can also be in the form of a Neural Network (NN), as attempted in the next section. Of course, any machine learning algorithm can be used in this phase, but we choose to illustrate the pseudocode of the algorithm using polynomial fitting to facilitate our later discussion. While fitting a multi-variate polynomial can itself be done in many ways, here, we use LASSO to find the unknown coefficients of the multi-variate polynomial. The complexity of LASSO depends on the number of training samples (Efron et al. 2004).

The sampling procedure uses two qualifying sets \(S_1\) and \(S_2\) (line 27), and all pairs \((v_i, v_j)\) with \(v_i \in S_1\) and \(v_j \in S_2\) are used as training samples (line 32). The number of training samples is therefore \(|S_1||S_2|\). Different variants of FastMap-D can be created by varying the choices of \(S_1\) and \(S_2\). To keep the learning procedure efficient, both \(S_1\) and \(S_2\) cannot simultaneously be large subsets of \(V\). On the other hand, restricting \(S_1\) and \(S_2\) to significantly smaller subsets can negatively impact the accuracy of the learning procedure. Therefore, FastMap-D chooses \(S_1\) and \(S_2\) judiciously, sometimes using \(\text{pivots}\) computed in the first phase of the algorithm.

Consider a multi-variate polynomial \(\psi(x_1, \ldots, x_K)\) of degree \(D\) having the form \(\sum_{r=1}^{M} c_{r_1, \ldots, r_K} x_1^{r_1} \cdots x_K^{r_K}\), where \(d_1^{r_1}, \ldots, d_K^{r_K} \geq 0\) and \(d_1^{r_1} + \ldots + d_K^{r_K} \leq D\) for \(1 \leq r \leq M\). The number of terms, \(M\), in the multi-variate polynomial is given by \(\sum_{i=0}^{\lfloor \frac{D}{K-1} \rfloor} i\). We construct \(c \in \mathbb{R}^M\) to be the vector of unknown coefficients of \(\psi(x_1, \ldots, x_K)\). \(A \in \mathbb{R}^{|S_1||S_2| \times M}\) is a matrix in which row \(s\) corresponds to sample \(s\). For sample \(s = (v_i, v_j)\), \(\psi([p_j]_1, \ldots, [p_j]_K) - \psi([p_i]_1, \ldots, [p_i]_K)\) evaluates to a linear combination of the unknown coefficients and is desired to be equal to the correction factor \(d_G(v_i, v_j) - d_G(v_i, v_j)\), which is held in \(b_s\). Therefore, \(A_{sh}\) is equal to the coefficient of \([c]_h\) in \(\psi([p_j]_1, \ldots, [p_j]_K) - \psi([p_i]_1, \ldots, [p_i]_K)\).\(^5\)

Like the Ordinary Least Squares (OLS) method, LASSO minimizes \((Ac - b)^T(Ac - b)\) in \(O(|S_1||S_2|^{2M^2 + M^3})\) time to determine the unknown coefficients \(c_1, \ldots, c_M\) (line 36). However, it uses \(L_1\) regularization to address the regularization issues of OLS (Strutz 2010).

\(^4\)Note that \(d_{ij} = \bar{d}_G(v_i, v_j)\).

\(^5\)\(\psi(x_1, \ldots, x_K)\) has unknown coefficients on line 35 and, thus, \(\psi([p_j]_1, \ldots, [p_j]_K) - \psi([p_i]_1, \ldots, [p_i]_K)\) evaluates to a linear combination of the unknown coefficients. However, on line 38, the unknown coefficients have been determined and, thus, \(\psi([p_i]_1, \ldots, [p_i]_K)\) evaluates to a real number.
Algorithm 1: Shows the FastMap-D algorithm. \( G = (V, E, w) \) is a non-negative edge-weighted directed graph; \( K_{\text{max}} \) is the user-specified upper bound on the dimensionality; \( \epsilon \) is a user-specified threshold; \( K + 1 \leq K_{\text{max}} \) is the dimensionality of the computed embedding; \( p_i \) is the embedding of vertex \( v_i \in V \).

1. **Function FastMap-D**
   
   **Input:** \( G = (V, E, w) \), \( K_{\text{max}} \) and \( \epsilon \).
   
   **Output:** \( K + 1 \) and \( p_i \in \mathbb{R}^{K+1} \) for all \( v_i \in V \).

   - Embed average distances in Euclidean space using FastMap.
   
   `pivots ← ∅`;
   
   for \( K = \{1, \ldots, K_{\text{max}} - 1 \} \) do
     
     - Heuristically choose the farthest pair.
     
     Choose \( v_a \in V \) uniformly at random and let \( v_b ← v_a \);
     
     for \( r = 1, \ldots, C \) do // \( C \) is a small constant
       
       \( \{d_{ai}\}_{v_i \in V} ← \text{Average-Distance}(G, v_a) \);
       
       \( v_c ← \arg \max_{v_i} \{d_{ai}^2 - \sum_{k=1}^{K-1} (|p_{ai}|_k - |p_a|_k)^2 \} \);
       
       if \( d_{ic} < \epsilon \) then
         
         Break;
       
       else
         
         \( v_b ← v_a \);
         
         \( v_a ← v_c \);
         
         `pivots ← pivots ∪ \{v_a, v_b\}`;

     - Compute the \( K \)th coordinate.
     
     \( \{d_{ai}\}_{v_i \in V} ← \text{Average-Distance}(G, v_a) \);
     
     \( \{d_{ib}\}_{v_i \in V} ← \text{Average-Distance}(G, v_b) \);
     
     \( d_{ab} ← d_{ai}^2 - \sum_{k=1}^{K-1} (|p_{ai}|_k - |p_a|_k)^2 \);
     
     if \( d_{ab} < \epsilon \) then
       
       Break;
     
     for each \( v_i \in V \) do
       
       \( d_{ai} ← \sum_{k=1}^{K-1} (|p_{ai}|_k - |p_a|_k)^2 \);
       
       \( d_{ai}^2 ← d_{ai}^2 - \sum_{k=1}^{K-1} (|p_{ai}|_k - |p_a|_k)^2 \);
       
       \( |p_{i}^K| ← (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}) \);

   - Learn potential function and compute the last coordinate.

   Let \( S_1 \) and \( S_2 \) be the two qualifying sets that define the sampling procedure;
     
     Let \( \psi(x_1, \ldots, x_K) = \sum_{r=1}^{M} c_r x_1^{d_1} \ldots x_K^{d_K} \) be a multi-variate polynomial of degree \( D \) with unknown coefficients \( c_1, \ldots, c_M \); note that \( M = \sum_{i=0}^{D} (i+K-1) \);
     
     Let \( A = [A]_{ij} \) be a matrix of dimensions \( |S_1||S_2| \times M \);
     
     Let \( b = [b]_i \) be a vector of length \( |S_1||S_2| \);
     
     Let \( c = [c]_i \) be a vector of length \( M \);
     
     for each \( (v_i, v_j) \) such that \( v_i \in S_1, v_j \in S_2 \) do
       
       Let \( 1 \leq s \leq |S_1||S_2| \) be the current sampling index;
       
       \( [b]_s ← d_G(v_i, v_j) - d_G(v_i, v_j) \);
       
       Let \( [A]_{sh} \) be the coefficient of \( [c]_h \) in \( \psi([p_{i}^j]_1, \ldots, [p_{j}^j]_K) - \psi([p_{i}]_1, \ldots, [p_{i}]_K) \) for \( 1 \leq h \leq M \);
       
       \( c^* ← \text{LASSO solution to } (Ac - b)^T (Ac - b) \);
       
     for each \( v_i \in V \) do
       
       \( [p_{i}]_{K+1} ← \psi([p_{i}]_1, \ldots, [p_{i}]_K) \);

2. **Function Average-Distance**
   
   **Input:** \( G = (V, E, w) \) and a root \( v_i \in V \).
   
   **Output:** Average distance \( \frac{d_G(v_i, v_j) + d_G(v_j, v_i)}{2} \) for all \( v_j \in V \).

   Compute the shortest path tree rooted at \( v_i \) in \( G \) to get \( d_G(v_i, v_j) \) for all \( v_j \in V \);
   
   Let \( G_R \) be \( G \) with every edge reversed;
   
   Compute the shortest path tree rooted at \( v_i \) in \( G_R \) to get \( d_{G_R}(v_i, v_j) \) (that is, \( d_G(v_j, v_i) \)) for all \( v_j \in V \);
   
   return \( \frac{d_G(v_i, v_j) + d_{G_R}(v_j, v_i)}{2} \) for all \( v_j \in V \);

This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/mairirs2/FastMap-D
Time Complexity

FastMap-D makes $O(K_{\text{max}})$ calls to AVERAGE-DISTANCE. The time complexity of AVERAGE-DISTANCE is $O(|E| + |V| \log |V|)$. Therefore, the time complexity of the first phase of FastMap-D is $O(K_{\text{max}}(|E| + |V| \log |V|))$. Since LASSO takes $O(|S_1||S_2|^{2M} + M^2)$ time, the overall time complexity of FastMap-D is $O(K_{\text{max}}(|E| + |V| \log |V|) + |S_1||S_2|^{2M} + M^2)$, which is linear in $K_{\text{max}}$, near-linear in the size of the graph, linear in the number of training samples and exponential in the degree of $\psi$. In the next section, we discuss how to keep $|S_1||S_2|$ low. We also keep the degree of $\psi$ to a low constant.

Experiments

In this section, we present experimental results that demonstrate the benefits of FastMap-D. We conduct three kinds of experiments: (1) Comparing the accuracy of the embedding produced by FastMap-D to that of FastMap; (2) Evaluating different combinations of parameter values, specifically, the number of dimensions and the degree of the potential function $D$; and (3) Evaluating the effectiveness of NNs trained on the FastMap coordinates of the vertices over NNs trained directly on the grid coordinates of the vertices.

All experiments were conducted and evaluated on a 3.4GHz Intel-Xeon CPU with 64GB RAM. All algorithms were implemented in Python.

In our experiments, we also use a few implementation-level enhancements of the pseudocode of Algorithm 1. First, to exercise more control over the number of dimensions $K$, and to experiment with larger values of it, we try to avoid the break condition on line 21. We recognize that, if $d_{ab}^2 = \sum_{k=1}^{K-1} (|p_k|^2 - |\alpha_0|^2) < \epsilon$ on line 11, the break condition on line 21 is satisfied. Therefore, we modify line 11 to reassign $v_a$ and $v_b$ randomly and continue the loop without breaking if indeed $d_{ab}^2 = \sum_{k=1}^{K-1} (|p_k|^2 - |\alpha_0|^2) < \epsilon$. We also modify line 21 to set $d_{ab}$ to 1 instead of breaking the loop. Second, to avoid obtuse triangles for the cosine law projection in Figure 1(a), we modify lines 23 and 24 so that $d_{ai} \leftarrow \max(0, d_{ai}^2 - \sum_{k=1}^{K-1} (|p_k|^2 - |\alpha_0|^2))$ and $d_{ib} \leftarrow \max(0, d_{ib}^2 - \sum_{k=1}^{K-1} (|p_k|^2 - |\alpha_0|^2))$.

Since the machine learning module of FastMap-D is designed to be a plug-and-play component, we implemented it using LASSO as well as an NN method. For LASSO, we found it beneficial to set $S_1$ to pivots since the pivots can be thought of as critical vertices identified in the first phase of FastMap-D. $S_2$ is set to a randomly chosen subset of vertices such that $|S_1||S_2| \geq M$. For training NNs, however, we generated training samples slightly differently (as described later in that subsection).

Although there exist benchmark instances for directed graphs, none of them come with the assurance of being strongly connected. For this reason, and to allow for a direct comparison with FastMap, the maps in this section are taken from a standard benchmark repository for undirected graphs (Sturtevant 2012), which were also used in (Cohen et al. 2018; Li et al. 2019). For each map, we converted every edge into two directed edges in opposite directions to generate a directed version of it. We first created a virtual height $h(v)$ for each vertex $v \in V$ based on its 2D grid coordinates $(x_v, y_v)$. The height is assigned according to two possibilities: (a) polynomial function $h(v) = x_v + y_v + (x_v + y_v)^3$, or (b) exponential function $h(v) = 1.01^{x_v} + 1.02^{y_v} + 1.03^{x_v+y_v}$. Then, we set $w(v_i, v_j)$ to be $2(h(v_j) - h(v_i))$ when $h(v_j) \geq h(v_i)$, and $(h(v_i) - h(v_j))/2$ otherwise.

To measure distortion, we use the Normalized Root Mean Square Error (NRMSE). We sample $N$ random distances between nodes, and compare them against their corresponding embedding distances $\|p_j - p_i\|$. To normalize the data coming from graphs of different sizes, the NRMSE is given by $\sigma/d$ where $\sigma = \sqrt{\frac{\sum_{i<j} w(i,j) (d_{ij} - \|p_j - p_i\|)^2}{N}}$ and $d = \frac{\sum_{i<j<i<j} d_{ij}}{N}$.

FastMap vs FastMap-D: Figure 3 shows the NRMSE values of FastMap and FastMap-D for different values of the number of dimensions $K$ on four different kinds of maps. The top four panels show the results using the polynomial height function, and the bottom four panels show the results using the exponential height function. In all cases, we set the degree $D$ of $\psi$ to 2. Since FastMap works only for undirected graphs, it can only embed the symmetric distances $d_{ij}(v_i, v_j)$. In other words, FastMap and FastMap-D differ only in the last coordinate that FastMap uses as an additional coordinate and FastMap-D uses as a correction factor to account for asymmetric distances.

We observe that FastMap-D outperforms FastMap on all maps for a sufficiently large number of dimensions $K$. Not only does FastMap-D outperform FastMap on mazes and random maps, but it also significantly outperforms FastMap on structured game maps and real-world city maps such as ‘hrt201n’ and ‘Boston 2,256’. We also note that the FastMap-D NRMSE values often decrease faster than the FastMap NRMSE values for increasing $K$. This shows that FastMap-D utilizes additional dimensions better than FastMap does.

Varying FastMap-D Parameter Values: Figure 4 shows the effect of $K$ and $D$ on the NRMSE values of FastMap-D for a representative map. In general, increasing $K$ improves the NRMSE values. However, increasing $D$ is not always very helpful.

NNs on FastMap Coordinates: NNs can learn pairwise distances between the vertices in a grid map. Naively applied, an NN can be trained on the 2D grid coordinates of the vertices.

\textsuperscript{6}$K + 1$ in pseudocode of Algorithm 1
\textsuperscript{7}Otherwise, $d_{ab}$ has a very low value, and the division in line 25 leads to numerical instability.
\textsuperscript{8}Unless specified otherwise, FastMap-D refers to the version with LASSO.
Figure 3: Shows the NRMSE values of FastMap and FastMap-D for different values of the number of dimensions $K$. In all cases, the degree $D$ of $\psi$ is 2. The undirected version of the map is shown as an inlay.

This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/marirsg2/FastMap-D
Table 1: The performance of FastMap-D compared to FastMap coordinates. The direct approach is used for training and FastMap-D is used for evaluating the performance. The direct approach uses 4, 1000, 500, 200, 2000, and 1 nodes in fully connected consecutive layers. The NN is used for FastMap-D uses $K = 10$ and 30, 1000, 500, and 1 nodes in fully connected consecutive layers. 535860, 703930 and 1458360 training samples were used for the Lak503d, htt201n and Boston 2.256 maps, respectively. ‘Poly’ and ‘Exp’ indicate polynomial and exponential height functions, respectively.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Direct NN</th>
<th>FastMap-D (NN)</th>
<th>FastMap-D (LASSO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lak503d Polyn</td>
<td>1.238</td>
<td>0.048</td>
<td>0.042</td>
</tr>
<tr>
<td>Lak503d Expyn</td>
<td>0.901</td>
<td>0.071</td>
<td>0.089</td>
</tr>
<tr>
<td>htt201n Polyn</td>
<td>0.944</td>
<td>0.028</td>
<td>0.077</td>
</tr>
<tr>
<td>htt201n Expyn</td>
<td>1.239</td>
<td>0.083</td>
<td>0.271</td>
</tr>
<tr>
<td>Boston 2.256 Polyn</td>
<td>0.994</td>
<td>0.039</td>
<td>0.109</td>
</tr>
<tr>
<td>Boston 2.256 Expyn</td>
<td>2.795</td>
<td>0.043</td>
<td>0.501</td>
</tr>
</tbody>
</table>

Table 1 shows the benefit of training NNs on the FastMap coordinates compared to training them on the grid coordinates. For the same number of training samples, the NRMSE values of FastMap-D with NNs are significantly smaller than those of the direct approach with NNs that uses the grid coordinates. They are also smaller than those of FastMap-D with LASSO.

Conclusions and Future Work

In this paper, we generalized FastMap for undirected graphs to FastMap-D for directed graphs. FastMap-D efficiently embeds the vertices of a given directed graph in a potential field. Unlike a Euclidean embedding, a potential-field embedding can represent asymmetric distances. FastMap-D uses a machine learning module to learn a potential function that defines the potential field. In experiments, we demonstrated the advantage of FastMap-D on various kinds of directed graphs. An important upshot of our approach is that applying machine learning algorithms to the FastMap coordinates of the vertices of a graph is much better than applying them directly to the grid coordinates of the vertices since the FastMap coordinates capture important information of the link structure of the graph - not to mention that the grid coordinates are not even defined for general graphs.

In future work, we will apply FastMap-D to very large directed graphs, such as knowledge graphs, and to intensional graphs, such as in automated planning and plan visualization. The success of FastMap-D exemplifies the benefits of using the FastMap coordinates as features for machine learning algorithms, and we hope to do the same for other graph problems that involve machine learning.

Acknowledgements

The research at Arizona State University was supported by ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840 and N00014-19-1-2119. The research at the University of Southern California was supported by NSF grants 1724392, 1409987, 1817189, 1837779 and 1935712.

This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/marirsg2/FastMap-D
References


This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: https://github.com/marirsg2/FastMap-D