Efficient Geometric Graph Matching Using Vertex Embedding

Ayser Armiti and Michael Gertz

Institute of Computer Science
Heidelberg University, Germany
{ayser.armiti,gertz}@informatik.uni-heidelberg.de

ABSTRACT

For many applications such as road network analysis and image processing, it is critical to study spatial properties of objects in addition to object relationships. Geometric graphs provide a suitable modeling framework for such applications, where vertices are located in some 2D space. For applications where the similarity between the structures of different graphs plays an important role, typically, inexact graph matching algorithms are employed. However, graph matching algorithms face many problems such as scalability with respect to graph size and less tolerance to changes in graph structure or labels.

In this paper, we propose a solution to the problem of inexact graph matching for geometric graphs in the 2D space. Our approach allows to effectively answer subgraph and common subgraph queries for geometric graphs that differ in structure, spatial properties, and labels. Initially, a spatial feature is extracted from each vertex, and string edit distance is used to find the distance between pairs of vertices. To speed up graph matching, we propose vertex embedding into the Euclidean space. Based on this, the distance between two vertices can be computed using the Euclidean distance in constant time. To answer subgraph and common subgraph queries, we introduce an iterative matching algorithm that matches two graphs using their similarity in the Euclidean space. Such an algorithm merges highly similar vertices to create similar connected subgraphs. Using representative geometric graphs extracted from road networks, we show that our approach outperforms existing graph matching approaches in terms of matching quality and runtime.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Database applications—*Data mining*; G.2.2 [**Discrete mathematics**]: Graph theory—*Graph algorithms*

SIGSPATIAL'13, November 05 - 08 2013, Orlando, FL, USA

ACM 978-1-4503-2521-9/13/11...$15.00.
http://dx.doi.org/10.1145/2525314.2525350

Keywords

Graph matching, Vertex embedding

1. INTRODUCTION

For many applications such as drug discovery, road network analysis, and image processing, it is critical to study geometric properties of objects in addition to object relationships. *Geometric graphs* provide a suitable modeling tool for such applications, where a vertex has a location in 2- or 3-dimensional space. For graph-based applications, *inexact graph matching* was introduced to determine the similarity between (sub)graphs that differ in spatial attributes, labeling information, and graph structure [20].

Several approaches have been proposed to address the inexact graph matching problem, which is known to be NP-hard [27]. *Spectral graph methods* [24] and the *graph edit distance* [10] are two popular solutions to the graph matching problem. However, such solutions have their own drawbacks. The main drawbacks of the spectral graph matching approach are: 1) it is incapable of handling labeled graphs. Weighted graphs are the only type of graphs that can be analyzed by spectral methods, 2) it is sensitive to differences in the number of vertices and graph structure.

On the other side, the complexity of finding the optimal solution of the graph edit distance is NP-hard [27]. Furthermore, suboptimal solutions are computed based on subgraphs assignment [17]. For geometric graphs, measuring the similarity between two subgraphs with spatial constraints is a hard task. Two subgraphs are considered spatially identical under spatial transformation, i.e., translation, rotation, and sometimes scaling. In addition to this, computing the similarity between subgraphs under non-rigid transformation is even more complicated [13].

In this paper, we propose a solution to the inexact graph matching problem for labeled geometric graphs. Such a solution answers subgraph and common subgraph queries for geometric graphs that differ in structure, spatial properties, and labels. Several application domains can utilize our proposed algorithm. For example, in pattern recognition and image processing, given an object and an image each represented as a geometric graph, our algorithm efficiently locates the object in the image. Other applications of our algorithm in the domain of road networks are, for example:

1. Given a road network and a geometric graph that is generated by a GPS-enabled device, locate that device in the road network.

2. Find differences in a road network over time.

3. Given an image of a road network, locate this road network in the map.

To speed up graph matching, in this paper, we propose vertex embedding into the Euclidean space. Instead of embedding the vertices based on the global structure of a graph, as in the spectral approach, our algorithm embeds the vertices based on their local structures. As shown in Figure 1, initially, each graph is decomposed into a multi-set of subgraphs. We call each subgraph *vertex signature*. Each vertex signature represents a vertex and its direct neighbors. Then, the coordinates for each vertex, in the Euclidean space, is defined by the distances between its vertex signature and a set of *prototype* vertex signatures. We define the distance between two vertex signatures as the edit distance between them. Once all vertices are embedded, the distance between any two vertices can be computed using the Euclidean distance in constant time.

Embedding the vertices as described above looses the global structure of a graph. As a result and to match two graphs based on their vector-based representations, we use an iterative graph matching algorithm [6, 29]. Such an algorithm merges highly similar vertices to create similar connected subgraphs. Each iteration consists of two steps: a match and a refinement steps. In the refinement step, the similarities between the vertices of two graphs are refined. To refine the similarity of two vertices, we adopt a probabilistic approach that integrates structural compatibility with the similarity in the Euclidean space. By the end of the refinement step, a set of candidate matches is selected. In the match step, the Hungarian algorithm [15] is used to select the maximum weighted match from the set of candidate matches.

The remainder of this paper is organized as follows. In Section 2, we survey related work. Section 3 discusses a metric function to compute the distance between different vertices. Such a function is then used in Section 4 to embed vertices into the Euclidean space. In Section 5, we show how to match different graphs based on their similarities in the Euclidean space. In Section 6, we present experimental results of our proposed approach. We summarize the paper and outline ongoing work in Section 7.

2. RELATED WORK

Well-known solutions to the graph matching problem are spectral and graph edit distance approaches. Spectral graph approaches solve the matching problem by utilizing the global structure of a graph. To match two graphs, vertices are initially embedded into a vector space (Eigenspace). Such a vector space is spanned by the Eigenvectors and the Eigenvalues of the adjacency or the Laplacian matrices. Then, the two graphs are matched based on the similarity of their embedded spaces. Umeyama in [24] was the first to use the spectra of the adjacency matrix to embed vertices into the Eigenspace. He used a vertex-to-vertex similarity matrix and the Hungarian algorithm to match different graphs. Knossow et al. [12] extended Umeyama's approach to match graphs that differ in the number of vertices. They used the Laplacian matrix to embed different graphs. Then, the well-known expectation maximization algorithm is used for graph matching. In [3], the shortest path distance matrix is used instead of the adjacency matrix. Based on such matrix, multi-dimensional scaling and singular value decomposition (SVD) are used to embed and match different graphs, respectively. Recently, Xiao et al. [26] used the heat kernel of the Laplacian matrix to embed different graphs. The heat kernel with its time parameter t is used to describe the flow of information over the edges of a graph over time. After that, two graphs are matched by using a variation of the Scott and Longuet-Higgins algorithm [23]. Such an algorithm uses SVD to create a similarity matrix between the vertices of different graphs. Two vertices are matched if their similarity is the maximum among all similarities of the matches containing any of them. Spectral graph methods with their strong mathematical foundation have been shown to give good results. Also, they are faster than other optimization or search-based graph matching algorithms.

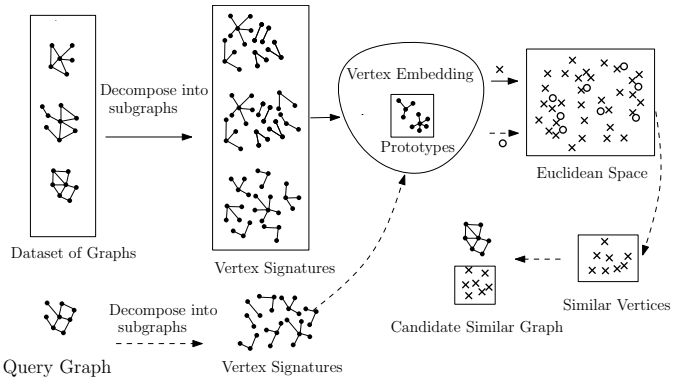


Figure 1: Proposed graph matching framework. Initially, a dataset of graphs is decomposed into a multi-set of vertex signatures. Then, all vertex signatures are embedded into the Euclidean space. Similarly, a query graph is decomposed and embedded. Based on the similarity in the Euclidean space, candidate similar vertices are retrieved, which are then used for graph matching.

The graph edit distance approach searches for the minimum cost sequence of edit operations that transforms one graph to be identical to another. Mainly three edit operations are used: substitution, insertion, and deletion. Sanfeliu and Fu were the first to propose the graph edit distance [22]. Since the complexity of finding the optimal solution of the graph edit distance is NP-hard [27], many approximate solutions have been proposed. Most of these solutions follow a graph decomposition approach. Eshera and Fu were the first to propose such an approach [9]. They created an acyclic directed lattice from the vertices of two graphs. Then, dynamic programming is used to find the edit path between the two graphs. Recently, Riesen and Bunke in [18] used the assignment problem to approximate the graph edit distance. A vertex-to-vertex distance matrix is created. The distance is defined as the distance between the labels of the vertices in addition to the labels of the edges. Munkres algorithm [15] is then used to find the correspondence between two graphs. Graph edit distance is very flexible since it handles graphs with labels for vertices and edges, in addition to spatial attributes. Also, it gives the possibilities to define the cost of each edit operation in a way that fits the notion of similarity in an application domain. For a survey of different solutions to the graph matching problem, we refer the reader to [7, 20].

The related graph matching algorithms face many problems including scalability with respect to graph size and less

tolerance to changes in graph structure or labels. In this paper, we propose an efficient graph matching algorithm that overcomes such problems and can easily be adopted to different application domains.

3. VERTEX SIMILARITY

For graph-based applications, even computing the similarity between two vertices is not trivial. In addition to structural information, labels and spatial attributes are also used to discover similar vertices. In this section, we propose a metric function to compute the distance between vertices. First, in Section 3.1, we define the concepts associated with geometric graphs. In Section 3.2, we introduce the notion of vertex signature and its spatial feature. Based on this spatial feature, in Section 3.3, we propose our metric function to compute the distance between vertices of geometric graphs in 2D space. This metric serves as the basis to our graph matching and embedding algorithms.

3.1 Preliminaries

In our framework, we consider labeled undirected geometric graphs that do not contain self-loops or multi-edges.

Definition 1. **(Geometric Graph)** A labeled undirected geometric graph $G = (V, E, l, c)$ consists of a finite set of vertices V, a finite set of edges $E \subseteq V \times V$, a labeling function $l : \{V \cup E\} \rightarrow \Sigma$, assigning a label to every vertex and every edge from a label alphabet Σ, and a function $c : V \rightarrow \mathbb{R}^2$, assigning a coordinate from the space $\mathbb{R}^2$ to every vertex.

Without loss of generality and through the rest of this paper, we represent a geometric graph G as $G = (V, E)$. We refer to the number of vertices and edges in a graph as $|V|$ and $|E|$, respectively. The size $|G|$ of a graph G is the number of vertices in G. The degree of a vertex v, denoted $\deg(v)$, is the number of vertices connected to it. The length of an edge e is denoted by $|e|$.

3.2 Spatial Feature

Before discussing how to compute the distance between vertices, in this section, we introduce the vertex signature and the spatial feature concepts. These concepts are the basis blocks of our vertex distance metric.

The similarity between two vertices can be estimated by the similarity of their neighbors, i.e., the more similar neighbors the two vertices have, the more similar they are. Based on this, vertex similarity can be estimated by subgraph similarity. However, the complexity of finding the optimal similarity between two subgraphs that differ in their structures and labels is exponential with respect to their sizes. To overcome this bottleneck, and for our framework, we use a minimal subgraph that consists of the vertex and its direct neighbors. We call such a subgraph *vertex signature*.

Definition 2. **(Vertex Signature)** Given a vertex v_i in a graph $G = (V, E)$, the vertex signature $S(v_i)$ is a subgraph $G' = (V', E')$ of G such that V' contains every vertex in G that is directly connected to v_i. For each vertex $v_j \in V'$, $v_j \neq v_i$, there exists an edge $e(v_i, v_j) \in E'$. v_i is called the *root vertex* of $S(v_i)$.

For geometric graphs, spatial constraints must be satisfied when computing the similarity of two vertex signatures. Two vertex signatures are considered spatially identical if there is a spatial transformation from one to the other, i.e., translation, rotation, and sometimes scaling [13]. However, the Euclidean distances between the coordinates of a vertex signature to another do not estimate the spatial transformation between them. This is because the coordinates of the vertices are measured with respect to the particular reference frame for each graph. To estimate the spatial transformation without utilizing the coordinates, we propose to extract a *spatial feature* from each vertex signature that is invariant to spatial transformation. It consists of the lengths of the edges in addition to the angles between them. The proposed spatial feature utilizes the property that edges in a vertex signature have a natural cyclic order. This order is a consequence of the embedding of neighboring vertices in 2D space. As a result, the spatial feature can be interpreted as a cyclic string of edges such that each edge is represented by two geometric properties: an angle and an edge length.

Definition 3. **(Spatial Feature)** Given a vertex signature $S(v)$, its spatial feature is a cyclic string of edges $F = [e_1, e_2, \ldots, e_n]$ such that $n = \deg(v)$ and $e_i = (\theta_{e_i}, l_{e_i})$, with θ_{e_i} denoting the angle between the edges e_i and e_{i-1}, and l_{e_i} denoting the length of the edge e_i.

3.3 String Edit Distance

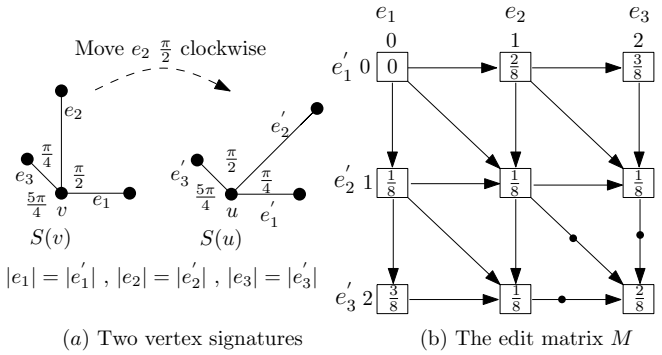
Based on the spatial features of two vertex signatures, the distance between two vertices is computed by the *cyclic string edit distance* (CS) [14]. The (cyclic) string edit distance is defined as the cost of the optimal sequence of edit operations that transfer a (cyclic) string identical to another.

A naive approach to solve the cyclic string edit distance runs in $O(nm^2)$, where n and m are the number of edges of the two vertex signatures. This is done by applying the algorithm by Wagner and Fisher [25] to the first spatial feature and all cyclic shifts of the second spatial feature. Maes in [14] proposed a faster solution to the cyclic string edit distance that runs in $O(nm \log m)$.

To utilize the CS approach, we define three edge edit operations: substitution, insertion, and deletion. We propose edit operations that combine spatial attributes, labeling information, and the structure around vertices. Given two vertex signature $S(v)$ and $S(u)$, $e_i \in S(v)$, $\deg(v) = n$, $e_j \in S(u)$, and $\deg(u) = m$, the substitution $\gamma(e_i \rightarrow e_j)$ is defined as:

$$\gamma(e_i \rightarrow e_j) := d_L(e_i, e_j) + d_S(e_i, e_j) + d_T(e_i, e_j) \quad (1)$$

In the case of labeled graphs, the function $d_L(e_i, e_j)$ computes the distance between the label of edge e_i and the label of e_j. It also computes the distance between the label of the neighboring vertex connected to e_i to the label of the one connected to e_j. The function $d_S(e_i, e_j)$ calculates the spatial distance based on the angles and the lengths of the edges. The function $d_T(e_i, e_j)$ defines the topological or structural distance based on the degree of neighboring vertices connected to those edges. For an edge e, let θ_e and l_e denote the angle and the edge length, as given in Definition 3. Also, let w_e denote the degree of the neighboring vertex connected to e. The functions d_S and d_T are formally defined as:



(a) Two vertex signatures (b) The edit matrix M

Figure 2: The edit matrix M used in one of the iterations to compute the cyclic string edit distance between $S(v)$ and $S(u)$.

$$d_S(e_i, e_j) := \frac{|\theta_{e_i} - \theta_{e_j}|}{2\pi} + \left| \frac{l_{e_i}}{\sum_{k=1}^{n} l_{e_k}} - \frac{l_{e_j}}{\sum_{k=1}^{m} l_{e_k}} \right| \quad (2)$$

$$d_T(e_i, e_j) := \left| \frac{w_{e_i}}{\sum_{k=1}^{n} w_{e_k}} - \frac{w_{e_j}}{\sum_{k=1}^{m} w_{e_k}} \right| \quad (3)$$

The angles and the lengths of the edges at a vertex signature are normalized, as can be seen by the denominators used in the above definitions. An angle is normalized by 2π since the sum of angles at a vertex signatures sums up to this value. Also, an edge length is normalized by the sum of edge lengths. For example, for a vertex signature $S(v)$, the edge length normalization factor is $\frac{1}{\sum_{k=1}^{n} l_{e_k}}$, where n is the number of edges connected to v. Also, the degree of a neighboring vertex connected to an edge is normalized by the sum of the neighbors' degrees.

In the following, we define the insertion and deletion operations. Let λ represent the null (non-existent) edge, then the insertion $\gamma(\lambda \to e_j)$ and deletion $\gamma(e_i \to \lambda)$ with respect to edges e_i and e_j are defined as follows:

$$\gamma(\lambda \to e_j) := \frac{\theta_{e_j}}{2\pi} + \frac{l_{e_j}}{\sum_{k=1}^{m} l_{e_k}} + \frac{w_{e_j}}{\sum_{k=1}^{m} w_{e_k}} + c(e_j) \quad (4)$$

$$\gamma(e_i \to \lambda) := \frac{\theta_{e_i}}{2\pi} + \frac{l_{e_i}}{\sum_{k=1}^{n} l_{e_k}} + \frac{w_{e_i}}{\sum_{k=1}^{n} w_{e_k}} + c(e_i) \quad (5)$$

The cost of edge insertion or deletion is computed based on the angle value, the edge length, and the neighbor degree. For labeled graphs, the function $c(e_i)$ defines the cost of inserting or deleting the label assigned to the edge e_i in addition to the label assigned to the neighboring vertex connected to e_i.

For unlabeled graphs, the cost of an edit operation lies in the range [0,3]. This is because each of the angle value, the edge length, and the neighbor degree is normalized to the range [0,1]. For labeled graphs, the range increases depending on the range of the function d_L for the substitution operation and c for the insertion and deletion.

Figure 2 shows an example on how the cyclic string edit distance is used to find the distance between two vertices. Figure 2(b) shows the edit matrix M for computing the distance between the two vertex signatures given in Figure 2(a).[1]The edit path between the two vertex signatures is $(e_1 \to e_1'), (e_2 \to e_2'), (e_3 \to e_3')$. When calculating the cost at $M[2,2]$, the string edit distance takes the minimum of $M[1,1] + \gamma(e_3 \to e_3')$ (substitution), $M[2,1] + \gamma(\lambda \to e_3)$ (insertion) and $M[1,2] + \gamma(e_3' \to \lambda)$ (deletion). Since $|e_3| = |e_3'|$, the substitution cost is $\gamma(e_3 \to e_3') = \frac{\frac{\pi}{2} - \frac{\pi}{4}}{2\pi} = \frac{1}{8}$.

After defining the three edge edit operations, the distance between two vertices is computed by the cyclic string edit distance, which in turn can be solved by using the dynamic programming paradigm as proposed by Wagner and Fischer [25]. Obviously, the proposed edit operations are metric functions, and under this condition, the cyclic string edit distance is also a metric function [14, 25].

We now show how the proposed vertex distance metric is used to embed vertices in the Euclidean space.

[1]For the sake of simplicity, labels and structure information are not included.

4. VERTEX EMBEDDING

Even though graphs are representative data structures, they have some weakness compared to a vector-based representation. An example is the complexity of computing the distance between two graphs. In vector spaces, the distance between two vectors can be computed easily by simple Euclidean distance, whereas computing the optimal distance between two graphs is NP-hard [27]. On the other hand, many algorithms for data mining and machine learning cannot be easily applied to graphs. As a result, embedding into vector spaces has been used to bridge the gab between both types of representations [4], e.g., vertices are embedded into vector spaces to answer shortest path and nearest neighbor queries in large social networks [28]. Graphs as a whole have been embedded to speedup graph classification [19].

Inspired by the work of Riesen and Bunke [19], in this section, we propose vertex embedding into the Euclidean space to speedup the matching of graphs. In Section 4.1, we formalize the embedding of vertices and introduce the concept of vertex prototype, which is the basis of the embedding scheme. In Section 4.2, we discuss how such prototypes are generated or selected from a training dataset.

4.1 Vertex Embedding

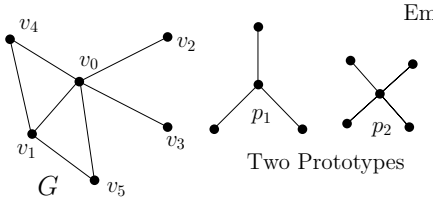
A vertex is embedded into the Euclidean space using its vertex signature. To embed all vertices from a graph, first, the graph is decomposed into a multi-set of vertex signatures. Then a representative set of vertex signatures is provided. Such vertex signatures are called *prototypes*. For a vertex, the distances between its vertex signature and the set of prototypes form a vector. This vector is considered the embedding of that vertex in the Euclidean space.

Definition 4. **(Vertex Embedding)** Let $G = (V, E)$ be a geometric graph with the multi-set of vertex signatures $S = \{s_1, s_2, \ldots, s_n\}$, $n = |V|$. Given a set of prototypes $P = \{p_1, p_2, \ldots, p_m\}$, s.t. $m < n$. The embedding using the prototype set is a mapping $\varphi : S \hookrightarrow \mathbb{R}^m$, defined as:

$$\varphi(s_i) = (d(s_i, p_1), d(s_i, p_2), \ldots, d(s_i, p_m))$$

where $d(s_i, p_j)$ is the cyclic string edit distance between the vertex signature s_i and the prototype p_j.

Each axis of the Euclidean space corresponds to a single prototype $p_i \in P$. This can be interpreted as a special case of the Lipschitz embedding where a set of prototypes is used to define each axis [11]. Figure 3 shows the embedding of



Embedding Coordinates

Vertex	p_1	p_2
v_0	1.11	0.68
v_1	0.67	1.07
v_2	1.92	2.16
v_3	1.92	2.16
v_4	1.38	1.86
v_5	1.39	1.88

Figure 3: The embedding of graph G into a 2-dimensional space using the prototypes p_1 and p_2.

graph G into a two-dimensional space that is spanned by the two prototypes p_1 and p_2. The two vertices v_4 and v_5, which are similar, are embedded near to each others.

After the embedding of vertices from different graphs, the distance between two vertices is computed by the Euclidean distance. Through the rest of the paper, the Euclidean distance between two vertices refers to the distance between their vector-representations. Based on the proposed embedding scheme, the complexity of computing the distance between two vertices is reduced from $O(nm^2)$, as proposed in Section 3.3, to the constant time complexity $O(1)$.

To find the relationship between the cyclic string edit distance of two vertices and their Euclidean distance, we follow the discussion in [19]. The Euclidean distance between the two vectors $\varphi(s_1)$ and $\varphi(s_2)$ is defined as:

$$\|\varphi(s_1)-\varphi(s_2)\|=\big(\|\varphi(s_1)\|^2+\|\varphi(s_2)\|^2-2\varphi(s_1)\cdot\varphi(s_2)\big)^{\frac{1}{2}} \quad (6)$$

Given that d, the cyclic string edit distance, is a metric, and due to the triangle inequality $|d(s_1,p_i)-d(s_2,p_i)|\leq d(s_1,s_2)$, we have:

$$\begin{aligned}\|\varphi(s_1)-\varphi(s_2)\|&=\Bigg(\sum_{i=1}^{m}d(s_1,p_i)^2+\sum_{i=1}^{m}d(s_2,p_i)^2\\&\quad-2\sum_{i=1}^{m}(d(s_1,p_i)d(s_2,p_i))\Bigg)^{\frac{1}{2}}\\&=\Bigg(\sum_{i=1}^{m}(d(s_1,p_i)-d(s_2,p_i))^2\Bigg)^{\frac{1}{2}}\\&\leq(m\cdot d(s_1,s_2)^2)^{\frac{1}{2}}\\&=\sqrt{m}\cdot d(s_1,s_2)\end{aligned} \quad (7)$$

As a result, $\frac{\|\varphi(s_1)-\varphi(s_2)\|}{\sqrt{m}}$ is a lower bound to the cyclic string edit distance between the two vertex signatures s_1 and s_2.

4.2 Prototype Selection

Matching two graphs based on their vector-based representations relies on the embedding of similar vertices near to each others. To guarantee this, a representative set of prototypes must be used. This makes the prototypes the basis and crucial part of the proposed embedding scheme. Furthermore, in addition to the prototypes themselves, the number of prototypes is also a crucial decision. In this section, we discuss how prototypes are generated or selected from a graph training dataset.

Prototype selection is studied in the literature to improve well-known problems for the k-NN classifier [16]. Examples for such problems are low tolerance to noise and less efficiency due to distance computation. For vertex embedding, a prototype selection method should also take care of these issues.

In the following, we discuss three prototype selection methods: *random selection* (RS), *medoids selection* (MS), and *spanning selection* (SP).

Random Selection. For graphs where a training sample is not available, the random selection method is used. Prototypes of vertex signatures are artificially created. The user specifies the number of prototypes. Then, labels and spatial attributes are randomly assigned to each prototype, i.e., labels for the edges and the neighboring vertices, in addition to the angles and the lengths of the edges.

Medoids Selection. The medoids selection method is used to select prototypes from a graph training dataset. It utilizes the well-known k-medoids clustering algorithm to select k prototypes. Each prototype represents the center of a cluster of vertex signatures. The k-medoids algorithm uses a distance matrix that is created from vertex signatures extracted from the training dataset. The distance between two vertex signatures is computed based on the cyclic string edit distance. The medoids selection can be interpreted as a frequent prototype selection method. Each cluster extracted by the k-medoids represents a cluster of a frequent vertex signature.

Spanning Selection. We adopt the spanning prototype selection method proposed in [19] to select prototypes of vertex signatures. This prototype selection method finds vertex signatures as uniformly distributed as possible from a training dataset. The median vertex signature is first selected, which is the one with the minimal sum of distances to all other vertex signatures. Here, the distance is also computed by the cyclic string edit distance. Then, iteratively, prototypes are selected, the vertex signature with the farthest distance from the already selected prototypes is selected. The advantage of this method over the medoids method is that it does not depend on an initial assignment.

5. GRAPH MATCHING

After the embedding of all graphs into the Euclidean space, the k-nearest neighbor algorithm can be used to find similar vertices. However, matching two graphs based on the similarity in the Euclidean space creates a match that is structurally inconsistent, i.e., two neighboring vertices from one graph maybe mapped to non-neighboring vertices from another one. This is because the distance function used in the embedding process (Section 3.3) considers only a vertex and its direct neighbors. As a result, graph connectivity is not preserved in the Euclidean space.

To solve this problem, we propose an iterative graph matching algorithm that integrates the similarity between vertices in the Euclidean space with structural compatibility. For this, in Section 5.1, we first introduce our iterative graph matching algorithm. In Section 5.2, we discuss how to initialize the match between two graphs. Then, in Section 5.3, we formalize a probabilistic approach to refine the match between two graphs based on structural compatibility.

5.1 Iterative Graph Matching

Our proposed algorithm iteratively refines the match between two graphs to create similar connected subgraphs. Each iteration consists of two steps: a ***graph matching*** step and a ***candidate refinement*** step. At an iteration t, the

graph matching step uses the Hungarian algorithm [15] to select the best match (M_t) from a set of candidate matches (C_t). In the candidate refinement step, the similarities between the vertices of one graph to the vertices of another one are updated in a voting scheme. Vertices from M_t vote to quantify structural compatibility between each pair of vertices. Then, a set of k candidate matches (C_{t+1}) is selected to be used in the next iteration. The candidate set (C_{t+1}) is created as follows. First, vertices from M_t are used to initialize C_{t+1}. Then, all neighboring vertices to the vertices in M_t are selected as candidates. Non-neighboring and highly similar vertices are further added to obtain a candidate set of size k. Since the Hungarian algorithms runs in cubic time, the graph matching step becomes the bottleneck of our iterative algorithm. To solve this problem, only k candidate matches are selected at the end of the candidate refinement step. This makes the similarity matrix used by the Hungarian algorithm sparse. As a result, our algorithm scales well with respect to graph size.

Given a scoring function $S : \mathcal{M} \rightarrow \mathbb{R}$, which quantifies the quality of a match, the algorithm converges when $S(M_{t+1}) - S(M_t) < \alpha$, where a α is a threshold defined by the user. Our experiments (Section 6) show that a good value of α is 10^{-4}. This value guarantees a good matching accuracy and a fast convergence time.

In the following, we detail how the candidate set is initialized and iteratively refined.

5.2 Candidate Initialization

The set of candidates used in the first iteration of the proposed graph matching algorithm is called the *initial candidate set*. The vertices from the initial candidate set are considered seeds for the proposed iterative algorithm. We call such seeds of vertices the *anchor vertices* or anchors. The anchors affect number of iterations needed for the convergence. Two issues should be considered when selecting them: 1) the similarity of the anchors from one graph to another, and 2) the distribution of anchors in each graph. Since the matching algorithm iteratively expands the match, a uniform distribution of the anchors reduces the number of iterations needed for the convergence. We propose two anchor selection methods guided by requirements from different application domains.

Common Subgraph Matching. The size of a common subgraph between two graphs Q and G is less than or equal to $\min\{|Q|, |G|\}$. As a result, distributing the anchors uniformly may include vertices from the non-common subgraph. However, to speedup the convergence, vertices from the non-common subgraph should be avoided. For this type of matching, only the similarity of vertices is used in the selection process. To do this, a similarity matrix is created from the vertices of Q and G. The similarity between two vertices is defined as the similarity of their embedding in the vector space. Then, the top k similar pairs of vertices from the similarity matrix are selected as the initial candidate set. Our experiments (Section 6) show that a good value for k is $1.3 \times \min\{|Q|, |G|\}$. In general, k must be greater than the size of the maximum common subgraph between Q and G. As a result, k must be greater than $\min\{|Q|, |G|\}$. On the other side, a high value of k increases the time required by the Hungarian algorithm at the graph matching step.

(Sub)graph Matching. In such type of graph matching and to increase the convergence speed, anchors are selected uniformly to cover the smaller graph Q. A *spanning selection* method is used to select anchors from Q [19]. The spanning selection method selects the first anchor as the *median vertex*. The median vertex is the vertex whose sum of distances to all other vertices in the graph is minimal. Here, the distance between two vertices in a graph is defined as the geodesic distance between them. Then, iteratively anchor vertices are selected. The vertex that is the farthest away from the already selected anchors is added to the anchor set. To guarantee a faster convergence rate, we choose only vertices with degree 3 or more. Once the anchors are selected from the graph Q, the similarities between them and all vertices of the other graph G are computed.

The similarity between two vertices is computed based on their similarity in the Euclidean space. Then, for each anchor from Q, the algorithm selects the k-NN vertices from the graph G as the initial candidate set. Experimentally, a good value for k is 10. As a result, the size of the candidate set is almost $10 \times |Q|$. In general, a higher value of k leads to a higher tolerance to spatial and structural differences between the two graphs. But on the other hand, the time required by the Hungarian algorithm increases.

5.3 Candidate Refinement

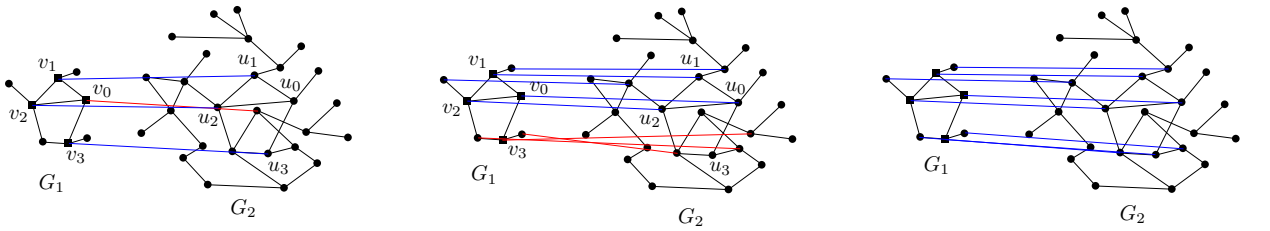
After initializing the candidate set with the anchors, the match is iteratively expanded and improved. The match computed at an iteration t is used to refine the similarity between vertices in the next iteration. Given two graphs Q and G with their vertex sets V and U, respectively, the similarity between the vertices is updated based on a probabilistic voting scheme. We adopt a Bayesian formulation similar to the one presented in [6]. Given a match $M_t = \{m_1, m_2, \ldots, m_{|M_t|}\}$ computed at an iteration t, the similarity between two vertices $v_k \in V$ and $u_j \in U$ is updated as a conditional joint probability distribution $p(V, U|M_t)$. To compute such a probability distribution, an auxiliary variable of a match $M \in M_t$ is used. The probability distribution is computed by marginalizing $p(V, U, M|M_t)$ over M. By using the chain rule, the similarity between v_k and u_j is defined as:

$$\begin{aligned} p(V{=}v_k,U{=}u_j|M_t) &= \sum_{\mathrm{m}_i \in M_t} p(V{=}v_k,U{=}u_j,M{=}\mathrm{m}_i|M_t) \\ &= \sum_{\mathrm{m}_i \in M_t} p(V{=}v_k|U{=}u_j,M{=}\mathrm{m}_i,M_t) \\ &\quad p(U{=}u_j|M{=}\mathrm{m}_i,M_t)p(M{=}\mathrm{m}_i|M_t) \quad (8) \end{aligned}$$

where $p(M{=}\mathrm{m}_i|M_t)$ is a prior representing the probability of choosing the match $\mathrm{m}_i{\in}M_t$. $p(U{=}u_j|M{=}\mathrm{m}_i,M_t)$ describes the probability of u_j being a neighbor to the match $\mathrm{m}_i{=}(v_i,u_i)$. $p(V{=}v_k|U{=}u_j,M{=}\mathrm{m}_i,M_t)$ represents the probability that v_k is related to u_j and the match m_i. This marginalizing can be seen as a probabilistic voting such that the voters are the matches $\mathrm{m}_i{\in}M_t$. In the following, we detail the realization of the three probabilities in Equation 8.

$$p(M{=}\mathrm{m}_i|M_t){=}\frac{\mathrm{score}(\mathrm{m}_i)}{\sum_{m_j \in M_t}\mathrm{score}(m_j)} \quad (9)$$

where $\mathrm{score}(\mathrm{m}_i)$ quantifies the quality of the match m_i. Let $\mathrm{m}_i{=}(v_i,u_i)$, where $v_i{\in}V$ and $u_i{\in}U$, $\mathrm{score}(\mathrm{m}_i)$ is the similarity between the vertices v_i and u_i as computed in the previous iteration.



a) iteration $t = 1$, accuracy = 37.5% b) iteration $t = 2$, accuracy = 62.5% c) iteration $t = 4$, accuracy 100%

Figure 4: The iterative graph matching between G_1 and G_2. The anchor vertices for G_1 are represented by squares. A correct correspondence is drawn in blue and a false one is drawn in red. a), b), and c) represent the matching results at the 1^{st}, 2^{nd}, and 3^{rd} iteration, respectively.

$$p(U{=}u_j|M{=}\mathrm{m}_i,M_t){=}\begin{cases}\frac{1}{\deg(u_i)}, & \text{if } u_j{\in}\mathrm{N}(u_i)\\ 0, & \text{otherwise}\end{cases} \quad (10)$$

for a vertex u_i, $\deg(u_i)$ denotes its degree and $\mathrm{N}(u_i)$ describes the set of its neighboring vertices.

$$p(V{=}v_k|U{=}u_j,M{=}\mathrm{m}_i,M_t)$$
$$={\begin{cases}1, & \text{if } (v_k,u_j){\in}M_t \text{ and } v_k{\in}N(v_i)\\ \frac{\exp(-d(v_k,u_j))}{Z}, & \text{if } (v_k,u_j){\notin}M_t \text{ and } v_k{\in}N(v_i)\\ 0, & \text{otherwise}\end{cases}} \quad (11)$$

where $d(v_k,u_j)$ is the Euclidean distance between the two vertices v_k and u_j. Z is a normalization factor. For the two vertices $v_i{\in}V$ and $u_j{\in}U$, $Z{=}\sum_{v_k\in N(v_i)} exp(-d(v_k,u_j))$. Once the similarity between the vertices in updated following this probabilistic approach, the candidate set C_{t+1} is selected as discussed earlier.

To summarize, the probability to match any two vertices $v_i{\in}V$ and $u_j{\in}U$ increases when 1) they are similar in the Euclidean space and 2) their neighbors exist in the match M_t. The more neighbors exist in M_t, the higher the probability that v_i is matched to u_j. Figure 4 illustrates our iterative matching algorithm. At the end of the 1^{st} iteration, the similarity between $v_0{\in}G_1$ and $u_0{\in}G_2$ increases since all the neighbors of v_0 are matched to the neighbors of u_0, whereas the similarity between v_3 and u_3 decreases since there is no neighbor of v_3 that is matched to a neighbor of u_3.

6. EVALUATION

In this section, our proposed solution to the graph matching problem is empirically evaluated. For this, we use geometric graphs that are extracted from three road networks: California, North America, and the city of Oldenburg [2]. Longitude and latitude are used as the x and y coordinates of the vertices. Since a road segment between two intersections is represented by several nodes in the road network, we simplified them using the Douglas Peucker algorithm [8]. Table 1 shows the number of vertices and edges for the simplified road networks.

The focus of this section is to empirically evaluate two criteria for a graph matching algorithm. The first criterion is the scalability with respect to graph size. It is measured by the running time required to match different graphs. The second criterion is the quality of the match computed by a graph matching algorithm. Algorithms with high matching quality are more resistant to changes in graph structure and spatial attributes. For our experiments, the quality of the match is measured by the matching accuracy. It is calculated as the number of correct matches, computed by a matching algorithm, over the total number of correct matches.

Table 1: Geometric graphs used in our experiments.

Road Network	|V|	|E|
California	1365	1990
Oldenburg	3494	4348
North America	7517	10088

We compare our graph matching algorithm **vem** to two other related graph matching algorithms **ged** and **heat**. Here, **heat** is the heat kernel embedding algorithm [26]. **ged** is the graph edit distance algorithm similar to the one proposed in [18]. For the **ged** algorithm, initially a distance matrix is created from the vertices of a graph to another. The distance between two vertices is computed by the cyclic string edit distance. Then, the Hungarian algorithm is used to compute the match between the two graphs.

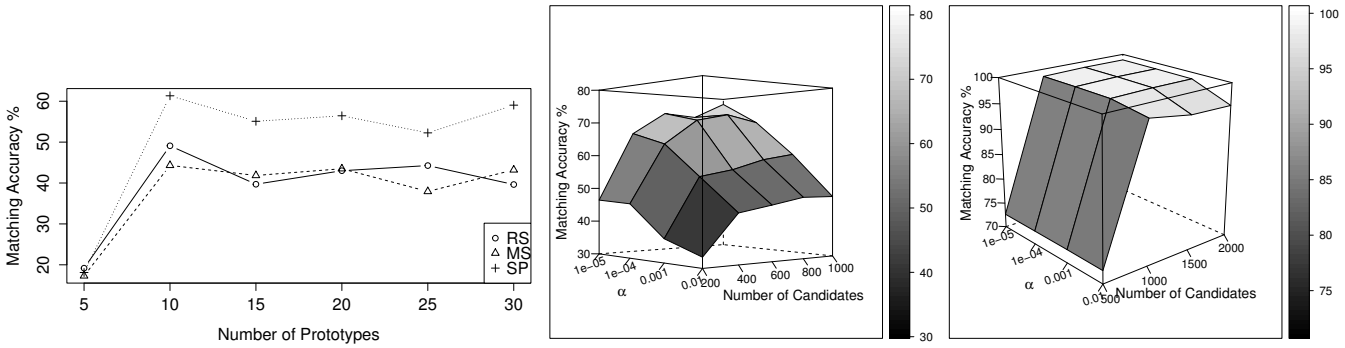
In the following Section 6.1, we study the effects of various parameters of our graph matching algorithm. This includes the prototype selection method, the size of the prototype set, the size of the candidate set, and the convergence threshold α. After that, in Section 6.2, we evaluate the matching quality followed by a scalability study in Section 6.3.

6.1 Parameters Analysis

In this section, we study the effects of different parameters of our algorithm on the matching accuracy. We detail the dataset in Section 6.1.1. In Section 6.1.2, we study the effect of the prototype selection method and the number of prototypes on the matching accuracy. Section 6.1.3 discusses the matching accuracy with different values for the size of the candidates set and the convergence threshold α.

6.1.1 Dataset

Two datasets are used for the parameter analysis: one for (sub)graph matching and the other for common subgraph matching. The subgraph matching dataset contains 5 subgraphs extracted from the California road network. We applied randomly spatial and a few structural distortions to those subgraphs [1]. The sizes of the subgraphs are 60, 92, 116, 123 and 128. We matched each of them against the California road network and averaged the matching accuracy. The common subgraph dataset consists of 5 different geometric graphs created from the California road network. Each geometric graph has a common subgraph with



(a) The effect of the number of prototypes and the prototype selection method on graph matching.

(b) The effect of the number of candidates and α on subgraph matching.

(c) The effect of the number of candidates and α on common subgraph matching.

Figure 5: The effects of different parameters on the matching accuracy.

the original California road network in addition to a non-common subgraph. To do this, we first partition the California road network into different clusters. The number of clusters used varies from 3 to 7. We apply spatial distortion to some clusters to make them non-similar [1]. For the five graphs, the number of distorted clusters are 1, 1, 2, 3, and 3, respectively. The rest of the clusters are considered as the common subgraph between the new created graph and the original California road network. The sizes of the common subgraphs are 451, 334, 546, 592, and 689 such that some common subgraphs are disconnected. We matched all 5 graphs against the California road network and averaged the matching accuracy.

6.1.2 Prototype Selection

We empirically evaluate the three prototype selection methods: random selection (RS), medoids selection (MS), and spanning selection (SP), see Section 4.2. For this test, we use the dataset of subgraph matching. Since the result of the MS method is affected by the initial medoids assignment, we run the k-medoids several times with different initial assignments. The prototypes that create the lowest within-cluster sum of distances are chosen. For both MS and SP, the average of the number of edges each prototype has is 3, whereas for the RS method, the average is $\frac{1}{2}k$ such that k is the total number of prototypes. As shown in Figure 5(a), the spanning selection method has the highest matching accuracy. On the other side, the RS method has nearly the same matching accuracy as MS. This result confirms the analysis reported in [16], which says that a random prototype selection method gives good results in many cases. Figure 5(a) also shows the effect of the number of prototypes on the matching accuracy. A high number of prototypes decreases the matching accuracy. This is because more non-representative vertex signatures are selected as prototypes. Also, a small number of prototypes does not cover the diversity of the vertex signatures extracted from the graphs. From this experiment, we conclude that 10 prototypes gives the best matching accuracy.

6.1.3 The Candidate Set

Here, we test the number of candidates that are selected at each iteration of the graph matching algorithm. Also, we test the convergence threshold α. We test the effects of those parameters on both subgraph and common subgraph matching. Figure 5(b) shows the accuracy for subgraph matching. 4 values used for α=$\{10^{-2}$,10^{-3},10^{-4},$10^{-5}\}$ and 5 values for the number of candidates {200,400,600,800,1000}. As shown in Figure 5(b), values of 10^{-4} for α and 1000 for the number of candidates give a good matching accuracy. It is obvious from that figure that a marginal increase in the matching accuracy occurs when using values higher than 10^{-4} and 1000. However, higher values for α and the size of the candidate set increase the time needed for the convergence. We conclude that a good value for the number of candidate is $10\times m$, where m is the average size of the subgraphs.

We also tested the effects of the candidate set and α on the accuracy of common subgraph matching. Four values are used for α as before. The four values for the number of candidates are {500,1000,1500,2000}. As shown in Figure 5(c), the best accuracy is with 10^{-4} for α and 1500 for the number of candidate set. As a result and for a graph with size m, a good value for the number of candidates is $1.3\times m$.

6.2 Matching Quality

In this section, we test the matching quality of our algorithm against the two matching algorithms **ged** and **heat**. In Section 6.2.1, we discuss the matching accuracy for subgraph matching. In Section 6.2.2, we analyze the results for the common subgraph matching.

6.2.1 Subgraph Matching

We use the California road network for subgraph matching. From this road network, we extracted 5 initial subgraphs. From those 5 subgraphs, another dataset of 40 subgraphs is generated. To do this, structural or spatial distortion are applied to the 5 initial subgraphs [1]. Distortion is applied at an increasing level. We computed the amount of distortion needed to make an initial subgraph non-similar to the distorted one. Then we divide this amount to create four levels of distortion such that each distortion level is represented by 10 subgraphs. We matched all subgraphs against the California road network and averaged the matching accuracy for each level of distortion. The dataset with zero distortion, in Figure 6, represents the average matching accuracy for the 5 initial subgraphs.

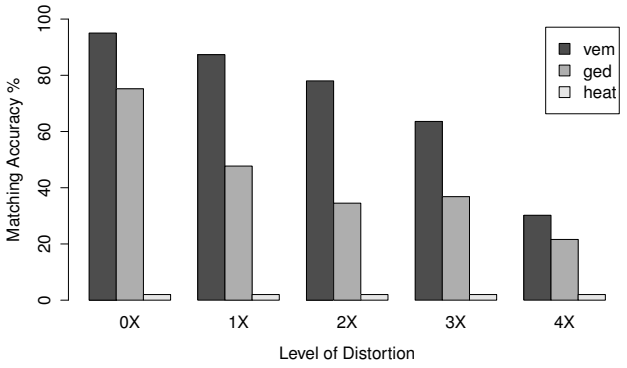


Figure 6: Subgraph matching accuracy.

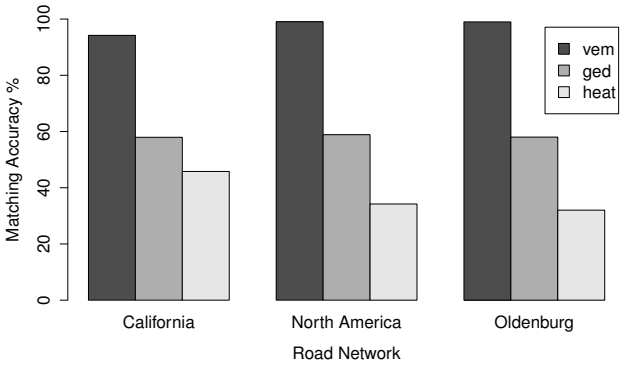


Figure 7: Common subgraph matching accuracy.

As shown in Figure 6, our proposed algorithm **vem** outperforms both related graph matching algorithms. The **heat** algorithm has nearly zero matching accuracy for all levels of distortions. This demonstrates the weakness of spectral approaches for matching graphs that differ in their sizes. On the other side, **ged** gave better results than **heat** even though the global connectivity of the graph is not considered by **ged**. We conclude that our proposed vertex similarity metric in Section 3.3, which is used by both **vem** and **ged**, is very efficient in finding similar vertices.

6.2.2 *Common Subgraph Matching*

The accuracy for common subgraph matching is tested by using all three road networks: California, North America, and the city of Oldenburg. Distortion is applied to each road network to create a new geometric graph [1]. Such a geometric graph has a common and a non-common subgraph with respect to the original road network. To do this, each road network is clustered into groups of vertices. Distortion is applied to some of the clusters to make them non-similar. The rest of the clusters are left without any distortion to create a common subgraph. Figure 7 shows the matching accuracy for common subgraph matching for all three road networks. The results show that our proposed algorithm outperforms the related graph matching algorithm. The matching accuracy for our algorithm is nearly 100% for the three road networks. Even for common subgraph matching, **heat** performs the worst. This shows that spectral approaches are sensitive to structural and spatial changes.

6.3 Scalability Study

Scalability with respect to graph size is one of the problems solutions to the graph matching problem face. In this section, we compare the scalability with respect to graph size for all three graph matching algorithms. We report the running time for common subgraph matching from Section 6.2.2. All experiments were carried out on an Ubuntu 12.04 platform with an Intel Core i5 CPU and 8GB RAM. For matrix operations we used *Armadillo*, which is a C++ linear algebra library [21].

Figure 8 shows all three road networks with the number of vertices for each. As one can see, our proposed algorithm scales well with respect to the graph size. **ged** comes in second place. Even though our algorithm and **ged** use the Hungarian algorithm, our algorithm uses a sparse similarity matrix that reduces the time needed to solve the assignment

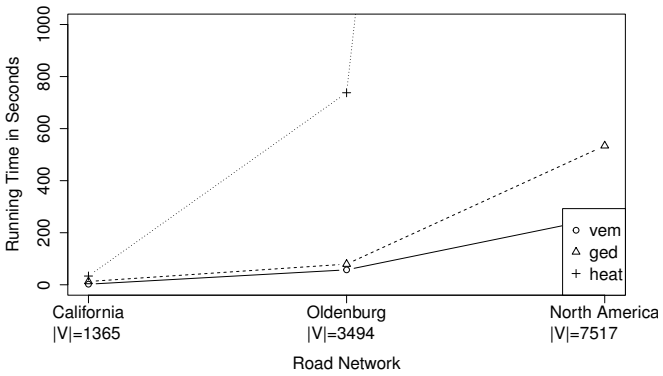


Figure 8: Scalability with respect to graph size.

problem by the Hungarian algorithm. The **heat** algorithm requires the longest running time. This algorithm uses matrix decomposition to compute the Eigenvectors for each of the graphs followed by SVD for their similarity matrix. However, matrix decomposition and SVD face scalability problems with dense and large matrices making **heat** the slowest algorithm.[3]

7. CONCLUSIONS AND ONGOING WORK

In this paper, we proposed an efficient framework for the matching of geometric graphs that differ in structure, spatial attributes, and labels. Our framework consists of three parts. First, a metric function to compute the distance between vertices of different graphs. Initially, a graph is decomposed into a multi-set of vertex signatures. Then, a spatial feature is extracted from each vertex signature. Based on such a spatial feature, our metric function utilizes the cyclic string edit distance to compute the distance between two vertices. The second part of our framework is a novel vertex embedding into the Euclidean space. Initially, a set

[3]Sometimes scalability in terms of memory consumption is also critical. For example, we were unable to test one of the related graph matching algorithm [5] on the road network dataset. Such graph matching algorithm uses a matrix of 2×10^{12} entries to find the common subgraph for the California road network. Storing such a matrix in RAM requires a huge amount of storage. This makes such an algorithm scale badly with respect to the memory consumption.

of representative vertex signatures (prototypes) is provided. Then, each vertex is embedded into the Euclidean space using the distances between its vertex signature and the set of prototypes. Embedding into the Euclidean space gives the possibility to compute the distance between two vertices in constant time. The third part of our framework is an iterative graph matching algorithm. Such an algorithm uses the similarity in the Euclidean space to create similar connected subgraphs. Using real datasets that were extracted from road networks, our algorithm outperforms related graph matching algorithms in terms of matching accuracy and running time.

Searching for similar graphs faces problems not only related to the sizes of the graphs but also to the number of graphs. As a result, recently graph indexing attracts more and more interests. Our embedding scheme can be easily used for geometric graph indexing. Instead of proposing a new graph indexing structure, embedding into the Euclidean space converts the graph indexing problem to the indexing of high-dimensional points. As a result, one can use *kd*-trees or R-trees for the indexing of geometric graphs.

So far, we focus on the matching of geometric graphs in 2D space. However, our framework can easily be extended for geometric graphs in 3D or even to non-geometric graphs. For this and for a specific type of graphs, a metric function that computes the distance between vertices should be provided. All other parts of our framework are not affected and scale well with respect to different graph types.

8. REFERENCES

[1] Distort geometric graphs. `https://code.google.com/p/itgrag/`.

[2] Road networks dataset. `http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`.

[3] X. Bai, H. Yu, and E. R. Hancock. Graph matching using spectral embedding and alignment. In *ICPR (3)*, pages 398–401, 2004.

[4] H. Bunke and K. Riesen. Towards the unification of structural and statistical pattern recognition. *Pattern Recognition Letters*, 33(7):811–825, 2012.

[5] M. Cho, J. Lee, and K. M. Lee. Reweighted random walks for graph matching. In *ECCV (5)*, volume 6315 of *LNCS*, pages 492–505, 2010.

[6] M. Cho and K. M. Lee. Progressive graph matching: Making a move of graphs via probabilistic voting. In *CVPR*, pages 398–405, 2012.

[7] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[8] D. H. Douglas and T. K. Peuker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Geographer*, 10(2):112–122, 1973.

[9] M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 14(3):398–408, 1984.

[10] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[11] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[12] D. Knossow, A. Sharma, D. Mateus, and R. Horaud. Inexact matching of large and sparse graphs using laplacian eigenvectors. In *GbR*, pages 144–153, 2009.

[13] M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. In *ICDM*, pages 258–265, 2002.

[14] M. Maes. On a cyclic string-to-string correction problem. *Informa. Process. Lett.*, 35(2):73–78, 1990.

[15] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[16] E. Pekalska, R. P. W. Duin, and P. Paclík. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208, 2006.

[17] R. Raveaux, J. Burie, and J. Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5):394–406, 2010.

[18] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[19] K. Riesen and H. Bunke. *Graph classification and clustering based on vector space embedding*. World Scientific Publishing Co., 2010.

[20] K. Riesen, X. Jiang, and H. Bunke. Exact and inexact graph matching: methodology and applications. In C. C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 217–247. 2010.

[21] C. Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, Australia, October 2010.

[22] A. Sanfeliu and K.-S. Fu. A Distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):353–362, 1983.

[23] G. L. Scott and H. C. L. Higgins. An algorithm for associating the features of two images. *Proceedings of Royal Society of London*, B-244:21–26, 1991.

[24] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

[25] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[26] B. Xiao, E. R. Hancock, and R. C. Wilson. A generative model for graph matching and embedding. *Computer Vision and Image Understanding*, 113(7):777–789, 2009.

[27] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: on approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[28] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao. Efficient shortest paths on massive social graphs. In *CollaborateCom*, pages 77–86, 2011.

[29] Y. Zhu, L. Qin, J. X. Yu, Y. Ke, and X. Lin. High efficiency and quality: large graphs matching. In *CIKM*, pages 1755–1764, 2011.