A FastMap-Based Framework for Efficiently Computing Top-K Projected Centrality

Ang Li[1], Peter Stuckey[2,3], 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
[3] OPTIMA ARC Industrial Training and Transformation Centre, Melbourne, Australia

Abstract. In graph theory and network analysis, various measures of centrality are used to characterize the importance of vertices in a graph. Although different measures of centrality have been invented to suit the nature and requirements of different underlying problem domains, their application is restricted to explicit graphs. In this paper, we first define implicit graphs that involve auxiliary vertices in addition to the pertinent vertices. We then generalize the various measures of centrality on explicit graphs to corresponding measures of projected centrality on implicit graphs. We also propose a unifying framework for approximately, but very efficiently computing the top-K pertinent vertices in implicit graphs for various measures of projected centrality. Our framework is based on FastMap, a graph embedding algorithm that 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. Using FastMap's ability to facilitate geometric interpretations and analytical procedures in Euclidean space, we show that the top-K vertices for many popularly used measures of centrality—and their generalizations to projected centrality—can be computed very efficiently in our framework.

Keywords: Projected Centrality · FastMap · Graph Embedding.

1 Introduction

Graphs are used to represent entities in a domain and important relationships between them: Often, vertices represent the entities and edges represent the relationships. However, graphs can also be defined implicitly by using two kinds of vertices and edges between the vertices: The *pertinent* vertices represent the main entities, i.e., the entities of interest; the *auxiliary* vertices represent the hidden entities; and the edges represent relationships between the vertices. For example, in an air transportation domain, the pertinent vertices could represent international airports, the auxiliary vertices could represent domestic airports, and the edges could represent flight connections between the airports. In a social

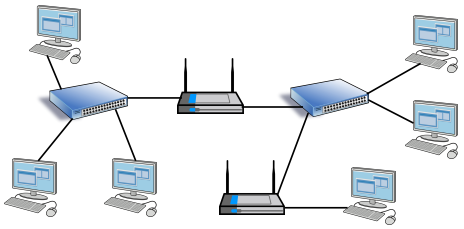


Fig. 1: Shows a communication network with user terminals, routers, and switches; solid lines show direct communication links. Depending on the application, the user terminals may be considered as the pertinent vertices while the routers and the switches may be considered as the auxiliary vertices.

network, the pertinent vertices could represent individuals, the auxiliary vertices could represent communities, and the edges could represent friendships or memberships. In a communication network, as shown in Figure 1, the pertinent vertices could represent user terminals, the auxiliary vertices could represent routers and switches, and the edges could represent direct communication links.

Explicit and implicit graphs can be used to model transportation networks, social networks, communication networks, and biological networks, among many others. In most of these domains, the ability to identify the "important" pertinent vertices has many applications. For example, the important pertinent vertices in an air transportation network could represent transportation hubs, such as Amsterdam and Los Angeles for Delta Airlines. The important pertinent vertices in a social network could represent highly influential individuals. Similarly, the important pertinent vertices in a communication network could represent admin-users, and the important pertinent vertices in a properly modeled biological network could represent biochemicals critical for cellular operations.

The important pertinent vertices in a graph (network) as well as the task of identifying them depend on the definition of "importance". Such a definition is typically domain-specific. It has been studied in explicit graphs and is referred to as a *measure of centrality*. For example, the *page rank* is a popular measure of centrality used in Internet search engines [20]. In general, there are several other measures of centrality defined on explicit graphs, such as the *degree centrality*, the *closeness centrality* [13], the *harmonic centrality* [2], the *current-flow closeness centrality* [22,5], the *eigenvector centrality* [3], and the *Katz centrality* [16].

The degree centrality of a vertex measures the immediate connectivity of it, i.e., the number of its neighbors. The closeness centrality of a vertex is the reciprocal of the average shortest path distance between that vertex and all other vertices. The harmonic centrality resembles the closeness centrality but reverses the sum and reciprocal operations in its mathematical definition to be able to handle disconnected vertices and infinite distances. The current-flow closeness centrality also resembles the closeness centrality but uses an "effective resistance" between two vertices instead of the shortest path distance between them. The eigenvector centrality scores the vertices based on the eigenvector corresponding

to the largest eigenvalue of the adjacency matrix. The Katz centrality generalizes the degree centrality by incorporating a vertex's k-hop neighbors with a weight α^k, where $\alpha \in (0, 1)$ is an attenuation factor.

While many measures of centrality are frequently used on explicit graphs, they are not frequently used on implicit graphs. However, for any measure of centrality, a measure of "projected" centrality can be defined on implicit graphs. The measure of projected centrality is equivalent to the regular measure of centrality applied on a graph that "factors out" the auxiliary vertices from the implicit graph. Auxiliary vertices can be factored out by conceptualizing a clique on the pertinent vertices, in which an edge connecting two pertinent vertices is annotated with a graph-based distance between them that, in turn, is derived from the implicit graph.[4] The graph-based distance can be the shortest path distance or any other domain-specific distance. If there are no auxiliary vertices, the graph-based distance function is expected to be such that the measure of projected centrality reduces to the regular measure of centrality.

For a given measure of projected centrality, the projected centrality of a pertinent vertex is referred to as its *projected centrality value*. Identifying the important pertinent vertices in a network is equivalent to identifying the top-K pertinent vertices with the highest projected centrality values. Graph theoretically, the projected centrality values can be computed for all pertinent vertices of a network in polynomial time, for most measures of projected centrality. However, in practical domains, the real challenge is to achieve scalability to very large networks with millions of vertices and hundreds of millions of edges. Therefore, algorithms with a running time that is quadratic or more in the size of the input are undesirable. In fact, algorithms with any super-linear running times, discounting logarithmic factors, are also largely undesirable. In other words, modulo logarithmic factors, a desired algorithm should have a near-linear running time close to that of merely reading the input.

Although attempts to achieve such near-linear running times exist, they are applicable only for certain measures of centrality on explicit graphs. For example, [9,6] approximate the closeness centrality using sampling-based procedures. [1] maintains and updates a lower bound for each vertex, utilizing the bound to skip the analysis of a vertex when appropriate. It supports fairly efficient approximation algorithms for computing the top-K vertices for the closeness and the harmonic centrality measures. [4] provides a survey on such approximation algorithms. However, algorithms of the aforementioned kind are known only for a few measures of centrality on explicit graphs. Moreover, they do not provide a general framework since they are tied to specific measures of centrality.

In this paper, we generalize the various measures of centrality on explicit graphs to corresponding measures of projected centrality on implicit graphs. Importantly, we also propose a framework for computing the top-K pertinent vertices approximately, but very efficiently, using a graph embedding algorithm

[4] The clique on the pertinent vertices is a mere conceptualization. Constructing it explicitly may be prohibitively expensive for large graphs since it requires the computation of the graph-based distance between every pair of the pertinent vertices.

called FastMap [7,18], for various measures of projected centrality. 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 essence, the FastMap framework allows us to conceptualize the various measures of centrality and projected centrality in Euclidean space. In turn, the Euclidean space facilitates a variety of geometric and analytical techniques for efficiently computing the top-K pertinent vertices. The FastMap framework is extremely valuable because it implements this reformulation for different measures of projected centrality in only near-linear time and delegates the combinatorial heavy-lifting to analytical techniques that are better equipped for efficiently absorbing large input sizes.

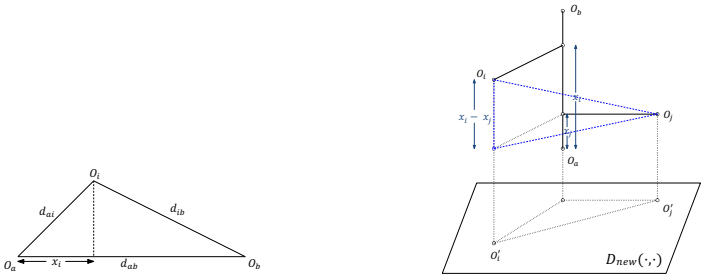
Computing the top-K pertinent vertices in the FastMap framework for different measures of projected centrality often requires interpreting analytical solutions found in the FastMap embedding back in the original graphical space. We achieve this via nearest-neighbor queries and Locality Sensitive Hashing (LSH) [8]. Through experimental results on a comprehensive set of benchmark and synthetic instances, we show that the FastMap+LSH framework is both efficient and effective for many popular measures of centrality and their generalizations to projected centrality. For our experiments, we also implement generalizations of some competing algorithms on implicit graphs. Overall, our approach demonstrates the benefits of drawing power from analytical techniques via FastMap.

2 Background: FastMap

FastMap [10] was introduced in the Data Mining community for automatically generating Euclidean embeddings of abstract objects. For many real-world objects such as long DNA strings, multi-media datasets like voice excerpts or images, 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[5] 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 κ-dimensional point $p_i \in \mathbb{R}^\kappa$ to each object O_i. A good Euclidean embedding is one in which the Euclidean distance χ_{ij} between any two points p_i and p_j closely approximates $D(O_i, O_j)$. For $p_i = ([p_i]_1, [p_i]_2 \dots [p_i]_\kappa)$ and $p_j = ([p_j]_1, [p_j]_2 \dots [p_j]_\kappa)$, $\chi_{ij} = \sqrt{\sum_{r=1}^{\kappa} ([p_j]_r - [p_i]_r)^2}$.

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



(a) the "cosine law" projection in a triangle

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

Fig. 2: Illustrates how coordinates are computed and recursion is carried out in FastMap, borrowed from [7].

FastMap creates a κ-dimensional Euclidean embedding of the abstract objects in $\mathcal{O}$, for a user-specified value of κ. In the very first iteration, FastMap heuristically identifies the farthest pair of objects O_a and O_b in linear time. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$ and $d_{ib} = D(O_i, O_b)$, as shown in Figure 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

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}). \tag{1}$$

FastMap sets the first coordinate of p_i, the embedding of O_i, to x_i. In the subsequent $\kappa - 1$ iterations, the same procedure is followed for computing the remaining $\kappa - 1$ coordinates of each object. However, the distance function is adapted for different iterations. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab}, respectively. Because these coordinates fully explain the true domain-specific distance between these two objects, from the second iteration onward, the rest of p_a and p_b's coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line $\overline{O_aO_b}$, as shown in Figure 2b. Although the hyperplane is never constructed explicitly, its conceptualization implies that the distance function for the second iteration should be changed for all i and j in the following way:

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

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 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

Algorithm 1 FastMap: A near-linear-time graph embedding algorithm.

Input: $G = (V, E)$, κ, and ϵ
Output: $p_i \in \mathbb{R}^r$ for all $v_i \in V$

for $r = 1, 2 \ldots \kappa$ **do**
  Choose $v_a \in V$ randomly and let $v_b = v_a$.
  for $t = 1, 2 \ldots C$ (a small constant) **do**
    $\{d_{ai} : v_i \in V\} \leftarrow \text{ShortestPathTree}(G, v_a)$.
    $v_c \leftarrow \text{argmax}_{v_i}\{d_{ai}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_i]_j)^2\}$.
    if $v_c == v_b$ **then**
      Break.
    else
      $v_b \leftarrow v_a$; $v_a \leftarrow v_c$.
    end if
  end for
  $\{d_{ai} : v_i \in V\} \leftarrow \text{ShortestPathTree}(G, v_a)$.
  $\{d_{ib} : v_i \in V\} \leftarrow \text{ShortestPathTree}(G, v_b)$.
  $d'_{ab} \leftarrow d_{ab}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_b]_j)^2$.
  if $d'_{ab} < \epsilon$ **then**
    $r \leftarrow r - 1$; Break.
  end if
  for each $v_i \in V$ **do**
    $d'_{ai} \leftarrow d_{ai}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_i]_j)^2$.
    $d'_{ib} \leftarrow d_{ib}^2 - \sum_{j=1}^{r-1}([p_i]_j - [p_b]_j)^2$.
    $[p_i]_r \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$.
  end for
end for
return $p_i \in \mathbb{R}^r$ for all $v_i \in V$.

for generating an embedding in linear time. This is 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. However, computing the shortest path distance between two vertices depends on the size of the graph.

The issue of having to retain (near-)linear time complexity can be addressed 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 [12]. 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 shortest path 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[6]. The amortized complexity for computing d_{ai} and d_{ib} for any single vertex O_i is therefore near-constant time.

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

The foregoing observations are used in [18] 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. Algorithm 1 presents the pseudocode for this algorithm. Here, κ is user-specified, but a threshold parameter ϵ is introduced to detect large values of κ that have diminishing returns on the accuracy of approximating pairwise shortest path distances.

3 Measures of Projected Centrality

In this section, we generalize measures of centrality to corresponding measures of projected centrality. Consider an implicit graph $G = (V, E)$, where $V^P \subseteq V$ and $V^A \subseteq V$, for $V^P \cup V^A = V$ and $V^P \cap V^A = \emptyset$, are the pertinent vertices and the auxiliary vertices, respectively. We define a graph $G^P = (V^P, E^P)$, where, for any two distinct vertices $v_i^P, v_j^P \in V^P$, the edge $(v_i^P, v_j^P) \in E^P$ is annotated with the weight $\mathcal{D}_G(v_i^P, v_j^P)$. Here, $\mathcal{D}_G(\cdot, \cdot)$ is a distance function defined on pairs of vertices in G. For any measure of centrality $\mathcal{M}$ defined on explicit graphs, an equivalent measure of projected centrality $\mathcal{M}^P$ can be defined on implicit graphs as follows: $\mathcal{M}^P$ on G is equivalent to $\mathcal{M}$ on G^P.

The distance function $\mathcal{D}_G(\cdot, \cdot)$ can be the shortest path distance function or any other domain-specific distance function. If it is a graph-based distance function, computing it would typically require the consideration of the entire graph G, including the auxiliary vertices V^A. For example, computing the shortest path distance between v_i^P and v_j^P in V^P requires us to utilize the entire graph G. Other graph-based distance functions are the probabilistically-amplified shortest path distance (PASPD) function [17] and the effective resistance between two vertices when interpreting the non-negative weights on edges as electrical resistance values.

4 FastMap for Top-K Projected Centrality

In this section, we show how to use the FastMap framework, coupled with LSH, for efficiently computing the top-K pertinent vertices with the highest projected centrality values in a given graph (network), for various measures of projected centrality. This subsumes the task of efficiently computing the top-K vertices in explicit graphs, for various regular measures of centrality. We note that the FastMap framework is applicable as a general paradigm, independent of the measure of projected centrality: The measure of projected centrality that is specific to the problem domain affects only the distance function used in the FastMap embedding and the analytical techniques that work on it. In other words, the FastMap framework allows us to interpret and reason about the various measures of projected centrality by invoking the power of analytical techniques. This is in stark contrast to other approaches that are tailored to a specific measure of centrality or its corresponding measure of projected centrality.

In the FastMap framework, any point of interest computed analytically in the Euclidean embedding may not map to a vertex in the original graph. Therefore, we use LSH [8] to find the point closest to the point of interest that corresponds to a vertex. In fact, LSH answers nearest-neighbor queries very efficiently in near-logarithmic time. It also efficiently finds the top-K nearest neighbors of a query point. The efficiency and effectiveness of FastMap+LSH allow us to rapidly switch between the original graphical space and its geometric interpretation.

We assume that the input is an edge-weighted undirected graph $G = (V, E, w)$, where V is the set of vertices, E is the set of edges, and for any edge $e \in E$, $w(e)$ specifies a non-negative weight associated with it. We also assume that G is connected since several measures of centrality and projected centrality are not very meaningful for disconnected graphs.[7] For simplicity, we further assume that there are no self-loops or multiple edges between any two vertices.

In the rest of this section, we first show how to use the FastMap framework for computing the top-K vertices in explicit graphs, for some popular measures of centrality. We then show how to use the FastMap framework more generally for computing the top-K pertinent vertices in implicit graphs, for the corresponding measures of projected centrality.

4.1 FastMap for Closeness Centrality on Explicit Graphs

Let $d_G(u, v)$ denote the shortest path distance between two distinct vertices $u, v \in V$. The closeness centrality [13] of v is the reciprocal of the average shortest path distance between v and all other vertices. It is defined as follows:

$$C_{clo}(v) = \frac{|V| - 1}{\sum_{u \in V, u \neq v} d_G(u, v)}. \tag{3}$$

Computing the closeness centrality values of all vertices and identifying the top-K vertices with the highest such values require calculating the shortest path distances between all pairs of vertices. All-pair shortest path computations generally require $O(|V||E|+|V|^2 \log |V|)$ time via the Floyd–Warshall algorithm [11].

The FastMap framework allows us to avoid the above complexity and compute the top-K vertices using a geometric interpretation. We know that given N points $q_1, q_2 \ldots q_N$ in Euclidean space $\mathbb{R}^\kappa$, finding the point q that minimizes $\sum_{i=1}^{N}(q - q_i)^2$ is easy. In fact, it is the centroid given by $q = (\sum_{i=1}^{N} q_i)/N$. Therefore, we can use the distance function $\sqrt{d_G(\cdot, \cdot)}$ in Algorithm 1 to embed the square-roots of the shortest path distances between vertices. This is done by returning the square-roots of the shortest path distances found by ShortestPathTree() in lines 4, 12, and 13. Computing the centroid in the resulting embedding minimizes the sum of the shortest path distances to all vertices. This centroid is mapped back to the original graphical space via LSH.

Overall, we use the following steps to find the top-K vertices: (1) Use FastMap with the square-root of the shortest path distance function between vertices to

[7] For disconnected graphs, we usually consider the measures of centrality and projected centrality on each connected component separately.

create a Euclidean embedding; (2) Compute the centroid of all points corresponding to vertices in this embedding; and (3) Use LSH to return the top-K nearest neighbors of the centroid.

4.2 FastMap for Harmonic Centrality on Explicit Graphs

The harmonic centrality [2] of a vertex v is the sum of the reciprocals of the shortest path distances between v and all other vertices. It is defined as follows:

$$C_{har}(v) = \sum_{u \in V, u \neq v} \frac{1}{d_G(u,v)}. \tag{4}$$

As in the case of closeness centrality, the time complexity of computing the top-K vertices, based on shortest path algorithms, is $O(|V||E| + |V|^2 \log |V|)$. However, the FastMap framework once again allows us to avoid this complexity and compute the top-K vertices using analytical techniques. Given N points $q_1, q_2 \ldots q_N$ in Euclidean space $\mathbb{R}^\kappa$, finding the point q that maximizes $\sum_{i=1}^{N} \frac{1}{\|q-q_i\|}$ is not easy. However, the Euclidean space enables gradient ascent and the standard ingredients of local search to avoid local maxima and efficiently arrive at good solutions. In fact, the centroid obtained after running Algorithm 1 is a good starting point for the local search.

Overall, we use the following steps to find the top-K vertices: (1) Use Algorithm 1 to create a Euclidean embedding; (2) Compute the centroid of all points corresponding to vertices in this embedding; (3) Perform gradient ascent starting from the centroid to maximize $\sum_{i=1}^{N} \frac{1}{\|q-q_i\|}$; and (4) Use LSH to return the top-K nearest neighbors of the result of the previous step.

4.3 FastMap for Current-Flow Centrality on Explicit Graphs

The current-flow closeness centrality [22,5] is a variant of the closeness centrality based on "effective resistance", instead of the shortest path distance, between vertices. It is also known as the *information centrality*, under the assumption that information spreads like electrical current. The current-flow closeness centrality of a vertex v is the reciprocal of the average effective resistance between v and all other vertices. It is defined as follows:

$$C_{cfc}(v) = \frac{|V| - 1}{\sum_{u \in V, u \neq v} R_G(u,v)}. \tag{5}$$

The term $R_G(u,v)$ represents the effective resistance between u and v. A precise mathematical definition for it can be found in [5].

Computing the current-flow closeness centrality values of all vertices and identifying the top-K vertices with the highest such values are slightly more expensive than calculating the shortest path distances between all pairs of vertices. The best known time complexity is $O(|V||E| \log |V|)$ [5].

Once again, the FastMap framework allows us to avoid the above complexity and compute the top-K vertices by merely changing the distance function used in Algorithm 1. We use the PASPD function presented in Algorithm 1 of [17]. (We ignore edge-complement graphs by deleting Line 14 of this algorithm.) The PASPD function computes the sum of the shortest path distances between two vertices in a set of graphs G_{set}. G_{set} contains different lineages of graphs, each starting from the given graph. In each lineage, a fraction of probabilistically-chosen edges is progressively dropped to obtain nested subgraphs. The PASPD captures the effective resistance between two vertices for the following two reasons: (a) The larger the $d_G(u, v)$, the larger the PASPD between u and v, as the effective resistance between them should be larger; and (b) The larger the number of paths between u and v in G, the smaller the PASPD between them, as the effective resistance between them should be smaller.

Overall, we use the following steps to find the top-K vertices: (1) Use FastMap with the PASPD function[8] between vertices to create a Euclidean embedding; (2) Compute the centroid of all points corresponding to vertices in this embedding; and (3) Use LSH to return the top-K nearest neighbors of the centroid.

4.4 Generalization to Projected Centrality

We now generalize the FastMap framework to compute the top-K pertinent vertices in implicit graphs for different measures of projected centrality. There are several methods to do this. The first method is to create an explicit graph by factoring out the auxiliary vertices, i.e., the explicit graph $G^P = (V^P, E^P)$ has only the pertinent vertices V^P and is a complete graph on them, where for any two distinct vertices $v_i^P, v_j^P \in V^P$, the edge $(v_i^P, v_j^P) \in E^P$ is annotated with the weight $\mathcal{D}_G(v_i^P, v_j^P)$. This is referred to as the All-Pairs Distance (APD) method. For the closeness and the harmonic centrality measures, $\mathcal{D}_G(\cdot, \cdot)$ is the shortest path distance function. For the current-flow closeness centrality measure, $\mathcal{D}_G(\cdot, \cdot)$ is the PASPD function. The second method also constructs the explicit graph G^P but computes the weight on each edge only approximately using differential heuristics [24]. This is referred to as the Differential Heuristic Distance (DHD) method. The third method is similar to the second, except that it uses the FastMap heuristics [7,17] instead of the differential heuristics. This is referred to as the FastMap Distance (FMD) method.

The foregoing three methods are inefficient because they construct G^P explicitly by computing the distances between all pairs of pertinent vertices. To avoid this inefficiency, we propose the fourth and the fifth methods. The fourth method is to directly create the FastMap embedding for all vertices of G but apply the analytical techniques only to the points corresponding to the pertinent vertices. This is referred to as the FastMap All-Vertices (FMAV) method. The fifth method is to create the FastMap embedding only for the pertinent vertices of G and apply the analytical techniques to their corresponding points. This is referred to as the FastMap Pertinent-Vertices (FMPV) method.

[8] It is also conceivable to use the square-root of the PASPD function.

Instance	Size ($\|V\|$, $\|E\|$)	Closeness			Harmonic			Current-Flow		
		GT	FM	nDCG	GT	FM	nDCG	GT	FM	nDCG
myciel5	(47, 236)	0.01	0.01	0.8810	0.01	0.08	0.8660	0.00	0.06	0.7108
games120	(120, 638)	0.06	0.03	0.9619	0.06	0.21	0.9664	0.02	0.12	0.9276
miles1500	(128, 5198)	0.41	0.09	0.9453	0.42	0.29	0.8888	0.05	0.79	0.9818
queen16_16	(256, 6320)	1.06	0.12	0.9871	1.07	0.49	0.9581	0.11	0.84	0.9381
le450_5d	(450, 9757)	3.13	0.23	0.9560	3.11	0.91	0.9603	0.30	1.59	0.8648
myciel4	(23, 71)	0.00	0.01	0.9327	0.00	0.04	0.8299	0.00	0.02	0.7697
games120	(120, 638)	0.07	0.03	0.8442	0.07	0.21	0.8004	0.02	0.14	0.9032
miles1000	(128, 3216)	0.28	0.07	0.9427	0.28	0.25	0.8510	0.04	0.47	0.7983
queen14_14	(196, 4186)	0.56	0.09	0.8866	0.55	0.38	0.8897	0.06	0.73	0.9188
le450_5c	(450, 9803)	3.32	0.24	0.8843	3.27	0.88	0.9203	0.29	2.09	0.8196
kroA200	(200, 19900)	2.59	0.52	0.9625	2.54	0.50	0.7275	0.14	2.95	0.7589
pr226	(226, 25425)	4.01	0.51	0.9996	4.06	0.63	0.6803	0.18	3.52	0.7978
pr264	(264, 34716)	6.87	0.62	0.9911	6.95	0.84	0.6506	0.30	6.30	0.8440
lin318	(318, 50403)	13.62	1.00	0.9909	13.16	1.19	0.9537	0.43	8.07	0.7243
pcb442	(442, 97461)	39.97	1.91	0.9984	39.25	2.01	0.9757	0.84	17.77	0.7283
orz203d	(244, 442)	0.11	0.06	0.9975	0.11	0.41	0.9943	0.05	0.13	0.8482
den404d	(358, 632)	0.23	0.08	0.9969	0.23	0.58	0.8879	0.10	0.14	0.9471
isound1	(2976, 5763)	18.19	0.63	0.9987	18.55	4.86	0.9815	6.18	1.95	0.9701
lak307d	(4706, 9172)	46.74	1.02	0.9996	48.56	7.66	0.9866	15.82	2.90	0.9845
ht_chantry_n	(7408, 13865)	131.30	1.51	0.9969	134.42	12.29	0.9144	37.92	3.64	0.9189
n0100	(100, 99)	0.01	0.02	0.9171	0.01	0.17	0.8102	0.01	0.05	0.9124
n0500	(500, 499)	0.34	0.10	0.8861	0.33	0.79	0.6478	0.18	0.16	0.9466
n1000	(1000, 999)	1.33	0.19	0.9125	1.38	1.63	0.9477	0.70	0.34	0.7292
n1500	(1500, 1499)	3.00	0.28	0.8856	3.15	2.39	0.6360	1.61	0.41	0.8118
n2000	(2000, 1999)	5.25	0.37	0.9078	5.56	3.21	0.8925	2.71	0.75	0.9516
n0100k4p0.3	(100, 262)	0.02	0.03	0.9523	0.03	0.17	0.9308	0.01	0.08	0.9326
n0500k6p0.3	(500, 1913)	0.83	0.11	0.9340	0.85	0.83	0.8951	0.23	0.51	0.8411
n1000k4p0.6	(1000, 3192)	3.00	0.24	0.9119	3.07	1.68	0.8975	1.13	0.83	0.8349
n4000k6p0.6	(4000, 19121)	86.37	1.09	0.9095	85.18	6.70	0.9160	214.99	8.13	0.8054
n8000k6p0.6	(8000, 38517)	387.25	2.86	0.9240	392.76	14.65	0.9368	474.30	11.08	0.7466

Table 1: Results for various measures of centrality. Entries show running times in seconds and nDCG values.

5 Experimental Results

We used six datasets in our experiments: DIMACS, wDIMACS, TSP, movingAI, Tree, and SmallWorld. The DIMACS dataset[9] is a standard benchmark dataset of unweighted graphs. We obtained edge-weighted versions of these graphs, constituting our wDIMACS dataset, by assigning an integer weight chosen uniformly at random from the interval $[1, 10]$ to each edge. We also obtained edge-weighted graphs from the TSP (Traveling Salesman Problem) dataset [21] and large unweighted graphs from the movingAI dataset [23]. In addition to these benchmark datasets, we synthesized Tree and SmallWorld graphs using the Python library NetworkX [14]. For the trees, we assigned an integer weight chosen uniformly at random from the interval $[1, 10]$ to each edge. We generated the small-world graphs using the Newman-Watts-Strogatz model [19]. For the regular measures of centrality, the graphs in the six datasets were used as such. For the projected measures of centrality, 50% of the vertices in each graph were randomly chosen

[9] https://mat.tepper.cmu.edu/COLOR/instances.html

	Instance	Running Time (s)						nDCG				
		APD	DHD	FMD	ADT	FMAV	FMPV	DHD	FMD	ADT	FMAV	FMPV
Closeness	queen16_16	0.55	0.32	0.12	0.05	0.12	0.10	0.9827	0.9674	0.9839	0.9707	0.9789
	le450_5d	1.58	0.98	0.29	0.09	0.27	0.26	0.9576	0.9621	0.9872	0.9577	0.9521
	queen14_14	0.29	0.19	0.08	0.03	0.08	0.09	0.9274	0.8996	0.9639	0.9377	0.8898
	le450_5c	1.71	1.00	0.32	0.09	0.28	0.23	0.9169	0.9231	0.9700	0.8959	0.8798
	lin318	7.01	0.68	0.50	0.45	0.97	1.05	0.9285	1.0000	0.9645	0.9867	1.0000
	pcb442	18.03	1.31	0.93	0.84	2.26	1.97	0.9455	1.0000	0.9663	0.9969	0.9950
	lak307d	24.78	116.69	25.85	1.98	0.36	0.35	0.8991	0.9928	0.9387	0.9994	0.9960
	ht_chantry_n	67.59	266.07	58.18	4.88	0.47	0.45	0.8956	0.9952	0.9522	0.9879	0.9879
	n1500	1.71	10.87	2.35	0.19	0.06	0.06	0.7990	0.9485	0.9830	0.7759	0.7758
	n2000	2.96	19.45	4.32	0.35	0.08	0.09	0.8187	0.9691	0.9720	0.9284	0.9229
	n4000k6p0.6	44.54	78.68	16.96	1.37	0.84	0.68	0.9403	0.9172	0.9582	0.9146	0.9299
	n8000k6p0.6	207.24	319.56	67.23	5.39	1.37	1.58	0.9432	0.9376	0.9522	0.9274	0.9296
Harmonic	queen16_16	0.54	0.31	0.12	0.00	0.40	0.32	0.9730	0.9602	0.9729	0.9466	0.9730
	le450_5d	1.59	0.95	0.31	0.00	0.65	0.52	0.9467	0.9499	0.9901	0.9447	0.9578
	queen14_14	0.31	0.18	0.08	0.00	0.29	0.22	0.9235	0.8912	0.9903	0.9381	0.8767
	le450_5c	1.74	0.98	0.30	0.00	0.65	0.51	0.8905	0.8694	0.9962	0.8485	0.8866
	lin318	6.93	0.72	0.52	0.00	1.22	1.06	0.9922	1.0000	0.8974	0.8128	0.8289
	pcb442	20.79	1.39	1.04	0.01	2.07	1.85	0.9924	1.0000	0.9859	0.9274	0.9755
	lak307d	24.76	105.98	23.11	0.16	4.09	3.94	0.9265	0.9915	0.9908	0.9819	0.9762
	ht_chantry_n	65.83	262.83	58.63	0.37	6.34	6.37	0.7549	0.6484	0.9947	0.8909	0.9925
	n1500	1.74	10.83	2.41	0.01	1.33	1.22	0.6280	0.7079	0.9858	0.6575	0.7469
	n2000	2.95	19.09	4.15	0.01	1.71	1.61	0.6695	0.6647	0.9354	0.9227	0.9252
	n4000k6p0.6	44.53	78.08	17.17	0.10	3.74	3.76	0.9252	0.9083	0.9971	0.9163	0.9072
	n8000k6p0.6	215.14	314.85	67.36	0.43	7.94	7.39	0.9351	0.9111	0.9999	0.9074	0.9439
Current-Flow	queen16_16	5.92	0.63	0.72	-	1.21	1.26	0.9638	0.9542	-	0.9690	0.9683
	le450_5d	17.50	1.47	1.45	-	2.02	2.05	0.9463	0.9678	-	0.9529	0.9632
	queen14_14	3.25	0.38	0.59	-	1.29	1.16	0.9243	0.8916	-	0.8867	0.8946
	le450_5c	30.90	2.03	2.58	-	5.55	5.03	0.9275	0.9372	-	0.8832	0.8778
	lin318	61.02	3.29	5.22	-	11.69	10.99	0.8995	1.0000	-	0.9993	0.9990
	pcb442	180.43	6.96	12.75	-	24.14	23.77	0.9781	1.0000	-	0.9997	0.9997
	lak307d	265.38	105.91	24.77	-	3.28	2.71	0.6864	0.6158	-	0.6800	0.6316
	ht_chantry_n	499.27	260.40	58.07	-	1.88	3.54	0.4634	0.4568	-	0.4314	0.4997
	n1500	5.70	10.62	2.45	-	0.55	0.30	0.6731	0.6820	-	0.6436	0.6487
	n2000	10.01	19.05	4.21	-	0.58	0.63	0.6689	0.6847	-	0.6487	0.6468
	n4000k6p0.6	448.20	76.73	19.42	-	7.74	7.65	0.9218	0.9130	-	0.9320	0.9126
	n8000k6p0.6	2014.92	306.53	73.18	-	16.04	14.40	0.9325	0.9174	-	0.9189	0.9229

Table 2: Results for various measures of projected centrality. Entries show running times in seconds and nDCG values.

to be the pertinent vertices. (The choice of the percentage of pertinent vertices need not be 50%. This value is chosen merely for presenting illustrative results.)

We note that the largest graphs chosen in our experiments have about 18500 vertices and 215500 edges.[10] Although FastMap itself runs in near-linear time and scales to much larger graphs, some of the baseline methods used for comparison in Tables 1 and 2 are impeded by such large graphs. Nonetheless, our choice of problem instances and the experimental results on them illustrate the important trends in the effectiveness of our approach.

We used two metrics for evaluation: the normalized Discounted Cumulative Gain (nDCG) and the running time. The nDCG [15] is a standard measure of the effectiveness of a ranking system. Here, it is used to compare the (projected)

[10] Tables 1 and 2 show only representative instances that may not match these numbers.

centrality values of the top-K vertices returned by an algorithm against the (projected) centrality values of the top-K vertices in the ground truth (GT). The nDCG value is in the interval $[0, 1]$, with higher values representing better results, i.e., closer to the GT. We set $K = 10$. All experiments were done on a laptop with a 3.1GHz Quad-Core Intel Core i7 processor and 16GB LPDDR3 memory. We implemented FastMap in Python3 and set $\kappa = 4$.

Table 1 shows the performance of our FastMap (FM) framework against standard baseline algorithms that produce the GT for various measures of centrality. The standard baseline algorithms are available in NetworkX.[11] The rows of the table are divided into six blocks corresponding to the six datasets in the order: DIMACS, wDIMACS, TSP, movingAI, Tree, and SmallWorld. Due to limited space, only five representative instances are shown in each block. For all measures of centrality, we observe that FM produces high-quality solutions and is significantly faster than the standard baseline algorithms on large instances.

Table 2 shows the performances of APD, DHD, FMD, FMAV, and FMPV for various measures of projected centrality. An additional column, called "Adapted" (ADT), is introduced for the closeness and harmonic measures of projected centrality. For the closeness and harmonic measures, ADT refers to our intelligent adaptations of state-of-the-art algorithms, presented in [6] and [1], respectively, to the projected case. The rows of the table are divided into three blocks corresponding to the three measures of projected centrality. Due to limited space, only twelve representative instances are shown in each block: the largest two from each block of Table 1. The nDCG values for DHD, FMD, ADT, FMAV, and FMPV are computed against the GT produced by APD. We observe that all our algorithms produce high-quality solutions for the various measures of projected centrality on most instances. While the success of ADT is attributed to the intelligent adaptations of two separate algorithms, the success of FMAV and FMPV is attributed to the power of appropriate analytical techniques used in the same FastMap framework. The success of DHD and FMD is attributed to their ability to closely approximate the all-pairs distances. We also observe that FMAV and FMPV are significantly more efficient than APD, DHD, and FMD since they avoid the construction of explicit graphs on the pertinent vertices. For the same reason, ADT is also efficient when applicable.

For all measures of centrality and projected centrality considered in this paper, Tables 1 and 2 demonstrate that our FastMap approach is viable as a unified framework for leveraging the power of analytical techniques. This is in contrast to the nature of other existing algorithms that are tied to certain measures of centrality and have to be generalized to the projected case separately.

6 Conclusions and Future Work

In this paper, we generalized various measures of centrality on explicit graphs to corresponding measures of projected centrality on implicit graphs. Computing

[11] https://networkx.org/documentation/stable/reference/algorithms/centrality.html

the top-K pertinent vertices with the highest projected centrality values is not always easy for large graphs. To address this challenge, we proposed a unifying framework based on FastMap, exploiting its ability to embed 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. We designed different distance functions for different measures of projected centrality and invoked various procedures for computing analytical solutions in the resulting FastMap embedding. We also coupled FastMap with LSH to interpret analytical solutions found in the FastMap embedding back in the graphical space. Overall, we experimentally demonstrated that the FastMap+LSH framework is both efficient and effective for many popular measures of centrality and their generalizations to projected centrality.

Unlike other methods, our FastMap framework is not tied to a specific measure of projected centrality. This is because its power stems from its ability to transform a graphical problem into Euclidean space in only near-linear time close to that of merely reading the input. Consequently, it delegates the combinatorics tied to any given measure of projected centrality to various kinds of analytical techniques that are better equipped for efficiently absorbing large input sizes.

In future work, we will apply our FastMap framework to various other measures of projected centrality not discussed in this paper, strengthening the confluence of discrete and analytical algorithms for graphical problems.

7 Acknowledgments

This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant number 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 is also partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

References

1. Bergamini, E., Borassi, M., Crescenzi, P., Marino, A., Meyerhenke, H.: Computing top-k closeness centrality faster in unweighted graphs. ACM Transactions on Knowledge Discovery from Data (2019)
2. Boldi, P., Vigna, S.: Axioms for centrality. Internet Mathematics (2014)
3. Bonacich, P.: Power and centrality: A family of measures. American Journal of Sociology (1987)
4. Bonchi, F., De Francisci Morales, G., Riondato, M.: Centrality measures on big graphs: Exact, approximated, and distributed algorithms. In: Proceedings of the 25th International Conference Companion on World Wide Web (2016)
5. Brandes, U., Fleischer, D.: Centrality measures based on current flow. In: Annual Symposium on Theoretical Aspects of Computer Science (2005)

6. Cohen, E., Delling, D., Pajor, T., Werneck, R.F.: Computing classic closeness centrality, at scale. In: Proceedings of the 2nd ACM Conference on Online Social Networks (2014)
7. 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 27th International Joint Conference on Artificial Intelligence (2018)
8. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the 20th Annual Symposium on Computational Geometry (2004)
9. Eppstein, D., Wang, J.: Fast approximation of centrality. Graph Algorithms and Applications (2006)
10. Faloutsos, C., Lin, K.I.: FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (1995)
11. Floyd, R.W.: Algorithm 97: shortest path. Communications of the ACM (1962)
12. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (1987)
13. Freeman, L.: Centrality in social networks conceptual clarification. Social Networks (1979)
14. 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)
15. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems (2002)
16. Katz, L.: A new status index derived from sociometric analysis. Psychometrika (1953)
17. Li, A., Stuckey, P., Koenig, S., Kumar, T.K.S.: A FastMap-based algorithm for block modeling. In: Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (2022)
18. 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)
19. Newman, M.E., Watts, D.J.: Renormalization group analysis of the small-world network model. Physics Letters A (1999)
20. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
21. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA Journal on Computing (1991)
22. Stephenson, K., Zelen, M.: Rethinking centrality: Methods and examples. Social Networks (1989)
23. Sturtevant, N.: Benchmarks for grid-based pathfinding. Transactions on Computational Intelligence and AI in Games (2012)
24. Sturtevant, N.R., Felner, A., Barrer, M., Schaeffer, J., Burch, N.: Memory-based heuristics for explicit state spaces. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (2009)