A FastMap-Based Algorithm for Block Modeling[⋆]

Ang Li[1], Peter Stuckey[2], Sven Koenig[1], and T. K. Satish Kumar[1]

[1] University of Southern California, Los Angeles, CA 90007, USA
{ali355,skoenig}@usc.edu, tkskwork@gmail.com
[2] Monash University, Wellington Rd, Clayton VIC 3800, Australia
peter.stuckey@monash.edu

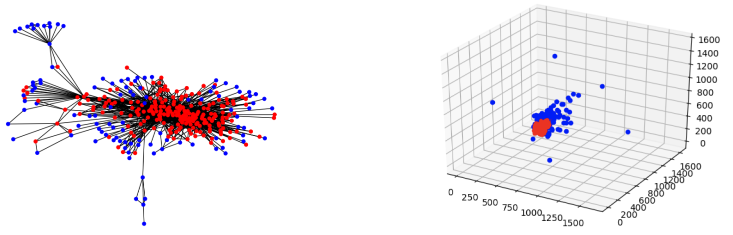
Abstract. Block modeling algorithms are used to discover important latent structures in graphs. They are the graph equivalent of clustering algorithms. However, existing block modeling algorithms work directly on the given graphs, making them computationally expensive and less effective on large complex graphs. In this paper, we propose a FastMap-based algorithm for block modeling on single-view undirected graphs. FastMap embeds a given undirected graph into a Euclidean space in near-linear time such that the pairwise Euclidean distances between vertices approximate a desired graph-based distance function between them. In the first phase, our FastMap-based block modeling (FMBM) algorithm uses FastMap with a probabilistically-amplified shortest-path distance function between vertices. In the second phase, it uses Gaussian Mixture Models (GMMs) for identifying clusters (blocks) in the resulting Euclidean space. FMBM outperforms other state-of-the-art methods on many benchmark and synthetic instances, both in efficiency and solution quality. It also enables a perspicuous visualization of clusters (blocks) in the graphs, not provided by other methods.

Keywords: Community Detection and Block Modeling · Graph Embeddings · FastMap.

1 Introduction

Finding inherent groups in graphs, i.e., the "graph" clustering problem, has important applications in many real-world domains, such as identifying communities in social networks [8], analyzing the diffusion of ideas in them [13], identifying functional modules in protein-protein interactions [11], and understanding the modular design of brain networks [2]. In general, identifying the

⋆ This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government. This research was partially supported by the OPTIMA ARC training centre IC200100009.



(a) a core-periphery graph in the airport domain

(b) a FastMap embedding of the graph on the left

Fig. 1: The left side shows a core-periphery graph in the airport domain with edges representing flight connections, red vertices representing "hub" airports at the core, and blue vertices representing "local" airports at the periphery. The right side shows a FastMap embedding of the graph in Euclidean space, in which the red and blue vertices correctly appear in the core and periphery, respectively.

groups involves mapping each vertex in the graph to a group (cluster), where vertices in the same group share important properties in the underlying graph.

The conditions under which two vertices are deemed to be similar and therefore belonging to the same group are popularly studied in community detection and block modeling [1]. In community detection, a group (community) implicitly requires its vertices to be more connected to each other than to vertices of other groups. Although this is justified in many real-world domains, such as social networks, it is not always justified in general.

Block modeling uses more general criteria for identifying groups (blocks) where community detection fails. For example, block modeling can be used to correctly identify groups in *core-periphery* graphs characterized by a core of vertices tightly connected to each other and a peripheral set of vertices loosely connected to each other but well connected to the core.[3] Core-periphery graphs are common in many real-world domains, such as financial networks and flight networks [1,19]. Figure 1a shows a core-periphery graph in an air flight domain [5].

Existing block modeling algorithms work directly on the given graphs and are inefficient. They typically use matrix operations that incur cubic time complexities even within their inner loops. For example, FactorBlock [3], a state-of-the-art block modeling algorithm, uses matrix multiplications in its inner loop and an expectation-maximization-style outer loop. Due to their inefficiency, existing block modeling algorithms are not scalable and result in poor solution qualities on large complex graphs.

In this paper, we propose a FastMap-based algorithm for block modeling on single-view[4] undirected graphs. FastMap embeds a given undirected graph into

[3] The conditions used in community detection prevent the proper identification of peripheral groups.

[4] In a single-view graph, there is at most one edge between any two vertices.

a Euclidean space in near-linear time such that the pairwise Euclidean distances between vertices approximate a desired graph-based distance function between them. In general, graph embeddings have been used in many different contexts, such as for shortest-path computations [4], multi-agent meeting problems [12], and social network analysis [18]. They are useful because they facilitate geometric interpretations and algebraic manipulations in vector spaces.

FastMap [4,12] is a recently developed graph embedding algorithm that runs in near-linear time[5]. While it has thus far been used to create Euclidean embeddings that approximate pairwise shortest-path distances between vertices, it can also be extended to creating Euclidean embeddings that approximate more general pairwise graph-based distances between vertices. In particular, for the purpose of block modeling in this paper, we propose a novel distance function that probabilistically amplifies the shortest-path distances between vertices.

Our FastMap-based block modeling algorithm (FMBM) works in two phases. In the first phase, FMBM uses FastMap to efficiently embed the vertices of a given graph in a Euclidean space, preserving the probabilistically-amplified shortest-path distances between them. In the second phase, FMBM identifies clusters (blocks) in the resulting Euclidean space using standard methods from unsupervised learning. Therefore, the first phase of FMBM efficiently reformulates the block modeling problem from a graphical space to a Euclidean space, as illustrated in Figure 1b; and the second phase of FMBM leverages any technique that is already known or can be developed for clustering in Euclidean space. In our current implementation of FMBM, we use Gaussian Mixture Models (GMMs) for identifying clusters in the Euclidean space.

We empirically show that, in addition to the theoretical advantages of FMBM, it outperforms other state-of-the-art methods on many benchmark and synthetic test cases. We report on the superior performance of FMBM both in terms of efficiency and solution quality. We also show that it enables a perspicuous visualization of clusters in the graphs, beyond the capabilities of other methods.

2 Preliminaries and Background

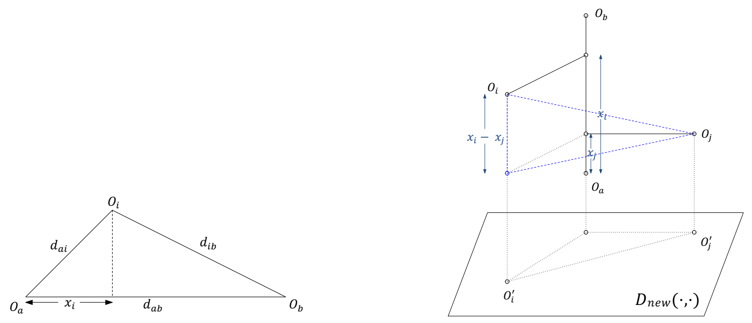
In this section, we review some preliminaries of block modeling and provide a background description of FastMap.

2.1 Block Modeling

Let $G = (V, E)$ be an undirected graph with vertices $V = \{v_1, v_2 \dots v_n\}$ and edges $E = \{e_1, e_2 \dots e_m\} \subseteq V \times V$. Let $A \in \{0,1\}^{n \times n}$ be the adjacency matrix representation of G, where $A_{ij} = 1$ iff $(v_i, v_j) \in E$.

A *block model* decomposes G 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. Therefore,

[5] i.e., linear time after ignoring logarithmic factors



(a) the “cosine law” projection in a triangle

(b) projection onto a hyperplane that is perpendicular to $\overline{O_a O_b}$

Fig. 2: The figure, borrowed from [4], illustrates how coordinates are computed and recursion is carried out in FastMap.

$\sum_{j=1}^{k} C_{ij} = 1$ for all $1 \leq i \leq n$. 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, as discussed in [3], tries to approximate A by $CMC^\top$ with the best choice for C and M. In other words, the objective is

$$\min_{C,M} \|A - CMC^\top\|_F^2, \tag{1}$$

where $\| \cdot \|_F$ is the Frobenius norm. An improved objective function is also considered in [3] to account for the imbalance of edges to non-edges[6] in A, since real-world graphs are typically sparse with significantly more non-edges than edges. The revised objective is

$$\min_{C,M} \|(A - CMC^\top) \circ (A - R)\|_F^2, \tag{2}$$

where $R \in [0,1]^{n \times n}$, $R_{ij} = \frac{m}{n^2}$, and $\circ$ represents element-wise multiplication.

The above formalization can be generalized to directed graphs and multi-view graphs [19]. It can also be generalized to soft partitioning, where each vertex partially belongs to each partition, i.e., $C \in [0,1]^{n \times k}$ with $\sum_{j=1}^{k} C_{ij} = 1$ for all $1 \leq i \leq n$.

2.2 FastMap

FastMap [6] was introduced in the Data Mining community for automatically generating Euclidean embeddings of abstract objects. For many complex objects (such as long DNA strings), multi-media datasets (like voice excerpts or images),

[6] i.e., a pair of vertices not connected by an edge

or medical datasets (like ECGs or MRIs), there is no geometric space in which they can be naturally visualized. However, there is often a well-defined distance function between every pair of objects in the problem domain. For example, the *edit distance* between two DNA strings is well-defined although an individual DNA string cannot be conceptualized in geometric space.

FastMap 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 $\mathcal{O}$, where $D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in \mathcal{O}$. A Euclidean embedding assigns a K-dimensional point $\vec{p}_i \in \mathbb{R}^K$ to each object O_i. For $\vec{p}_i = ([\vec{p}_i]_1, [\vec{p}_i]_2 \ldots [\vec{p}_i]_K)$ and $\vec{p}_j = ([\vec{p}_j]_1, [\vec{p}_j]_2 \ldots [\vec{p}_j]_K)$, we define the Euclidean distance $\chi_{ij} = \sqrt{\sum_{r=1}^{K}([\vec{p}_j]_r - [\vec{p}_i]_r)^2}$. A good Euclidean embedding is one in which χ_{ij} between any two points $\vec{p}_i$ and $\vec{p}_j$ closely approximates $D(O_i, O_j)$.

FastMap creates a K-dimensional Euclidean embedding of the abstract objects in $\mathcal{O}$ for a user-specified value of K. 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 2a. The sides of the triangle define its entire geometry, and the projection of O_i onto the line $\overline{O_aO_b}$ is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}). \tag{3}$$

FastMap sets the first coordinate of $\vec{p}_i$, the embedding of O_i, to x_i. 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 between these two objects, from the second iteration onward, the rest of $\vec{p}_a$ and $\vec{p}_b$'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 $\overline{O_aO_b}$, as shown in Figure 2b. Although the hyperplane is never constructed explicitly, its conceptualization implies that the distances for the second iteration should be changed for all i and j so that:

$$D_{new}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2. \tag{4}$$

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 can also be used to embed the vertices of a graph in a Euclidean space to preserve the pairwise shortest-path distances between them. The idea is to view the vertices of a given graph $G = (V, E)$ as the objects to be embedded. As such, the Data Mining FastMap algorithm cannot be directly used for generating an embedding in near-linear time. This is so because it assumes that the distance d_{ij} between any two objects O_i and O_j can be computed in constant time, independent of the number of objects in the problem domain. However, computing the shortest-path distance between two vertices depends on the size of the graph.

The near-linear time complexity of FastMap can be retained as follows: In each iteration, after we heuristically identify the farthest pair of vertices O_a and O_b, the distances d_{ai} and d_{ib} need to be computed for *all* other vertices O_i. Computing d_{ai} and d_{ib} for any single vertex O_i can no longer be done in constant time but requires $O(|E| + |V| \log |V|)$ time instead [7]. However, since we need to compute these distances for all vertices, computing two shortest-path trees rooted at each of the vertices O_a and O_b yields all necessary distances in one shot. The complexity of doing so is also $O(|E| + |V| \log |V|)$, which is only linear in the size of the graph[7]. The amortized complexity for computing d_{ai} and d_{ib} for vertex O_i is therefore near-constant time.

The foregoing observations are used in [12] to build a graph-based version of FastMap that embeds the vertices of a given undirected graph in a Euclidean space in near-linear time. The Euclidean distances approximate the pairwise shortest-path distances between vertices. A slight modification of this FastMap algorithm, presented in [4], can also be used to preserve *consistency* and *admissibility* of the Euclidean distance approximation, which is important when using it as a heuristic in A* search for shortest-path computations. In both [4] and [12], K is user-specified, but a threshold parameter ϵ is introduced to terminate with a smaller value of K once diminishing returns on the accuracy of approximating pairwise shortest-path distances are detected.

3 FastMap-Based Block Modeling Algorithm (FMBM)

In this section, we describe FMBM, our novel algorithm for block modeling based on FastMap [12]. As mentioned before, FMBM works in two phases. In the first phase, FMBM uses FastMap to efficiently embed vertices in a K-dimensional Euclidean space, preserving the probabilistically-amplified shortest-path distances between them. In the second phase, FMBM identifies the required blocks in the resulting Euclidean space using GMM clustering.

To facilitate the description of FMBM, we first examine what happens when FastMap is used naively in the first phase, i.e., when it is used to embed the vertices of a given undirected graph in a K-dimensional Euclidean space for preserving the pairwise shortest-path distances. This naive attempt fails even in relatively simple cases. For example, Figures 3a-3d show that it fails on a bipartite graph and a core-periphery graph. This is so because preserving the pairwise shortest-path distances in Euclidean space does not necessarily help GMM clustering to identify the two blocks (partitions). In fact, in a bipartite graph, the closest neighbors of a vertex are in the other partition.

[7] unless $|E| = O(|V|)$, in which case the complexity is near-linear in the size of the input because of the $\log |V|$ factor

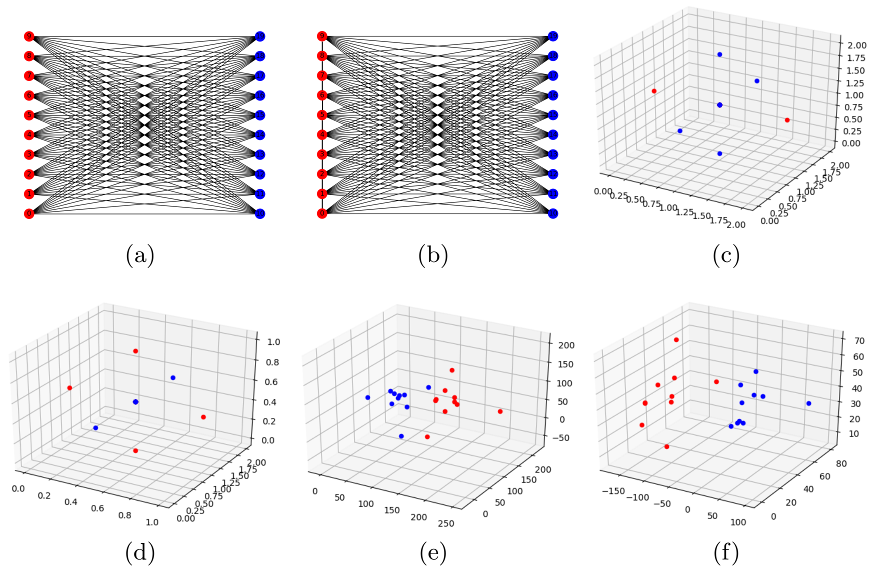


Fig. 3: The figure shows two simple graphs that guide the design of a proper FastMap distance function for block modeling. (a) shows a fully-connected bipartite graph with the red and blue vertices indicating the two partitions. (b) shows a core-periphery graph with the red vertices indicating the core and the blue vertices indicating the periphery. All pairs of red vertices are connected by edges (not all shown to avoid clutter). (c) and (d) show the FastMap Euclidean embeddings of the graphs in (a) and (b), respectively, using the shortest-path distance function. This naive FastMap distance function fails for block modeling. Red and blue points correspond to red and blue vertices of the graphs, respectively. Many vertices are mapped to the same point. (e) and (f) show the FastMap Euclidean embeddings of the graphs in (a) and (b), respectively, when using the probabilistically-amplified shortest-path distance function. This FastMap distance function is appropriate for block modeling.

3.1 Probabilistically-Amplified Shortest-Path Distances

From the foregoing discussion, it is clear that the shortest-path distance between two vertices v_i and v_j is not a viable distance function for block modeling. Therefore, in this subsection, we create a new distance function $D(v_i, v_j)$ for pairs of vertices based on the following 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*[8] $\bar{G}$ of the given graph G should yield the same distance function as G: The distance function should be independent of the arbitrary choice of representing a relationship between two vertices as either an

[8] The complement graph $\bar{G}$ has the same vertices as the original graph G but represents every edge in G as a non-edge and every non-edge in G as an edge.

edge or a non-edge. Intuitively, these guidelines capture an effective "resistance" between vertices and facilitate the subsequent embedding to represent relative "potentials" of vertices in Euclidean space. The effectiveness of these guidelines is validated through test cases in this section and comprehensive experiments in the next section.

We define a new distance function $D_P(v_i, v_j)$, referred to as the *probabilistically-amplified shortest-path distance* (PASPD) between v_i and v_j, as:

$$\sum_{\mathbb{G} \in G_{set}} d_{\mathbb{G}}(v_i, v_j). \tag{5}$$

Here, $d_{\mathbb{G}}(v_i, v_j)$ represents the shortest-path distance between v_i and v_j in an undirected graph $\mathbb{G}$. 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 $\bar{G}$.[9] The edge-induced subgraphs are created by probabilistically dropping edges from G or $\bar{G}$.

Intuitively, the use of shortest-path distances on multiple graphs that are probabilistically derived from the same input graph G accounts for $D_P(\cdot, \cdot)$. Indeed, the smaller $d_G(v_i, v_j)$, the smaller $D_P(v_i, v_j)$ also is. Similarly, the more paths between v_i and v_j in G, the more likely it is for such paths to survive in its edge-induced subgraphs, and the smaller $D_P(v_i, v_j)$ consequently is. Moreover, since the subgraphs in G_{set} are derived from both G and $\bar{G}$, $D_P(\cdot, \cdot)$ satisfies all these intuitive guidelines mentioned above. From an efficiency perspective, the use of multiple graphs does not create much overhead if the number of graphs does not depend on the size of G. However, $\bar{G}$ can have significantly more edges than G if G is sparse. In such cases, if G has n vertices and $m < \binom{n}{2}/2$ edges, $\bar{G}$ itself is probabilistically derived from G by randomly retaining only m out of the $\binom{n}{2} - m$ edges that it would otherwise have. This keeps the size of $\bar{G}$ upper-bounded by the size of the input.

Although more details on FMBM are presented in the next subsection, the benefits of using a probabilistically-amplified distance function are visually apparent in Figures 3e and 3f. In both cases, the red and blue vertices are mapped to linearly-separable red and blue points, respectively, in Euclidean space. Its benefits can also be seen in Figure 1b, where the core red vertices are mapped to a core set of red points and the peripheral blue vertices are mapped to a peripheral set of blue points, respectively, in Euclidean space. In this case, although the red and blue points are not linearly separable, GMM clustering [15] in the second phase of FMBM is capable of separating them using two overlapping but different Gaussian distributions.

3.2 Main Algorithm

Algorithm 1 shows the pseudocode for computing the PASPD function $D_P(\cdot, \cdot)$ parameterized by L and F. Like the shortest-path distance function, it, too, can

[9] An edge-induced subgraph of G has the same vertices as G but a subset of its edges.

Algorithm 1 SS-PASPD: Single-Source Probabilistically-Amplified Shortest-Path Distance Function

```
Input: $G = (V, E)$ and $v_s \in V$
Parameters: $L$ and $F$
Output: $d_{si}$ for each $v_i \in V$
 1: Let $\bar{G} = (V, \bar{E})$ be the complement graph of $G$.
 2: $G_{set} \leftarrow \{\}$ and $T_{set} \leftarrow \{\}$.
 3: for $l = 1, 2 \ldots L$ do
 4:     $\mathbb{G} \leftarrow G$ and $\bar{\mathbb{G}} \leftarrow \bar{G}$.
 5:     if $|\bar{E}| > |E|$ then
 6:         Drop $|\bar{E}| - |E|$ randomly chosen edges from $\bar{\mathbb{G}}$.
 7:     end if
 8:     $G_{set} \leftarrow G_{set} \cup \{\mathbb{G}\}$ and $f \leftarrow |E|/F$.
 9:     while $\mathbb{G}$ has edges do
10:         Drop $f$ randomly chosen edges from $\mathbb{G}$ to obtain $\hat{G}$.
11:         $G_{set} \leftarrow G_{set} \cup \{\hat{G}\}$.
12:         $\mathbb{G} \leftarrow \hat{G}$.
13:     end while
14:     Repeat lines 8-13 for $\bar{\mathbb{G}}$.
15: end for
16: for $G_i \in G_{set}$ do
17:     $T_i \leftarrow$ SS-SHORTESTPATHDISTANCE($G_i$, $v_s$).
18:     $T_{set} \leftarrow T_{set} \cup \{T_i\}$.
19: end for
20: for each $v_j \in V$ do
21:     $d_{sj} \leftarrow \sum_{T_i \in T_{set}} T_i(v_j)$.
22: end for
23: return $d_{si}$ for each $v_i \in V$.
```

be computed efficiently (in one shot) for all pairs (v_s, v_i), for a specified source v_s and all $v_i \in V$. On Lines 3-15, the algorithm populates G_{set} with L lineages of F nested edge-induced subgraphs of G and $\bar{G}$. On Lines 5-7, the algorithm constructs the complement graph $\bar{G}$ but probabilistically retains at most $|E|$ of its edges. On Lines 16-23, it uses the single-source shortest-path distance function to compute and return the sum of the shortest-path distances from v_s to v_i in all $\mathbb{G} \in G_{set}$, for all $v_i \in V$. If v_s and v_i are disconnected in any graph $\mathbb{G} \in G_{set}$, $d_{\mathbb{G}}(v_s, v_i)$ is technically equal to $+\infty$. However, for practical reasons in such cases, $d_{\mathbb{G}}(v_s, v_i)$ is set to twice the maximum shortest-path distance from v_s to any other vertex connected to it in $\mathbb{G}$. T_i on Line 17 refers to the array of shortest-path distances from v_s in G_i. $T_i(v_j)$ on Line 21 is the array element that corresponds to vertex v_j.

Algorithm 2 shows the pseudocode for FMBM. On Lines 3-25, it essentially implements FastMap as described in [12] but calls the SS-PASPD distance function in Algorithm 1 instead of the regular single-source shortest-path distance function. As in FastMap [12], K represents the user-specified upper bound on the dimensionality of the Euclidean embedding, ϵ represents the user-specified

Algorithm 2 FMBM: FastMap-Based Block Modeling

```
Input: $G = (V, E)$ and $k$
Parameters: $L$, $F$, $T$, $K$, $Q$, and $\epsilon$
Output: $c_i$ for each $v_i \in V$
 1: $MinObj \leftarrow +\infty$ and $BestC \leftarrow \emptyset$.
 2: for $t = 1, 2 \dots T$ do
 3:     for $r = 1, 2 \dots K$ do
 4:         Choose $v_a \in V$ randomly and let $v_b \leftarrow v_a$.
 5:         for $q = 1, 2 \dots Q$ do
 6:             $\{d_{ai} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_a$).
 7:             $v_c \leftarrow \text{argmax}_{v_i} \{d_{ai}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_i]_j)^2\}$.
 8:             if $v_c == v_b$ then
 9:                 Break.
10:             else
11:                 $v_b \leftarrow v_a$ and $v_a \leftarrow v_c$.
12:             end if
13:         end for
14:         $\{d_{ai} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_a$).
15:         $\{d_{ib} : v_i \in V\} \leftarrow$ SS-PASPD($G$, $v_b$).
16:         $d'_{ab} \leftarrow d_{ab}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_b]_j)^2$.
17:         if $d'_{ab} < \epsilon$ then
18:             Break.
19:         end if
20:         for each $v_i \in V$ do
21:             $d'_{ai} \leftarrow d_{ai}^2 - \sum_{j=1}^{r-1}([\vec{p}_a]_j - [\vec{p}_i]_j)^2$.
22:             $d'_{ib} \leftarrow d_{ib}^2 - \sum_{j=1}^{r-1}([\vec{p}_i]_j - [\vec{p}_b]_j)^2$.
23:             $[\vec{p}_i]_r \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$.
24:         end for
25:     end for
26:     $P \leftarrow [\vec{p}_1, \vec{p}_2 \dots \vec{p}_{|V|}]$.
27:     $C \leftarrow$ GMM($P$, $k$).
28:     $Obj \leftarrow$ GetObjectiveValue($G$, $C$).
29:     if $Obj \leq MinObj$ then
30:         $MinObj \leftarrow Obj$.
31:         $BestC \leftarrow C$.
32:     end if
33: end for
34: return $c_i$ for each $v_i \in V$ according to $BestC$.
```

threshold to recognize an accurate embedding, and Q represents a small constant number of pivot changes used to heuristically identify the farthest pair of vertices. L and F are simply passed to Algorithm 1 in the function call SS-PASPD. Because Algorithm 2 employs randomization, it qualifies as a Monte-Carlo algorithm. It implements an outer loop to boost the performance of FMBM using T independent trials. On Lines 26-32, each trial invokes the GMM clustering algorithm and evaluates the results on the objective function in Equation 2,[10] keeping record of the best value. The results of the best trial, i.e., the block assignment c_i for each $v_i \in V$, are returned on Line 34.[11]

A formal time complexity analysis of FMBM is evasive since Line 27 of Algorithm 2 calls the GMM clustering procedure, which has no defined time complexity. Therefore, we only claim to be able to reformulate the block modeling problem on graphs to its Euclidean version in $O(LFK(|E| + |V| \log |V|))$ time in each of the T iterations. Here, the factor LF comes from the cardinality of G_{set} in Algorithm 1, and the factor $K(|E|+|V| \log |V|)$ comes from the complexity of FastMap, that uses SS-PASPD on Lines 3-25 of Algorithm 2. The time complexity of GetObjectiveValue on Line 28 is technically $O(|V|^2 k+|V|k^2)$, where k is the user-specified number of blocks, also passed to the GMM clustering algorithm. This time complexity comes from the matrix multiplication $CMC^\top$ in Equation 2. The factor $|V|^2$ in this matrix multiplication, and more generally in Equation 2, can be reduced to $O(|E|)$ by evaluating $|E|$ entries corresponding to edges and $\min(|E|, \binom{|V|}{2} - |E|)$ randomly chosen entries corresponding to non-edges in the matrix expression $(A - CMC^\top) \circ (A - R)$. The matrix multiplication CM takes $O(|V|k^2)$ time and results in a $|V| \times k$ matrix. $|E|+\min(|E|, \binom{|V|}{2} - |E|)$ entries in the multiplication of this matrix with $C^\top$ can be computed in $O(|E|k)$ time. Overall, therefore, the reformulation to Euclidean space can be done in near-linear time, i.e., linear in $|V|$ and $|E|$, after ignoring logarithmic factors.

4 Experiments

In this section, we present empirical results on the comparative performances of FMBM and three other state-of-the-art solvers for block modeling: Graph-Tool, DANMF, and CPLNS. We also compared against two other solvers for block modeling: FactorBlock [3] and ASBlock [19]. However, they are not competitive with the other solvers; and we exclude them from Tables 1, 2, 3, 4, and 5 to save column space. Graph-Tool [17] uses an agglomerative multi-level Markov Chain Monte Carlo algorithm and has been largely ignored in the computer science literature on block modeling; DANMF [20] uses deep autoencoders; and CPLNS [14] uses constraint programming with large neighborhood search.

We used the following hyperparameter values for FMBM:[12] $L = 4$, $F = 10$, $T = 10$, $K = 4$, $Q = 10$, and $\epsilon = 10^{-4}$. The value of k, i.e., the number of blocks,

[10] M can be computed from A and C in $O(|E|+k^2)$ time while evaluating the objective function in Equation 2.

[11] The domain of each c_i is $\{1, 2 \dots k\}$. Block $\mathcal{B}_h$ refers to the collection of all vertices $v_i \in V$ such that $c_i = h$.

[12] These values are only important as ballpark estimates. We observed that the performance of FMBM often stays stable within broad ranges of hyperparameter values, imparting robustness to FMBM. Moreover, only a few different hyperparameter settings had to be examined to determine the best one.

Test Case	Size ($\|V\|$, $\|E\|$)	FMBM			Graph-Tool			DANMF			CPLNS		
		Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time
adjnoun	(112, 425)	616.86	0.0025	6.11	612.98	**0.2978**	**0.04**	636.75	0.0083	1.62	**591.76**	0.0154	1.51
baboons	(14, 23)	11.97	0.0158	0.54	**11.49**	**0.2244**	**0.00**	15.49	0.1341	0.97	12.81	0.0172	0.87
football	(115, 613)	665.97	0.5608	9.22	**343.32**	**0.9150**	**0.03**	863.91	0.2574	1.55	558.94	0.6991	83.33
karate	(34, 78)	74.66	**0.6127**	1.47	**64.67**	0.2512	**0.00**	81.94	0.1672	0.77	75.43	0.2228	1.06
polblogs	(1,490, 16,715)	98,788.53	0.0098	239.33	99,014.21	**0.4668**	**2.14**	101,195.89	0.0465	404.02	**95,859.73**	0.0543	506.29
polbooks	(105, 441)	522.33	0.5329	6.20	**496.02**	**0.5462**	**0.02**	590.20	0.3177	1.98	531.48	0.2073	2.09

Table 1: Real-World Single-View Undirected Graphs.

Test Case	Size ($\|V\|$, $\|E\|$)	FMBM			Graph-Tool			DANMF			CPLNS		
		Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time
adjnoun	(112, 5,791)	611.04	0.0048	25.34	636.34	0.0168	**0.41**	641.40	0.0000	8.34	**591.54**	**0.0169**	1.85
baboons	(14, 68)	**12.64**	**0.0547**	0.62	12.86	0.0416	**0.01**	15.46	0.0500	1.23	13.35	0.0316	0.86
football	(115, 5,944)	595.52	0.5899	27.53	**344.38**	**0.9111**	**0.17**	815.71	0.2229	9.56	525.54	0.7040	82.11
karate	(34, 483)	**72.73**	**0.7625**	2.46	77.31	0.2065	**0.02**	84.23	0.0914	1.78	75.00	0.2439	1.04
polblogs	(1,490, 1,094,951)	26,896.52	0.0153	3155.33	26,048.04	0.0454	**49.90**	-	-	> 1 hour	**25,871.42**	**0.0541**	470.48
polbooks	(105, 5,019)	**509.88**	**0.5409**	21.55	606.96	0.0867	**0.13**	631.77	0.0141	8.09	531.65	0.2056	2.15

Table 2: Complement Graphs of the Graphs in Table 1.

Test Case	Size ($\|E\|$)	FMBM			Graph-Tool			DANMF			CPLNS		
		Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time
V0400b04	6,722	9,355.91	0.1622	83.22	**8,385.61**	**0.6565**	**1.49**	9,465.48	0.0111	10.88	9,400.46	0.0534	39.74
V0800b04	14,723	24,201.35	0.1775	199.14	**22,848.79**	**0.6599**	**3.99**	24,387.24	0.0019	62.50	24,303.43	0.0394	195.08
V1600b04	25,103	45,849.52	0.2357	391.92	**44,598.02**	**0.9667**	**7.04**	46,379.74	0.0043	753.62	46,292.59	0.0206	1018.54
V3200b04	70,973	134,217.82	0.0348	1,376.41	**131,751.34**	**0.6654**	**108.08**	-	-	> 1 hour	-	-	> 1 hour
V0400b10	3,246	5,461.99	0.1217	40.9	**4,728.69**	**0.8542**	**0.63**	5,489.8	0.0509	7.25	5,364.07	0.1289	101.01
V0800b10	7,499	13,623.69	0.0425	108.69	**12,612.44**	**0.9596**	**2.01**	13,636.29	0.0156	47.49	13,485.93	0.0734	423.03
V1600b10	15,118	28,782.35	0.0691	272.65	**27,828.70**	**0.8556**	**4.82**	28,829.58	0.0117	537.27	28,682.31	0.0384	2019.76
V3200b10	36,170	70,292.36	0.0369	782.44	**68,653.51**	**0.9173**	**17.62**	70,315.35	0.0074	3,272.65	-	-	> 1 hour
V0400b20	2,297	4,048.03	0.1639	29.66	**3,632.92**	**0.6256**	**0.54**	4,064.77	0.1859	8.03	-	-	> 1 hour
V0800b20	5,049	9,451.30	0.0848	72.92	**8,960.16**	**0.5857**	**1.90**	9,460.80	0.0828	62.32	-	-	> 1 hour
V1600b20	11,575	22,305.01	0.0457	251.06	**21,591.83**	**0.6718**	**3.91**	22,315.63	0.0445	444.49	-	-	> 1 hour
V3200b20	24,639	48,321.14	0.0212	579.90	**47,650.73**	**0.6067**	**12.89**	-	-	> 1 hour	-	-	> 1 hour

Table 3: Sparse Single-View Undirected Graphs Using Generative Model 1.

Test Case	Size ($\|E\|$)	FMBM			Graph-Tool			DANMF			CPLNS		
		Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time
V0100b06	4,125	815.58	0.1308	20.06	818.83	0.1291	**0.46**	829.20	0.0829	5.85	**750.28**	**0.2727**	3.50
V0300b06	41,771	**4,702.6**	**0.2061**	176.65	4,819.43	0.0311	**1.93**	4,828.49	0.0149	143.10	4,746.32	0.0629	18.14
V0500b06	118,536	10,473.0	**0.0537**	513.31	10,489.79	0.0193	**4.86**	10,497.63	0.0053	658.45	**10,419.98**	0.0411	41.13
V0100b08	4,212	782.91	0.2182	19.23	761.68	0.3024	**0.33**	807.05	0.1532	6.02	**712.92**	**0.3331**	4.59
V0300b08	42,078	4,468.67	**0.0944**	175.55	4,487.12	0.0443	**1.48**	4,498.23	0.0151	144.96	**4,397.28**	0.0895	19.95
V0500b08	120,166	8,188.55	**0.1503**	505.41	8,278.07	0.0267	**4.94**	8,290.82	0.0056	680.16	**8,175.52**	0.0582	59.04
V0100b10	4,268	731.61	0.3446	19.22	720.38	0.3290	**0.32**	779.34	0.1570	6.41	**662.42**	**0.4100**	6.93
V0300b10	42,385	4,069.42	**0.1652**	171.93	4,126.01	0.0569	**1.47**	4,141.94	0.0292	146.23	**4,009.88**	0.1160	29.28
V0500b10	120,366	7,931.97	**0.1174**	516.16	7,985.47	0.0347	**4.03**	8,000.60	0.0000	616.74	**7,872.77**	0.0761	60.22

Table 4: Dense Single-View Undirected Graphs Using Generative Model 1.

was given as input for all the solvers in the experiments.[13] We used three metrics for comparison: the value of the objective function stated in Equation 2, the Normalized Mutual Information (NMI) value with respect to the ground truth, and the running time in seconds. Unlike other methods, FMBM is an anytime algorithm since it uses multiple trials. Each trial takes roughly $(1/T)$'th, i.e., one-tenth, of the time reported for FMBM in the experimental results. For each

[13] Although Graph-Tool does not require a user-specified value of k, it has a tendency to produce trivial solutions with $k = 1$, resulting in 0 NMI values when the value of k is not explicitly specified.

Test Case	Size ($\lvert E \rvert$)	FMBM			Graph-Tool			DANMF			CPLNS		
		Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time	Objective	NMI	Time
V0400b04	7,176	9,843.84	0.0327	75.09	**9,444.48**	**0.8214**	**1.77**	9,855.97	0.0116	12.23	9,786.67	0.0246	34.55
V0800b04	16,780	27,040.21	0.0087	192.97	26,982.65	**0.0698**	**5.67**	27,053.24	0.0008	69.68	**26,945.04**	0.0087	243.28
V1600b04	28,183	51,531.41	**0.0146**	386.32	51,556.38	0.0043	**8.04**	51,559.96	0.0025	684.58	**51,467.00**	0.0073	1,098.12
V3200b04	72,182	136,376.53	0.0087	1,196.30	**135,967.24**	**0.5287**	**32.92**	-	-	> 1 hour	-	-	> 1 hour
V0400b10	7,069	9,729.49	0.0639	73.76	**9,349.76**	**0.4827**	**1.52**	9,753.04	0.0625	10.69	9,585.60	0.0837	127.68
V0800b10	15,613	25,528.33	0.0385	181.56	**25,022.25**	**0.4893**	**4.05**	25,553.52	0.0274	86.58	25,353.08	0.0418	494.10
V1600b10	32,294	58,279.85	0.0195	428.40	58,244.14	**0.0305**	**13.92**	58,305.57	0.0157	687.84	**58,103.92**	0.0224	2459.38
V3200b10	79,173	**148,741.84**	**0.0082**	1,307.21	148,745.65	0.0065	**32.88**	-	-	> 1 hour	-	-	> 1 hour
V0400b20	6,829	9,453.81	0.1425	72.78	**9,139.14**	**0.2039**	**1.38**	9,506.35	0.1790	9.98	-	-	> 1 hour
V0800b20	15,106	24,810.19	0.0784	176.60	**24,521.35**	**0.1251**	**3.49**	24,866.54	0.0918	62.93	-	-	> 1 hour
V1600b20	30,462	55,265.42	0.0436	420.49	**55,207.39**	0.0366	**9.38**	55,310.7	**0.0440**	606.39	-	-	> 1 hour
V3200b20	67,675	128,267.45	**0.0232**	1,199.32	**128,234.10**	0.0168	**40.91**	-	-	> 1 hour	-	-	> 1 hour

Table 5: Single-View Undirected Graphs Using Generative Model 2.

method and test case, we averaged the results over 10 runs. All experiments were conducted on a laptop with a 3.1GHz Quad-Core Intel Core i7 processor and 16GB LPDDR3 memory. Our implementation of FMBM was done in Python3 with NetworkX [10].

Although the underlying theory of FMBM can be generalized to directed graphs with weighted edges [9] and to multi-view graphs, the current version of FMBM is operational only on singe-view undirected graphs, sufficient to illustrate the power of FastMap embeddings. Therefore, only such test cases are borrowed from other commonly used datasets [16,19]. However, we also created new synthetic test cases to be able to do a more comprehensive analysis.[14]

The synthetic test cases were generated according to two similar stochastic block models [1] as follows. In Generative Model 1, given a user-specified number of vertices $|V|$ and a user-specified number of blocks k, we first assign each vertex to a block chosen uniformly at random to obtain the membership matrix C, representing the ground truth. The image matrix M is drafted using certain "block structural characteristics" designed for that instance with a parameter p. Each entry M_{ij} is set to either p or $10p$ according to a rule explained below. If M_{ij} is set to p ($10p$), the two blocks $\mathcal{B}_i$ and $\mathcal{B}_j$ are weakly (strongly) connected to each other with respect to p. The adjacency matrix A, representing the entire graph, is constructed from C and M by connecting any two vertices $v_s \in \mathcal{B}_i$ and $v_t \in \mathcal{B}_j$ with probability M_{ij}. In Generative Model 2, each entry M_{ij} is set to cp, where c is an integer chosen uniformly at random from the interval $[1, 10]$.

Tables 1, 2, 3, 4, and 5 show the comparative performances of FMBM, Graph-Tool, DANMF, and CPLNS.[15] Table 1 contains commonly used real-world test cases from [16] and [19]. Here, FMBM outperforms DANMF and CPLNS with respect to the value of the objective function on 3 out of 6 instances, despite the fact that it uses the expression in Equation 2 only for evaluation on Line 28 of Algorithm 2. Graph-Tool performs well on all the instances. Table 2 shows the comparative performances on the complement graphs of the graphs in Table 1. This is done to test the robustness of the solvers against encoding the same

[14] https://github.com/leon-angli/Synthetic-Block-Modeling-Dataset

[15] DANMF did not assign any block membership to a few vertices in some synthetic test cases. We assign Block $\mathcal{B}_1$ by default to such vertices.

relationships between vertices as either edges or non-edges. While the value of the objective function and the running time are expected to change, the NMI value is expected to be stable. We observe that FMBM and CPLNS are the only solvers that convincingly pass this test. Moreover, FMBM outperforms the other solvers on more instances than in Table 1. Tables 1 and 2 do not test scalability since $|V|$ is small in these test cases.

Table 3 contains synthetic sparse test cases from Generative Model 1, named "Vnbk", where n indicates the number of vertices and k indicates the number of blocks. These test cases have the following block structural characteristics. Each block is strongly connected to two other randomly chosen blocks and weakly connected to the remaining ones (including itself). We set $p = (\ln|V|)/|V|$, making $|E| = O(|V| \log |V|)$ in expectation. After generating A, we also add some noise to it by flipping each of its entries independently with probability $0.05/|V|$. FMBM outperforms DANMF and CPLNS with respect to both the value of the objective function and the NMI value on 8 out of 12 instances. We also begin to see FMBM's advantages in scalability. However, Graph-Tool outperforms all other methods by a significant margin on all the instances. Table 4 contains synthetic dense test cases from Generative Model 1 constructed by setting $p = (\ln|V|)/|V|$, modifying each entry M_{ij} to $1 - M_{ij}$, and adding noise, as before. We observe that the performance of Graph-Tool is poor on such dense graphs. FMBM outperforms DANMF and CPLNS with respect to the NMI value on 6 out of 9 instances. Although CPLNS produces marginally better values of the objective function, its performance on large sparse graphs in Table 3 is bad.

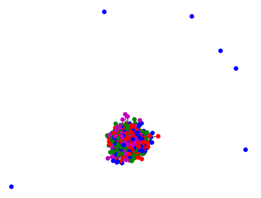
Table 5 contains synthetic test cases from Generative Model 2 constructed by setting $p = (\ln|V|)/|V|$. FMBM outperforms DANMF and CPLNS with respect to the value of the objective function on 6 out of 12 instances. It also outperforms DANMF and CPLNS with respect to the NMI value on a different set of 6 instances. Graph-Tool performs comparatively well on all the instances but occasionally produces low NMI values.

4.1 Visualization

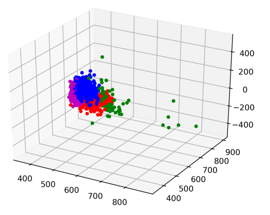
In addition to identifying blocks, their visualization is important for uncovering trends, patterns, and outliers in large graphs. A good visualization aids human intuition for gauging the spread[16] of blocks, both individually and relative to each other. In market analysis, for example, a representative element can be chosen from each block with proper visualization. Figure 4 shows that FMBM provides a much more perspicuous visualization compared to a standard graph visualization procedure in NetworkX[17] used with Graph-Tool, even though Graph-Tool shows good overall performance in Tables 1, 3, and 5. This is so because FMBM solves the block modeling problem in Euclidean space, while other approaches use abstract methods that are harder to visualize.

[16] The spread here refers to how a block extends from its center to its periphery.

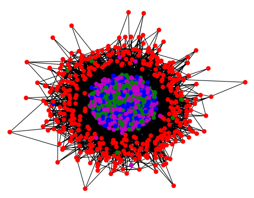
[17] https://networkx.org/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw.html



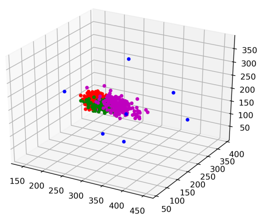
(a) standard graph visualization of blocks in an instance with $1,600$ vertices and $4,353$ edges



(b) FMBM visualization of blocks in Euclidean space for the instance from 4a



(c) standard graph visualization of blocks in an instance with $1,600$ vertices and $25,103$ edges



(d) FMBM visualization of blocks in Euclidean space for the instance from 4c

Fig. 4: The left column shows a visualization of two different instances with four blocks obtained by a standard graph visualization procedure in NetworkX used with Graph-Tool. The right column shows a visualization of the same two instances obtained in the Euclidean embedding by FMBM. Four different colors are used to indicate the four different blocks. The FMBM visualization is more helpful for gauging the spread of blocks, both individually and relative to each other.

5 Conclusions and Future Work

In this paper, we proposed FMBM, a FastMap-based algorithm for block modeling. In the first phase, FMBM adapts FastMap to embed a given undirected graph into a Euclidean space in near-linear time such that the pairwise Euclidean distances between vertices approximate a probabilistically-amplified graph-based distance function between them. In doing so, it avoids having to directly work on the given graphs and instead reformulates the graph block modeling problem to a Euclidean version. In the second phase, FMBM uses GMM clustering for identifying clusters (blocks) in the resulting Euclidean space. Empirically, FMBM outperforms other state-of-the-art methods like FactorBlock, Graph-Tool, DANMF, and CPLNS on many benchmark and synthetic test cases. FMBM also enables a perspicuous visualization of blocks in the graphs, not provided by other methods.

In future work, we will generalize FMBM to work on directed graphs and multi-view graphs. We will also apply FMBM and its generalizations to real-world graphs from various domains, including social and biological networks.

References

1. Abbe, E.: Community detection and stochastic block models: Recent developments. Journal of Machine Learning Research (2017)
2. Antonopoulos, C.G.: Dynamic range in the C. elegans brain network. Chaos: An Interdisciplinary Journal of Nonlinear Science (2016)
3. Chan, J., Liu, W., Kan, A., Leckie, C., Bailey, J., Ramamohanarao, K.: Discovering latent blockmodels in sparse and noisy graphs using non-negative matrix factorisation. In: Proceedings of the ACM International Conference on Information & Knowledge Management (2013)
4. Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., Kumar, T.K.S.: The FastMap algorithm for shortest path computations. In: Proceedings of the International Joint Conference on Artificial Intelligence (2018)
5. Davis, T.: USAir97 (2014), https://www.cise.ufl.edu/research/sparse/matrices/Pajek/USAir97
6. Faloutsos, C., Lin, K.I.: 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 (1995)
7. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (JACM) (1987)
8. Girvan, M., Newman, M.E.: Community structure in social and biological networks. National Academy of Sciences (2002)
9. Gopalakrishnan, S., Cohen, L., Koenig, S., Kumar, T.K.S.: Embedding directed graphs in potential fields using FastMap-D. In: Proceedings of the International Symposium on Combinatorial Search (2020)
10. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using NetworkX. Tech. rep., Los Alamos National Lab, Los Alamos, NM (United States) (2008)
11. Lee, J., Gross, S.P., Lee, J.: Improved network community structure improves function prediction. Scientific Reports (2013)
12. Li, J., Felner, A., Koenig, S., Kumar, T.K.S.: Using FastMap to solve graph problems in a Euclidean space. In: Proceedings of the International Conference on Automated Planning and Scheduling (2019)
13. Lin, S., Hu, Q., Wang, G., Yu, P.S.: Understanding community effects on information diffusion. In: Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (2015)
14. Mattenet, A., Davidson, I., Nijssen, S., Schaus, P.: Generic constraint-based block modeling using constraint programming. Journal of Artificial Intelligence Research (2021)
15. Murphy, K.P.: Machine Learning: A probabilistic perspective. The MIT Press (2012)
16. Newman, M.E.: Finding community structure in networks using the eigenvectors of matrices. Physical Review E (2006)
17. Peixoto, T.P.: Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models. Physical Review E (2014)
18. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online learning of social representations. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)

19. Ramteke, R., Stuckey, P.J., Chan, J., Ramamohanarao, K., Bailey, J., Leckie, C., Demirović, E.: Improving single and multi-view blockmodelling by algebraic simplification. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN) (2020)
20. Ye, F., Chen, C., Zheng, Z.: Deep autoencoder-like nonnegative matrix factorization for community detection. In: Proceedings of the ACM International Conference on Information and Knowledge Management (2018)