Embedding Directed Graphs in Potential Fields Using FastMap-D

Sriram Gopalakrishnan[1] and **Liron Cohen**[2] and **Sven Koenig**[2] and **T. K. Satish Kumar**[2]
[1]Arizona State University
[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[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.[2]

The difference in potential of two points in a potential

[1]linear time after ignoring logarithmic factors

[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*.

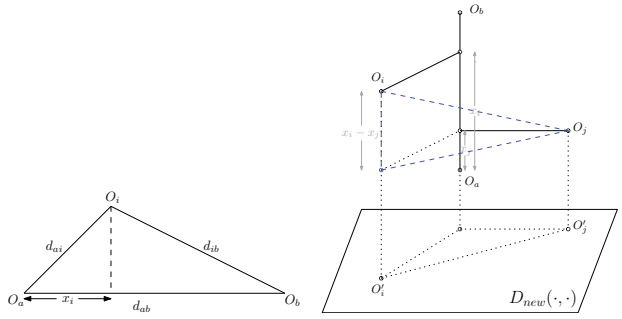
field is inherently asymmetric and therefore a good choice for capturing distances in a directed graph. FastMap-D constructs a potential function defining the potential field using a machine learning module. Through experiments conducted on various kinds of directed graphs, we demonstrate the advantage 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 machine learning module that works well even with the Least Absolute Shrinkage and Selection Operator (LASSO) regression method, in which case it has a strongly polynomial runtime (Strutz 2010). HOPE (Ou et al. 2016) is another algorithm that tries to preserve higher-order proximities, but it forgoes the use of random walks in favor of an approximate Singular Value Decomposition (SVD) of the similarity matrix. The top eigenvectors define the embedding space. Since HOPE relies on solving SVDs, it is prohibitively expensive for large graphs.



(a) (b)

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

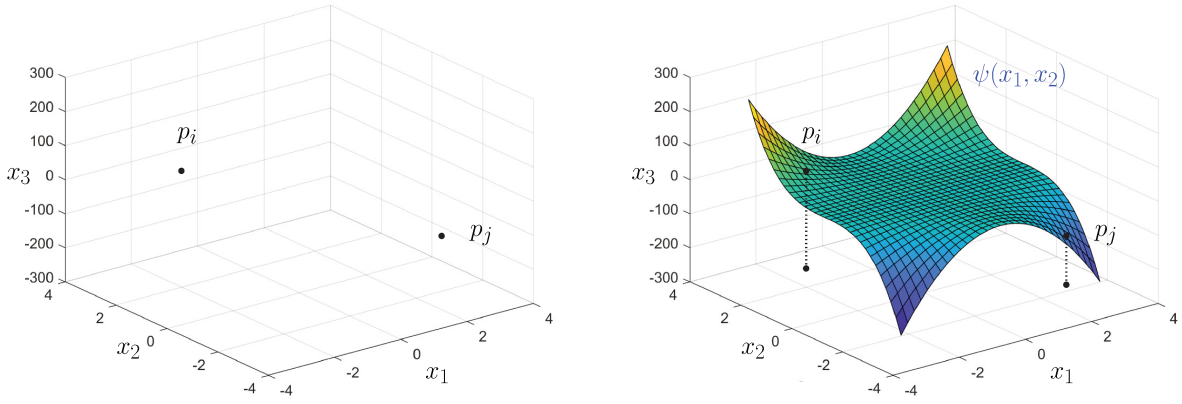
Background

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*[3] 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 $p_i \in \mathbb{R}^K$

[3]The edit distance between two strings is the minimum number of insertions, deletions or substitutions that are needed to transform one to the other.

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



(a) Embedding produced by FastMap

(b) Embedding produced by FastMap-D

Figure 2: Illustrates the difference between (a) the embedding produced by FastMap and (b) the embedding produced by FastMap-D. In (a), 3 dimensions are used to represent the symmetric distances between vertices. In (b), 2 dimensions are used to represent the symmetric average distances between vertices and the 3rd dimension is used to represent the correction factors via the potential function ψ. Points p_i and p_j are the embeddings of vertices v_i and v_j, respectively.

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_b and then choosing O_a to be the object farthest away from O_b. 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 $d_{ib} = D(O_i, O_b)$. Figure 1(a) shows this triangle. The sides of the triangle define its entire geometry, and the projection of O_i onto $\overline{O_a O_b}$ is given by $x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab})$. 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 $i, j \notin \{a, b\}$ 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 $\overline{O_a O_b}$. 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 to: $D_{new}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2$. 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 $\psi : \mathbb{R}^K \to \mathbb{R}$. 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/marirsg2/FastMap-D

$(K-1)$-dimensional potential field ψ. The FastMap-D distance $\|p_j - p_i\|_\odot$, defined to be $\sqrt{\sum_{k=1}^{K-1}([p_j]_k - [p_i]_k)^2} + [p_j]_K - [p_i]_K$, 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 $\bar{d}_G(v_i, v_j) = \frac{d_G(v_i,v_j)+d_G(v_j,v_i)}{2}$, and the second term, $[p_j]_K - [p_i]_K$, approximates the asymmetric correction component $d_G(v_i, v_j) - \bar{d}_G(v_i, v_j)$.

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 ϵ. 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. ϵ 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 $\bar{d}_G(v_i, v_j)$ 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 AVERAGE-DISTANCE that is invoked only $O(K_{max})$ times.

The function AVERAGE-DISTANCE (lines 39-43) computes $\bar{d}_G(v_i, v_j)$ 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 $d_{G_R}(v_i, v_j) = d_G(v_j, v_i)$ 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 `pivots`, a list of pivots, before the Kth coordinate for each vertex is computed (lines 16-25) as follows: First, the AVERAGE-DISTANCE function is called on v_a and v_b to yield d_{ai} and d_{ib} 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 Kth 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 `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 $\sum_{r=1}^{M} c_r x_1^{d_1^r} \ldots x_K^{d_K^r}$, where $d_1^r, \ldots, d_K^r \geq 0$ and $d_1^r + \ldots + d_K^r \leq D$ for $1 \leq r \leq M$. The number of terms, M, in the multi-variate polynomial is given by $\sum_{i=0}^{D} \binom{i+K-1}{K-1}$. 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 $d_G(v_i, v_j) - \bar{d}_G(v_i, v_j)$, 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 $O(|S_1||S_2|M^2 + M^3)$ 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.

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

Algorithm 1: Shows the FastMap-D algorithm. $G = (V, E, w)$ is a non-negative edge-weighted directed graph; K_{max} is the user-specified upper bound on the dimensionality; ϵ is a user-specified threshold; $K + 1 \leq K_{max}$ is the dimensionality of the computed embedding; p_i is the embedding of vertex $v_i \in V$.

Function `FastMap-D`
- **Input:** $G = (V, E, w)$, K_{max} and ϵ.
- **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` $\leftarrow \emptyset$;
  - **for** $K = \{1, \ldots, K_{max} - 1\}$ **do**
    - • *Heuristically choose the farthest pair.*
      - Choose $v_a \in V$ uniformly at random and let $v_b \leftarrow v_a$;
      - **for** *t = 1, . . . , C* **do** `// C is a small constant`
        - $\{d_{ai}\}_{v_i \in V} \leftarrow$ `Average-Distance` (G, v_a);
        - $v_c \leftarrow \arg\max_{v_i}\{d_{ai}^2 - \sum_{k=1}^{K-1}([p_i]_k - [p_a]_k)^2\}$;
        - **if** $v_c = v_b$ **then**
          - Break;
        - **else**
          - $v_b \leftarrow v_a$;
          - $v_a \leftarrow v_c$;
    - `pivots` $\leftarrow$ `pivots` $\cup \{v_a, v_b\}$;
    - • *Compute the Kth coordinate.*
      - $\{d_{ai}\}_{v_i \in V} \leftarrow$ `Average-Distance` (G, v_a);
      - $\{d_{ib}\}_{v_i \in V} \leftarrow$ `Average-Distance` (G, v_b);
      - $d'_{ab} \leftarrow d_{ab}^2 - \sum_{k=1}^{K-1}([p_b]_k - [p_a]_k)^2$;
      - **if** $d'_{ab} < \epsilon$ **then**
        - Break;
      - **for** *each* $v_i \in V$ **do**
        - $d'_{ai} \leftarrow d_{ai}^2 - \sum_{k=1}^{K-1}([p_i]_k - [p_a]_k)^2$;
        - $d'_{ib} \leftarrow d_{ib}^2 - \sum_{k=1}^{K-1}([p_b]_k - [p_i]_k)^2$;
        - $[p_i]_K \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$;
- • *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 $\psi(x_1, \ldots, x_K) = \sum_{r=1}^{M} c_r x_1^{d_1^r} \ldots x_K^{d_K^r}$ be a multi-variate polynomial of degree D with unknown coefficients $c_1, \ldots, c_M$; note that $M = \sum_{i=0}^{D} \binom{i+K-1}{K-1}$;
  - 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;
    - $[b]_s \leftarrow d_G(v_i, v_j) - \bar{d}_G(v_i, v_j)$;
    - Let $[A]_{sh}$ be the coefficient of $[c]_h$ in $\psi([p_j]_1, \ldots, [p_j]_K) - \psi([p_i]_1, \ldots, [p_i]_K)$ for $1 \leq h \leq M$;
  - $c^* \leftarrow$ LASSO solution to $(Ac - b)^T(Ac - b)$;
  - **for** *each* $v_i \in V$ **do**
    - $[p_i]_{K+1} \leftarrow \psi([p_i]_1, \ldots, [p_i]_K)$;

Function `Average-Distance`
- **Input:** $G = \langle V, E, w\rangle$ 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** $\frac{d_G(v_i,v_j)+d_G(v_j,v_i)}{2}$ *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/marirsg2/FastMap-D

Time Complexity

FastMap-D makes $O(K_{max})$ 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 $O(K_{max}(|E| + |V|\log|V|))$. Since LASSO takes $O(|S_1||S_2|M^2 + M^3)$ time, the overall time complexity of FastMap-D is $O(K_{max}(|E| + |V|\log|V|) + |S_1||S_2|M^2 + M^3)$, which is linear in K_{max}, near-linear in the size of the graph, linear in the number of training samples and exponential in the degree of ψ. In the next section, we discuss how to keep $|S_1||S_2|$ low. We also keep the degree of ψ 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[6] K 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 $d_{ab}^2 - \sum_{k=1}^{K-1}([p_b]_k - [p_a]_k)^2 < \epsilon$ 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 $d_{ab}^2 - \sum_{k=1}^{K-1}([p_b]_k - [p_a]_k)^2 < \epsilon$. We also modify line 21 to set d'_{ab} to 1 instead of breaking the loop.[7] Second, to avoid obtuse triangles for the cosine law projection in Figure 1(a), we modify lines 23 and 24 so that $d'_{ai} \leftarrow \max(0, d_{ai}^2 - \sum_{k=1}^{K-1}([p_i]_k - [p_a]_k)^2)$ and $d'_{ib} \leftarrow \max(0, d_{ib}^2 - \sum_{k=1}^{K-1}([p_b]_k - [p_i]_k)^2)$.

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.[8] 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 $h(v) = x_v + y_v^2 + (x_v + y_v)^3$, 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 $2(h(v_j) - h(v_i))$ if $h(v_j) \geq h(v_i)$, 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 $\|p_j - p_i\|_\odot$. To normalize the data coming from graphs of different sizes, the NRMSE is given by $\sigma/\bar{d}$ where $\sigma = \sqrt{\frac{\sum_{1\leq i\neq j\leq S}(d_{ij} - \|p_j - p_i\|_\odot)^2}{N}}$ and $\bar{d} = \frac{\sum_{1\leq i\neq j\leq S} d_{ij}}{N}$.

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,[9] and the bottom four panels show the results using the exponential height function. In all cases, we set the degree D of ψ to 2. Since FastMap works only for undirected graphs, it can only embed the symmetric distances $\bar{d}_G(v_i, v_j)$. 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

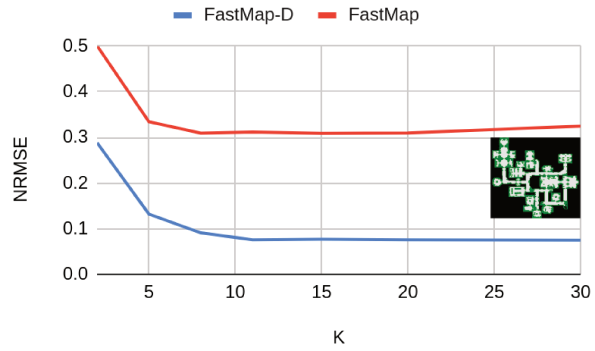
[6] $K + 1$ in pseudocode of Algorithm 1

[7] Otherwise, d'_{ab} has a very low value, and the division in line 25 leads to numerical instability.

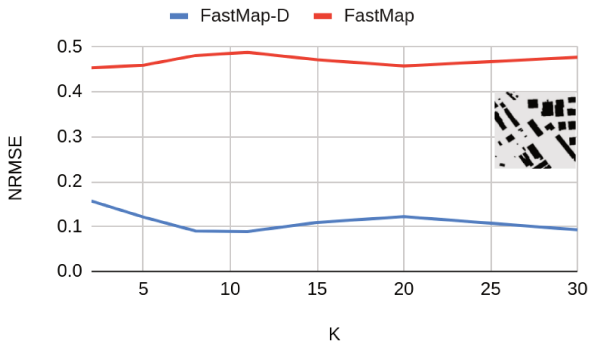
[8] Unless specified otherwise, FastMap-D refers to the version with LASSO.

[9] Using the polynomial height function for edge weights does not mean that the shortest path distances between vertices follow the same pattern. This is so because the map still has obstacles and the edge weights combine in complex ways to form shortest paths and graph distances.

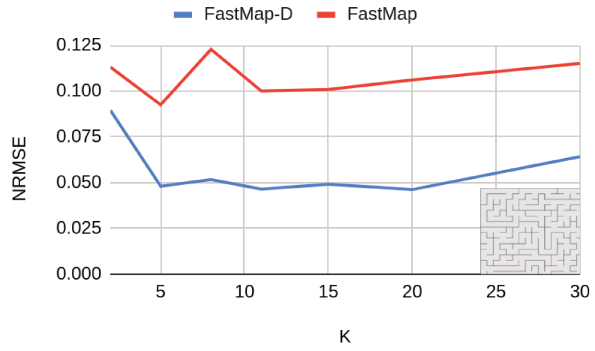
This version of the paper is intended to update the version published in the Proceedings of the Thirteenth International Symposium on Combinatorial Search (SoCS-2020). The update has been made in recognition of a programming bug, fixing which has improved our experimental results. Code available at: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



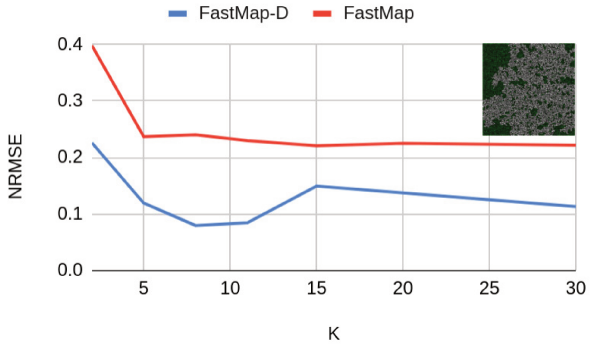
(a) hrt201n with polynomial height function



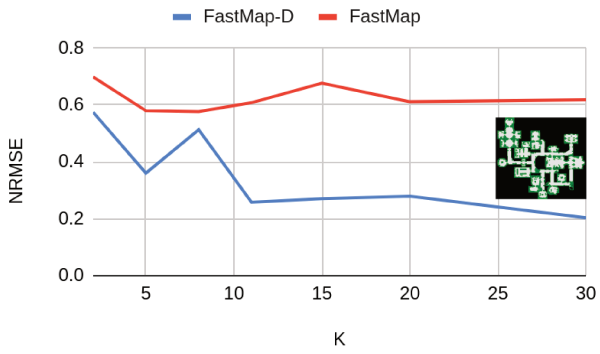
(b) Boston 2_256 with polynomial height function



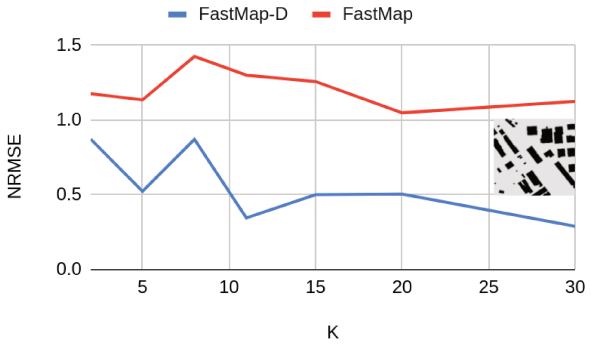
(c) maze 512-32-0 with polynomial height function



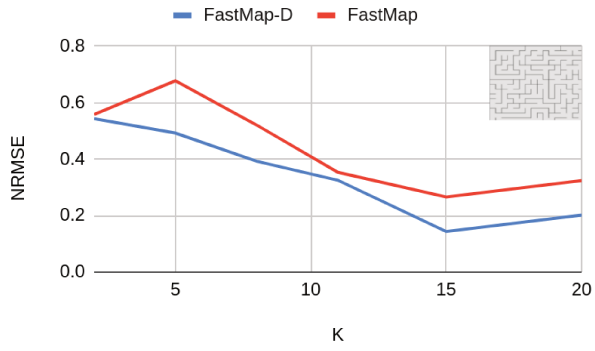
(d) random 512-40-0 with polynomial height function



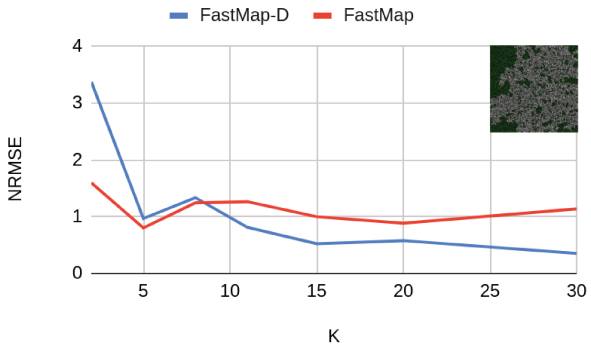
(e) hrt201n with exponential height function



(f) Boston 2_256 with exponential height function



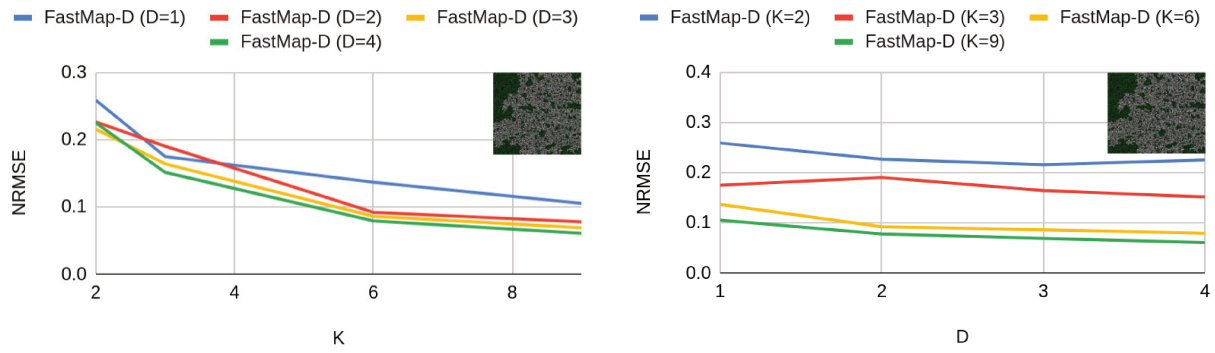
(g) maze 512-32-0 with exponential height function



(h) random 512-40-0 with exponential height function

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 ψ 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



(a) random 512-40-0 with polynomial height function

(b) random 512-40-0 with polynomial height function

Figure 4: Shows the NRMSE values of FastMap-D with different parameter values. K is the number of dimensions; and D is the degree of ψ. The undirected version of the map is shown as an inlay.

Instance	Direct NN	FastMap-D (NN)	FastMap-D (LASSO)
Lak503d Poly	1.238	0.048	0.042
Lak503d Exp	0.901	0.071	0.089
hrt201n Poly	0.944	0.028	0.077
hrt201n Exp	1.239	0.083	0.271
Boston 2_256 Poly	0.994	0.039	0.109
Boston 2_256 Exp	2.795	0.043	0.501

Table 1: Compares the NRMSE values of the direct NN approach and FastMap-D on a few representative maps. The best NN designed for the direct approach uses 4, 1000, 500, 200, 200 and 1 nodes in fully connected consecutive layers. The NN designed for FastMap-D uses $K = 15$ and 30, 1000, 500 and 1 nodes in fully connected consecutive layers. 538560, 709530 and 1458360 training samples were used for the Lak503d, hrt201n and Boston 2_256 maps, respectively. 'Poly' and 'Exp' indicate polynomial and exponential height functions, respectively.

the source and destination vertices. However, there are several problems with this direct approach. First, it is not applicable to general graphs where vertices do not have coordinates. Second, even for grid maps, the feature set is very small since it is limited to the grid coordinates. Third, it remains oblivious of many parts of the graph even with a large number of training samples since there are a quadratic number of pairs of source and destination vertices.

There are several benefits of using the FastMap coordinates instead of the grid coordinates for training NNs. First, this approach is applicable to general graphs. Second, the feature set is larger and depends on the user-controlled parameter K. Third, since the link structure of the graph is summarized in the FastMap coordinates, not too many samples are required. In fact, we simply use $|\texttt{pivots}|$ randomly selected vertices, compute the shortest path trees rooted at each of them, and draw training samples only from these trees with the source vertex restricted to be the root vertex. This keeps the number of training samples linear in the size of the graph.

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

Alpaydin, E. 2010. *Introduction to Machine Learning*. The MIT Press, 2nd edition.

Bordes, A.; Usunier, N.; Garcia-Durán, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2787–2795.

Bordes, A.; Weston, J.; and Usunier, N. 2014. Open question answering with weakly supervised embedding models. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

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 International Joint Conference on Artificial Intelligence*, 1427–1433.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Annual Symposium on Computational Geometry*, 253–262.

Efron, B.; Hastie, T.; Johnstone, I.; and Tibshirani, R. 2004. Least angle regression. *The Annals of Statistics* 32(2):407–499.

Faloutsos, C., and Lin, K.-I. 1995. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Grover, A., and Leskovec, J. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855–864.

Li, J.; Felner, A.; Koenig, S.; and Kumar, T. K. S. 2019. Using fastmap to solve graph problems in a euclidean space. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 273–278.

Linial, N.; London, E.; and Rabinovich, Y. 1995. The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15(2):215–245.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Ng, T. E., and Zhang, H. 2002. Predicting internet network distance with coordinates-based approaches. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, 170–179.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1105–1114.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710.

Rayner, C.; Bowling, M.; and Sturtevant, N. 2011. Euclidean heuristic optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Shavitt, Y., and Tankel, T. 2004. Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Transactions on Networking* 12(6):993–1006.

Strutz, T. 2010. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. Vieweg and Teubner.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the International World Wide Web Conference*, 1067–1077.

Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; and Gao, J. 2017. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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