Noname manuscript No.
(will be inserted by the editor)

Network Convexity Algorithms via Graph Embeddings

Ilya Makarov · Dmitrii Gavrilev · Serafim Korovin · Lovro Šubelj

Received: 07/2021 / Accepted: date

Abstract In this paper, we propose a new approach for fast convex hull construction for a subset of nodes on a network using graph embeddings. We used the well-known convexity concept in the embedding space to solve a similar problem for geometric learning on a graph, thus optimizing the process of finding all the shortest paths in the induced subgraph. Several encoder-decoder architectures were tested, from which the L1 decoder proved to preserve metric characteristics of networks with low convexity. As a result, it enabled us to search and verify convexity of subgraphs in a faster and more convenient way than standard algorithms on graphs.

Keywords Network Convexity · Graph Embedding · Shortest Path · Network Science

1 Introduction

Recent years have seen substantial growth of interest in the technologies and algorithms that study data and provide its efficient representation. One of the most widespread data representations is a graph, which is usually represented as a collection of nodes and edges connecting them. Nowadays, graphs are ubiquitously used for organizing information: social interactions, scientific citations, road maps, etc. For example, in online social networks, an edge can display that users, who are represented by nodes, are following each other. Thus, graphs are useful for representing relationships between objects and patterns within data.

Convexity is a property common to various mathematical objects. Nevertheless, in the context of networks [22] it drew attention only recently and, to our knowledge, still remains insufficiently researched. Convexity is a core property underlying the shortest path problem. Consider a connected graph G and a subgraph on a subset

Ilya Makarov
HSE University, Moscow, Russia
University of Ljubljana, Ljubljana, Slovenia
Artificial Intelligence Research Institute, Moscow, Russia
E-mail: iamakarov@hse.ru

Dmitrii Gavrilev
HSE University, Moscow, Russia
E-mail: dmitrygavrilyev@gmail.com

Serafim Korovin
Vrije Universiteit, Amsterdam, Netherlands
HSE University, Moscow, Russia
E-mail: serafim.korovin@gmail.com

Lovro Šubelj
University of Ljubljana, Ljubljana, Slovenia
E-mail: Lovro.Subelj@fri.uni-lj.si

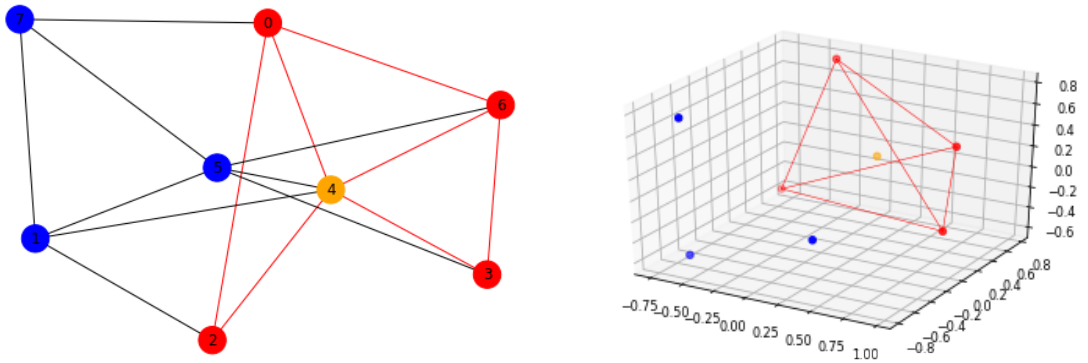


Fig. 1: An example of (Left) graph and (Right) its 3-dimensional embedding. The base nodes of the convex hull are colored red. An orange node denotes another node which belongs to the convex hull. The edges which are induced by the convex hull are colored red. The induced subgraph is convex, since it contains every geodesic path between its nodes. Nodes that do not belong to the subgraph are colored blue.

of nodes S. The subgraph is said to be convex if all geodesic paths between the nodes in S are entirely included into S [22]. In this paper, we also use the definition of convexity in a metric space. A set S is convex if for any $x_1, x_2 \in S$ and $\forall\ \theta$ such that $0 \leq \theta \leq 1$ we have $\theta x_1 + (1-\theta)x_2 \in S$ [3]. Consequently, finding and constructing convex subgraphs is also computationally intractable for large networks. We can also define the convex hull in the context of graphs similar to the one in a metric space: the convex hull of a set S is the minimal convex set comprising S.

However, most graph analysis algorithms are too computationally or memory-heavy, especially when applied to real-world networks with millions of nodes and links. For example, to measure the distance between two nodes of an unweighted graph, we have to solve the problem of finding the shortest path, which depends on the number of vertices and edges ($O(|V|+|E|)$ with applying breadth-first search). On the other hand, machine learning methods, which provide solutions to many novel problems on graphs, expect the data to be in a vector space of independent features, hence cannot be applied on graph structures in a straightforward manner. Therefore, graph embeddings were introduced as a method of data representation that maps the nodes of the graph to low-dimensional vectors, conserving their core properties presented by certain similarity or distance functions.

For the purposes of demonstration, consider Figure 1, which depicts a graph with 8 nodes and one of the possible embeddings. Consider the problem of searching the convex hull of nodes with labels 0, 2, 3 and 6. In order to do this, standard graph algorithms could be applied. The result would be the subset of base nodes combined with node 4. The subgraph induced by the obtained subset contains all geodesic paths between its nodes, therefore, it is convex. It is also minimal, meaning that the exclusion of node 4 would remove convexity. The same problem could be approached via graph embeddings. The right side of Figure 1 illustrates the 3-dimensional embedding space of the graph from the left side, in which the nodes are mapped to the points. If graph embedding indeed preserves convexity, then the problem of searching for the convex hull is reduced to constructing such in a metric space.

The aim of the paper is to research and devise a method for faster convex hull construction in networks. The model is expected to preserve convexity as well as metric properties. This enables us to solve the problems of search and evaluation of convex subgraphs with much faster algorithms having lower computational costs. Since convexity is an inherent property of real networks, the research may prove useful in various applications of network classification as well as distinguishing the networks from random graphs. Another direction for research involves community detection in networks based on their convex skeleton, which can be viewed as a

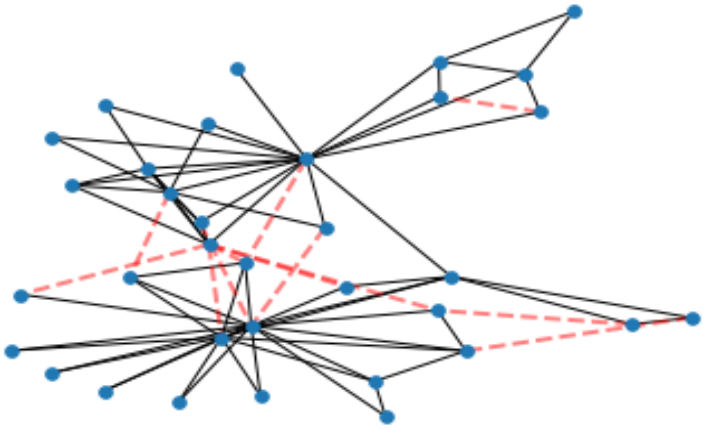


Fig. 2: An illustration of the convex skeleton of a network (Zachary's Karate Club) obtained by targeted removal of edges (see section 3.1.3). The removal of 15% of edges (dashed and red) allowed us to achieve convexity X of value 0.90, while the original network has X of value 0.58.

generalization of a spanning tree. Furthermore, the research results might be applied to the problem of finding the optimal subgraph of roads in transportation networks [22, 26].

This paper represents the continuation of the previous work by Šubelj [22, 26] and Korovin [19]. The main contribution of our work consists of:

1. Applying and testing various graph embedding models on convex set construction problems in networks;
2. Proposing an encoder-decoder architecture which works best out of the models that have been tested;
3. Highlighting the class of high-convexity networks, on which graph embeddings work best.

The rest of the paper is structured as follows. In Section 2, we overview the related work relevant to our research area. Section 3 includes the description of various methods and techniques for graph embedding construction and convexity verification. Section 4 contains the outline of an experimental part where our model is tested. In Section 5, we conclude with the closing remarks as well as outline directions for future research.

2 Related work

This chapter briefly overviews research relevant to the topic of our study. First, we review studies regarding convexity on networks. Then, we proceed to discuss existing graph embedding techniques.

2.1 Convexity on networks

Convexity on graphs, originally suggested by Marc and Šubelj [22], was studied in three distinct forms: global one, which refers to a tree of cliques and grows at a noticeably low pace, local one, which is more typical for random graphs rather than for real networks, and regional one that refers to any partly convex heterogeneous network. Furthermore, the authors show that convexity is an inherent structural property of networks. According to their research, random graph models like Erdos-Renyi, Barabasi-Albert and Watts-Strogatz fail to recreate convexity in networks (see [24] for random graphs description).

Another novel term of a convex skeleton was introduced in [26]. The author defines it as "the largest high-convexity part of empirical networks obtained by removing the least number of edges". It was shown that they retain numerous graph characteristics, such as degree distribution and clustering. Convex skeletons

also preserve communities, which can be defined as subsets of densely connected nodes. Additionally, a few different approaches for extracting convex skeletons and their applications to various networks were presented. For instance, regarding scientific collaboration networks, these approaches can serve the following goals [27]:

- Revealing influential researchers which were not trivial to detect in the original networks by computing centralities on the convex skeleton;
- Optimizing research funding by excluding the ties that are not presented in the convex skeleton;
- Recommending potential collaborators.

To that end, convex skeletons can be deployed as a lightweight abstraction of networks.

2.2 Graph embeddings

In [11,21], authors present surveys on the graph embedding algorithms and methods. The authors' overviews include a taxonomy of methods (division into three main categories: matrix factorization, random walks and Deep Learning), their comparison in time complexity, preserved properties and embedding under different scenarios (supervised and unsupervised learning, learning embeddings for homogeneous and heterogeneous networks, etc.). In what follows, all methods can be unified under the encoder-decoder framework. An encoder, ENC, maps graph nodes to the embedding vectors. The notation in this section is common regardless of the approach: by v_i and $z_i = ENC(v_i)$ we refer to a node and its respective embedding. In addition, node similarity function $sim(v_i, v_j)$ is defined to measure similarity between nodes in a graph, and decoder function $DEC(z_i, z_j)$ is created to represent the pairwise similarity values of embedding vectors from the embedding matrix. For the similarity function, we propose two ways of definition. First is through the adjacency matrix of the graph, i.e.

$$sim(v_i, v_j) = A_{ij}$$

And the second way is through the Floyd-Warshall matrix:

$$sim(v_i, v_j) = FW_{ij},$$

where FW_{ij} denotes the length of the shortest path between the i-th and j-th nodes of the graph. While for the decoder, the choice can consist of the functions below.

As the dot product

$$DEC(z_i, z_j) = z_i^T z_j$$

As the Euclidean distance (L_2 distance):

$$DEC(z_i, z_j) = \sqrt{\sum_{k=1}^{d} \left(z_i^k - z_j^k\right)^2}$$

As the taxicab distance (L_1 distance):

$$DEC(z_i, z_j) = \sum_{k=1}^{d} \left|z_i^k - z_j^k\right|$$

Or as cosine similarity:

$$DEC(z_i, z_j) = \frac{z_i^T z_j}{||z_i|| \cdot ||z_j||}$$

In Figure 3, we schematically depict the process of training embeddings. The nodes of a graph are encoded in the columns of matrix Z. Thus, Z has d rows, which is equal to the dimensionality of a vector space. Matrix Z is updated iteratively until the loss function, defined by the decoder, reaches its minimum.

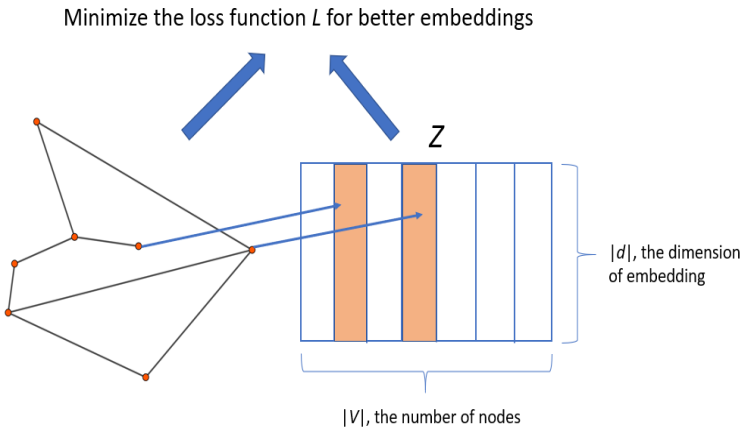


Fig. 3: Graph and embedding matrix

Node2vec One of the most salient methods in representation learning is node2vec. Grover et al. [10] present node2vec as a framework for data representation in networks which learns a mapping that maximizes the probability of preserving network neighborhoods, a novel notion introduced by the authors. The network neighborhoods are acquired through one of two different neighborhood sampling strategies: breadth-first sampling, which considers only immediate neighbors, and depth-first sampling, which includes nodes sampled at increasing distances from the source node. While the former generates the global view of the graph, the latter produces the local one. If we concentrate on the local view, the neighborhood will consist primarily of the closest neighbors of v, whereas if we take the global view, it will include mostly paths of nodes that are far from v. Simply speaking, the nodes are supposed to have similar embedding vectors if they tend to co-occur on a random walk over the graph [11, 10]. Therefore, the loss function is the following:

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(DEC(z_v, z_u)) \to \min,$$

where $N_R(v)$ is a multiset of nodes visited on random walks starting from the node v with the neighborhood sampling strategy R, and the decoder function is:

$$DEC(z_i, z_j) = P(i|z_j) = \frac{e^{z_i^T z_j}}{\sum_{v_k \in V} e^{z_i^T z_k}}.$$

Matrix factorization Another approach involves factoring matrices. Let S be the matrix consisting of pairwise similarities ($S_{ij} = sim(v_i, v_j)$). If the goal is to capture the similarity between the nodes in their embeddings, then ideally $DEC(z_i, z_j)$ would be equal to $sim(v_i, v_j)$ for all indices i and j. This goal can be achieved by optimizing the loss function

$$L = \sum_{v_i, v_j \in V} ||DEC(z_i, z_j) - sim(v_i, v_j)||^2 \to \min, \tag{1}$$

which, in other words, indicates how close matrix S and the matrix constructed from the values of DEC are. The embeddings can be derived via directly factoring the similarity matrix S or by performing stochastic gradient descent (SGD) on the loss function and the trainable parameters, which in this case, are just elements of the embedding matrix Z.

Particularly, we employed one of the matrix factorization algorithms presented by [23], High-Order Proximity preserved Embedding (HOPE). It focuses on preserving asymmetric transitivity, which is a property of

directed graphs. Although this research is not aimed to specifically study directed networks, the fact that HOPE operates with high-order proximities at its core can be useful. The variety of proximity measures is rather wide: Katz Index [15], Rooted Page Rank, Common Neighbors and Adamic-Adar. These measures are used to construct a proximity matrix S. Then the desired embeddings are obtained by performing the generalized singular value decomposition (via Jacobi-Davidson type algorithm [12]) on a proximity matrix. The performance of this algorithm is fairly high due to its scalability, since its complexity increases linearly with the number of edges.

High-order proximities can be passed as the similarity function. The result of this factorization is HOPE embeddings. The crucial difference between HOPE and classical matrix factorization techniques is that HOPE is designed for approximating asymmetric transitivity. Thus, two embeddings representing the source and target are assigned for each vertex. In this case, the decoder function is the dot product of the source and target vectors:

$$DEC(z_i^s, z_j^t) = z_i^{sT} z_j^t.$$

Graph neural networks The methods we have discussed have been shallow so far in the sense that obtained embeddings do not take the graph structure explicitly into account. Furthermore, Hamilton et. al incorporate various neural network models, such as Graph Convolutional Networks (GCN) and GraphSAGE, into a category called *neighborhood aggregation algorithms*. These algorithms take both node feature matrix and adjacency matrix as input. Hence, they account for individual node parameters and the graph structure. The output of each layer of a network is a matrix of node embeddings. A node is assigned its embedding as a combination of the aggregated neighborhood and an embedding corresponding to the previous layer. The aggregation function can be viewed as a convolutional kernel and may differ from model to model. In particular, a variation of GNNs, Graph Convolutional Network [17], uses an aggregation rule in the form of

$$Z^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}\right),$$

where $Z^{(l)}$ and $W^{(l)}$ are a matrix of embeddings and a trainable weight matrix in the l-th layer respectively. $\tilde{A} = A + I_n$ is the adjacency matrix of the given graph modified such way that it has self-loops and $\tilde{D}$ is its degree matrix. σ is a (non-linear) activation function. Stacking of convolutional layers allows to propagate information not only from the local neighborhood but also from the distant neighbor nodes.

3 Methodology

In the following section, we discuss methods regarding convex set creation in graphs and vector spaces. We review the overall approach towards the construction and covers the computational complexity of the algorithms. First, we present a naive convex hull algorithm in graphs and provide modifications, which allow enhancing its performance. Additionally, we describe an algorithm for the convex skeleton extraction. Then, we examine convex set methods in two vector spaces, Euclidean and taxicab, which are induced by L2 and L1 norms respectively.

3.1 Convex set creation

Below, we present a set of algorithms used for constructing convex sets on networks. We start with a naive algorithm, which strictly follows the definition of the convex hull. Then, we optimize it by discarding excess computations. We also provide an algorithm for extracting the convex skeleton.

3.1.1 Naive convex hull construction

Algorithm 1 solves the problem of convex hull construction. Let S be a set of nodes, on which we should build the convex hull. We can accomplish it by simply following the definition of a convex set in graphs: the final set must comprise all the geodesics between its elements. Moreover, we should check its minimality in order for it to satisfy the definition of the hull. It can be done by terminating the algorithm once the subgraph becomes convex.

```
while a new node is added to S do
    for all pairs in S do
        for all shortest paths between the pair do
            if a node from the path is not in S then
                S = S ∪ node;
            end
        end
    end
end
```

Algorithm 1: Naive convex hull

The given algorithm is correct, as it exactly follows the definition of the convex set. However, its working time is on average $O(n^5)$ in the case of $|S| = n$ of the line graph. Therefore, we have implemented a quicker solution.

3.1.2 Improved convex hull construction

In Algorithm 2, we precompute the distance matrix and use its values to check the distance between the nodes. In the case of unweighted networks, the distance matrix can be constructed by running breadth-first search from each node. Despite the $O(n(n+m))$ complexity, we only have to calculate it once, which gives this approach a substantial advantage in running time.

```
while a new node is added to S do
    for all pairs A, B in S do
        for all nodes C except nodes in S do
            if dist(A,B) > 1 and dist(A,C) + dist(C, B) = dist(A,B) then
                S = S ∪ C;
            end
        end
    end
end
```

Algorithm 2: Improved convex hull

In essence, our algorithm performs the same steps as literal implementations. The only difference is the replacement of the straightforward shortest paths check with the $O(1)$ check in the distance matrix. The complexity for the following modification is $O(n^3)$ in the worst case.

3.1.3 Convex skeleton extraction

In order to extract convex skeletons, [26] proposes the following strategy based on optimizing the average node clustering coefficient.

```
S = initial_graph;
i = 0;
candidates = S.edges;
while i <number of edges to be removed do
    max_change = -∞;
    if candidates is empty then
        break;
    end
    for u, v in candidates do
        if ΔC(u) + ΔC(v) >max_change then
            target = u, v;
            max_change = ΔC(u) + ΔC(v);
        end
    end
    if S \ target is connected then
        S.edges.remove(target);
    end
    candidates.remove(target);
    i++;
end
```

Algorithm 3: Convex skeleton extraction

Algorithm 3 consists of targeted removal of edges. Let ΔC_u and ΔC_v denote the change in the clustering coefficients of nodes u and v after the removal of edge (u,v). On each iterative step, we remove an edge with the greatest sum of ΔC at its endpoints. Note, since convexity implies connectivity, we need to check that the graph remains connected after each step. The number of iterations should be adjusted to maximize a measure of convexity X.

The procedure of computing X involves generation of convex subsets. First, the subset is initialized with a random node. Then, we append one node with probability proportional to the number of its neighbors that contain in the current subset. Next, we ensure its convexity by augmenting the subset to its convex hull. We keep expanding the subset until it induces the network. The procedure must be repeated several times to ensure conformity to initial conditions. Once this routine is completed, convexity of a network can be measured as

$$X = 1 - \sum_{t=1}^{n-1} \max\{\Delta s(t) - 1/n, 0\}, \tag{2}$$

where $\Delta s(t)$ is the average number of appended nodes at step t and n is the network size.

3.2 Convex sets in a Euclidean vector space

Having constructed both the embedding and the convex set on the graph, we proceed to build the convex set in the embedding space with Euclidean distance. Our general approach towards the acquirement of the convex set in the embedding space is following. First, we build the convex hull for the given set of points. The set may vary depending on the metric we intend to use. Next, we subdivide the convex hull into simplices. A k-simplex is a set of points such that $\{\sum_{i=0}^{k} \theta_i u_i \mid \sum_{i=0}^{k} \theta_i = 1 \text{ and } \forall i\ \theta_i \geq 0\}$. In other words, we apply triangulation. According to Carathéodory's theorem, if a point p of $\mathbb{R}^d$ belongs to the convex hull of S, then p can be represented as a convex combination of at most $d+1$ points in S [5]. Thus, Carathéodory's theorem states that every point in a convex hull lies in a k-simplex with nodes in the convex hull, where $k \leq d+1$. Therefore, the next step is to

locate the points that belong to the simplices. These points form the convex set. After that, we proceed to the calculation of the metrics. We provide a review of the concrete algorithms in Appendix.

It is worth mentioning that the dimension of a vector space has a lower bound, which is defined by individual graphs. If a graph comprises a k-clique, the embedding must have at least $k-1$ dimensions. Since our embedding's target is tailoring the distances between the vectors to the lengths of the shortest paths, the only matching way to arrange the vectors of a clique is in a form of a regular simplex. $(k-1)$-simplex requires k nodes, therefore, the dimensionality of the embedding must correspond to the dimensionality of the simplex and equal to $k-1$.

The shown fact impedes further progress since the vast majority of the convex hull algorithms were designed for 2- and 3-dimensional planes. Besides, this may become important in real networks since they often include large cliques. However, a straightforward brute-force k-clique detection algorithm takes $O(n^k k^2)$ time, reaching an exponential time in the case k varies [6], which is too heavy for large-scale detection.

It goes without saying that the higher the number of dimensions is, the fewer mistakes the embedding makes. However, we have to compromise with the technical limitations of the tools for convex hull construction. Performing triangulation in high-dimensional spaces comes with extreme computational costs: the complexity of the Quickhull and Delaunay triangulations explodes with the increase of dimensionality. One way would be to train a low-dimensional (2D or 3D) embedding despite the theorem above. The embedding built this way is almost guaranteed to be imperfect distance-wise, however, it might save resources for convex hull calculation in space.

Another way is to train a high-dimensional embedding and apply dimensionality reduction algorithms like PCA and t-SNE after training the embedding and before the calculation of the convex sets in space. It would be especially useful when it is known that the structure of the graph is predisposed to have large cliques. However, t-SNE uses a non-convex objective function, which is minimized using a randomly initiated gradient descent optimization. This leads to several runs on the same parameters producing different results, which does not happen in the case of application of PCA. On the other hand, PCA uses $O(n^2 d + d^3)$ in computation complexity (where n is the number of data points and d is the number of dimensions), whereas t-SNE requires only $O(n^2)$ [20].

3.3 Convex sets in a taxicab space

In the following section, we refer to a real vector space with L_1 norm as a taxicab space. In other words, we consider a vector space with a norm

$$\|z\| = \sum_{k=1}^{d} |z^k|.$$

Consequently, the distance between two points x and y in a d-dimensional taxicab space is

$$d(x,y) = \|x-y\| = \sum_{k=1}^{d} |x^k - y^k|.$$

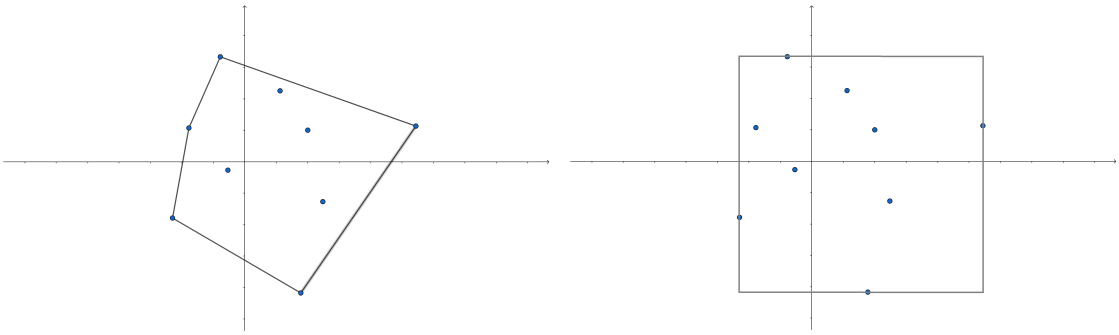
A simplex in a d-dimensional taxicab space is an orthotope whose facets are parallel to axes. Let π_k be the orthogonal projection onto axis k. Then, we can define the convex hull of a set of points S as

$$\operatorname{conv}(S) = \left\{ \min \pi_1(S) \le x^1 \le \max \pi_1(S), \ldots, \min \pi_d(S) \le x^d \le \max \pi_d(S) \right\}.$$

Therefore, in order to construct the convex hull in a taxicab space, one needs to find the coordinate-wise minima and maxima of a given set. A task like this can be solved by looking at each vector's coordinate once. Hence, the computational complexity is $O(nd)$. Due to its linear complexity, building convex sets with L_1 distance is fast even on higher dimensions as opposed to the convexity algorithms in a Euclidean space.

Another advantage of a taxicab space lies in the fact that there are multiple geodesics between two points, whereas geodesics are always unique in a space with L_2 distance. Moreover, we are no longer limited by the

dimension of space. In a Euclidean space, the cardinality of a set needs to be greater than $d+1$ for it to have a convex hull with non-zero volume. The same does not hold true for a taxicab space: the convex hull of set S has positive volume unless S is affinely dependent. Figure 4 illustrates the difference between convex hulls in Euclidean and taxicab spaces. Note that a convex taxicab set tends to encompass more space.



(a) Euclidean (b) Taxicab

Fig. 4: The difference between convex hulls in a Euclidean space and a taxicab space.

4 Experiments

In this section, we describe the experiments we have conducted. First, we provide information about datasets on which we have tested our methods. Then, we introduce metrics that show the ability of embeddings to recreate and preserve convexity of networks. Next, we measure the performance of algorithms and compare them to each other. We proceed to train the embedding models in the following section 2.2. Furthermore, we evaluate the quality of these models by measuring scores introduced in section 4.2 and highlight the best approach. We demonstrate that mapping nodes to a taxicab space shows better results at capturing convex hulls in graphs than in a Euclidean space.

4.1 Datasets

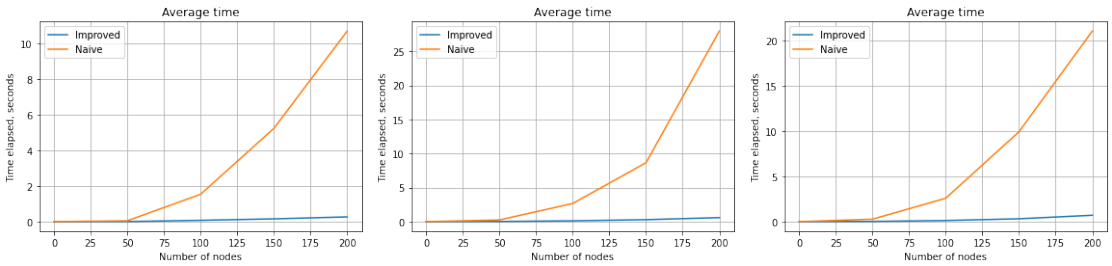
We tested the aforementioned methods on several datasets.

The first dataset we used is Zachary's Karate Club dataset [8]. At present, it is one of the most frequently utilized datasets for testing the correctness of graph algorithms' performance. Zachary's Karate Club comprises 34 nodes divided into two communities. Approximately one half of the vertices form a community around node 0, the other half is based around node 33.

Another dataset that we employed is Enron-only [25], which is a network of the email correspondence between managers of Enron. It includes 143 nodes and 693 edges. We employed this dataset in order to measure the performance of our algorithms. For this purpose we used Erdos-Renyi [7] random graph generation to obtain various graphs of different sizes: 100, 150 and 200 nodes.

4.2 Metrics

We propose two ways of measuring embedding accuracy. One compares two independently built convex sets, another assesses the similarity of the convex hull of the convex set built on the graph after projecting it



(a) base of 2 (b) base of 3 (c) base of 4

Fig. 5: Average time of convex hull construction versus the number of nodes in graph. Orange and blue lines correspond to naive implementation (see Algorithm 1) and our improved implementation (see Algorithm 2), respectively.

onto the vector space. The former measures the ability of the embeddings to recreate convexity, whereas the latter estimates how well they preserve convexity.

Comparison For the following method, we build two convex sets: one on the graph, and another in the embedding space. First, we choose the *base* of our sets — a set of randomly picked nodes. Following that, the convex hulls are built from the base both on the graph and in the embedding space. For the first convex hull, we employ the aforementioned algorithm except for the initialization part - we use the base as a starting set instead of picking random nodes. As for the convex hull construction in space, an in-depth look is provided below.

Having retrieved two sets s_1 and s_2, we use the Jaccard coefficient to measure the similarity of the two sets:

$$J(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$$

Projection For this method, in contrast to the previous one, we first build only one convex set on the graph. We may use any variation of the convex hull algorithm since this is the only essential set. Once we have the graph convex set obtained, the vector representations of the corresponding nodes are gathered from the embedding matrix. Following that, we build the convex hull S of those points, search for the additional points which lay inside its triangulation and call them the *error set*. Then we measure the accuracy of the set's recreation with the following formula:

$$precision(S, error) = \frac{|S| - |error|}{|S|},$$

which essentially measures the percentage of the correct nodes in the convex hull in the embedding.

To assess the metric for the whole embedding, we randomly grow several convex sets. Then, we evaluate the accuracy of their recreation and average scores with respect to the set sizes.

4.3 Time comparison

In this section, we compared the naive implementation of the convex hull construction and our improved algorithm which were described in Sections 3.1.1 and 3.1.2. We tested it on Zachary's Karate Club dataset and random graphs up to 200 nodes. The convex sets were constructed based on 2, 3, and 4 random nodes. Figure 5 shows performance of both algorithms for convex hull construction. As it can be seen from the graphs, our algorithm shows a fairly stable performance and is clearly superior to the naive implementation, with just around 1s runtime on the 200 nodes graph.

Moreover, since the aim of our research was to devise an embedding, which would enhance the performance of the convex hull construction, we compared the runtimes of building a convex set on the graph and in the

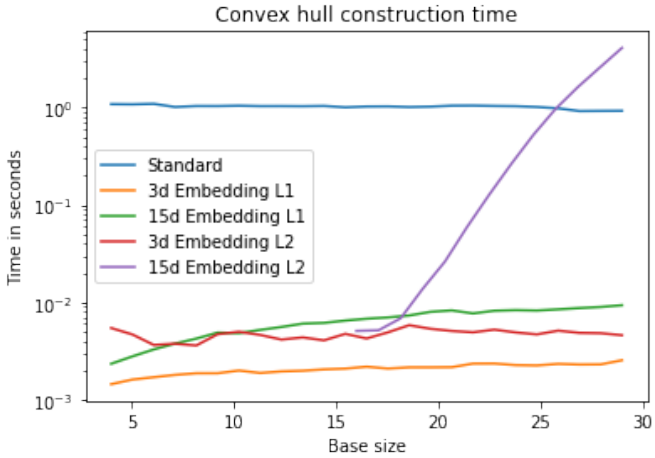


Fig. 6: Convex hull construction time (in logarithmic scale) versus the size of a base set. The blue line corresponds to the improved convex hull algorithm in graphs described in Section 3.1.2. The rest of the lines represent convex hull algorithms in Euclidean and taxicab spaces. The 15-dimensional L2 algorithm (purple) starts from a base of size 16 because we discard convex sets of zero volume in a Euclidean space.

embedding space. Our research also included investigation into the impact of dimensionality and the norm of a vector space. For this purpose, we used Enron-only dataset. We trained four GCN models with the following pool of hyperparameters: L1 and L2 decoders, 3 and 15 dimensions. The loss function was fixed as Eq. 1. As for the similarity function, we chose the distance matrix obtained by the Floyd-Warshall algorithm. The initial layer $Z^{(0)}$ (feature matrix) was common for every model — the identity matrix of size 143. Each hidden layer was activated by ReLU.

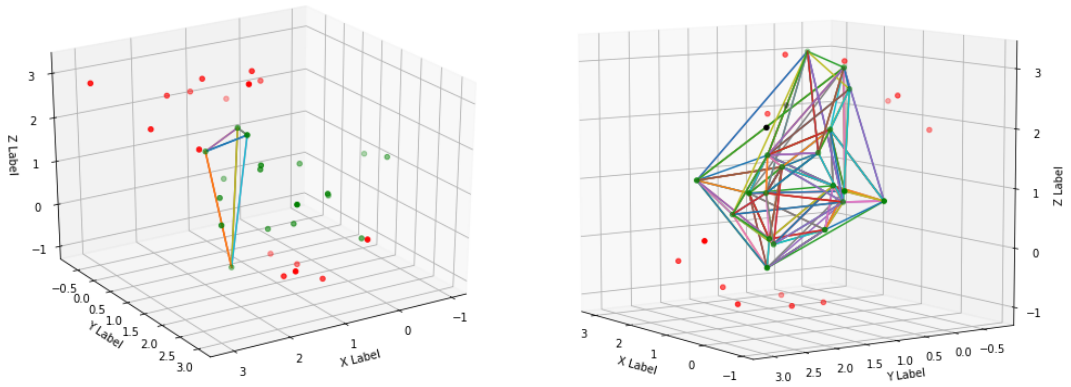
For each value of the base size, we sampled 5 sets and constructed the convex hull on the graph and in vector spaces. Next, we calculated the average over these samples, which is depicted in Figure 6. As can be seen, all algorithms in vector spaces, with the exception of 15-dimensional L2 hull construction, perform faster than the standard graph algorithm from Section 3.1.2. Note that the construction time for the 15-dimensional L2 algorithm increases rapidly and, at the base size 26, exceeds the time for the standard algorithm. Overall, taxicab algorithms performed better than Euclidean algorithms. To that end, we have managed to enhance the speed of the convex sets' construction, but the quality of the embedding, according to the comparison score, was not adequate — the best result was obtained by 3d GCN with L1 decoder, yet it still showed only 0.655 on average over all samples.

4.4 Embedding models

In this section, we analyze the output of the two methods for embedding quality assessment. We combined various parameters to find the best solution. The chosen dataset was Zachary's Karate Club. The list of tested algorithms with their respective results is presented in Table 1. What follows is the description of the pipeline on which the algorithms were tested.

First, we sample 50 subsets of size 4. In order to evaluate the comparison score, we construct the convex hulls from the sampled subsets in the embedding space. After that, we locate the points which belong to the hull and calculate the metric. In Figure 7a, the nodes which belong to the convex set are colored green and the rest are colored red. As for the projection method, we build the convex hull in the embedding space from the convex hull obtained on the graph as shown in Figure 7b . After that, we find the error set and calculate the

metric. Then, for each method, we evaluate the average score over all samples. For nondeterministic training algorithms, such as SGD, we repeat this procedure 10 times.



(a) Comparison

(b) Projection

Fig. 7: The difference between the two approaches of assessing embedding quality.

For random walks, we applied Node2Vec algorithm with 3 dimensions, $p = 1$, $q = 20$ (i.e. BFS-like walks), with walk length equal to the graph's diameter and the number of walks of 750. In addition, we obtained high dimensional embeddings (with dimensions numbering in 8, 16 and 30) and applied dimensionality reduction algorithms. Reduced embeddings have lower projection scores than an embedding trained directly with 3 dimensions. The comparison score is roughly on the same level. To test the performance of these dimensionality reduction algorithms, we obtained 8-, 12-, 16- and 30- dimensional embeddings through Node2Vec and applied them to reduce the number of dimensions to 3. Comparing the two techniques, we find PCA to be more accurate and stable than t-SNE based on the projection score.

With the matrix factorization approach, we optimized a matrix Z through stochastic gradient descent with the loss function described by Equation 1 and convergence tolerance at 0.1. The similarity matrix was fixed as the Floyd-Warshall matrix, except for the algorithm that uses the cosine function as the decoder. Out of the decoders we tested, Euclidean distance showed the best results on both metrics. Further, we analyzed the influence of the initial value of matrix Z. Initialization of Z with the embedding obtained through Node2vec did not show significant changes in results.

Furthermore, for HOPE embeddings, we first searched for the best tuning parameter for a proximity matrix, based on the theoretical relative approximation error of proximity. With Rooted PageRank, we found that the approximation error decreases monotonously with the increase of damping factor α. Thus, we set α to be close to 1. For the Katz matrix, we observed that the minimum is reached when the decay parameter β is equal to the inverse spectral radius of the adjacency matrix. Next, we derived embeddings via singular value decomposition of proximity matrices. Here we did not apply the JDGSVD algorithm, as the authors originally suggested, since the data was small enough for SVD to perform in a reasonable time. Although proximity matrices were well approximated (except for Common Neighbors), the comparison scores were still extremely poor.

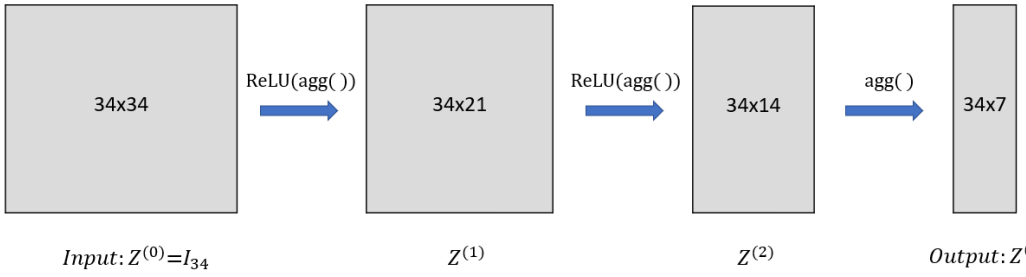


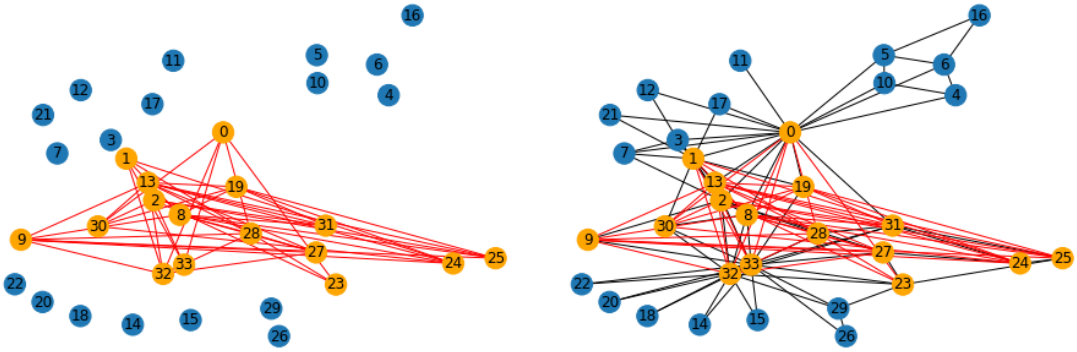
Fig. 8: The architecture of the GCN model built for Zachary's Karate Club. We employed two hidden layers of dimensions with ReLU as an activation function 21 and 14. The output embeddings of dimension 7 are gathered without applying non-linear activation. To motivate the reasoning behind the small number of layers, we can bound it by the diameter of network, which is equal to 5.

Table 1: Projection and comparison scores of different models. Values in bold correspond to best results.

Type	Projection	Comparison
Node2vec	0.909	0.275
Node2vec (t-SNE 8)	0.744	0.279
Node2vec (PCA 8)	0.793	0.276
Node2vec (t-SNE 16)	0.734	0.278
Node2vec (PCA 16)	0.820	0.278
Node2vec (t-SNE 30)	0.745	0.277
Node2vec (PCA 30)	0.854	0.274
SGD Euclidean	0.938	0.301
SGD dot product	0.848	0.275
SGD cosine (adjacency matrix)	0.909	0.281
SGD Euclidean (Node2vec init)	0.933	0.296
HOPE RPR (source)	0.827	0.261
HOPE RPR (target)	0.834	0.253
HOPE Katz	0.862	0.259
HOPE CN	0.814	0.254
GCN Taxicab	**0.971**	**0.353**
GCN Taxicab (discretized)	0.794	**0.608**

As for Graph Convolutional Networks, we aimed to minimize the loss function described by of Equation 1. However, the decoder function was replaced with taxicab distance this time. The optimizer was set to Adam [16]. The architecture of our model is illustrated in Figure 8. We find this particular model to perform better than any other method we have tested. Besides, we have tested the discretized version of the embeddings obtained by GCN. The value of projection score 0.971 was quite close to 1. It means that this model is able to preserve convexity of sets. Additionally, we evenly rounded the embedding coordinates to one decimal place, which led to a significant improvement in the comparison score.

As can be seen from Table 1, the projection method showed better scores than the comparison method on all of the approaches. The obtained results signify that our embedding is able to preserve convex sets, but hardly can reconstruct them within itself. In other words, if a set is convex on the graph, it is most certainly convex in the embedding space (due to high value of the projection score). However, if we build a convex hull in the embedding space, the obtained hull is unlikely to be convex on the graph (since the comparison score is rather low).



(a) Convex set (b) Zachary's Karate Club

Fig. 9: The relation between the elements of a convex set (Left). A pair of orange nodes connected by a red line induces the rest of the set. Blue nodes do not belong to the convex set. The original network is depicted on the right. Red edges correspond to the edges from the left. Black edges are the edges from the original network. The layout is determined by the Fruchterman-Reingold force-directed algorithm (spring layout).

5 Impact of convexity measure on the embedding quality

In addition, we investigated the impact of convexity measure (see Equation 2 from Section 3.1.3) on the scores. We extracted the convex skeleton of Zachary's Karate Club and ran the pipeline from Section 4.4 (the model was fixed as GCN with the same parameters). Figure 2 demonstrates such skeleton obtained by the targeted removal (see Algorithm 3) of 15% of edges. Despite the removal of a small fraction of edges, the comparison score has increased drastically: 0.710 and 0.744 with applying discretization. Other statistics are provided in Table 2.

In Zachary's Karate Club, we discovered that a large convex set can be induced by pairs of its elements in a non-trivial way. Figure 9a shows the nodes of a convex set of size 16 (which is 47% of the network) and their relations. Each pair of nodes connected by a red line induces a set. The inverse of the convex hull operator [1] in a Euclidean space is unique, thus, it can not produce such relations. On the other hand, the convex hulls of different sets can be the same in a taxicab space. Nevertheless, the structure of the inverse image is limited to a bijective graph: a set in the inverse image consists of two opposite vertices of an orthotope (which every convex set in L_1 space is).

Statistics	Original network	Convex skeleton
Degree $\langle k \rangle$	4.59	3.88
Clustering $\langle C \rangle$	0.57	0.77
Geodesics $\langle \sigma \rangle$	2.77	1.48
Convexity X	0.59	0.90
Average convex hull size	8.9	4.5
Maximal convex hull size	21	8
Comparison score	0.353	0.710
Projection score	0.971	0.983

Table 2: Statistics of Zachary's Karate Club and its convex skeleton.

[1] Here by the result of the inverse operator we denote a collection of minimal (in a sense that none of the elements can be discarded) sets which induce the hull.

It is natural to assume that the lower convexity X leads to larger convex hulls. We tested this hypothesis by evaluating the average convex hull size in the original network and its skeleton. A disconnected pair of nodes in the original network induces a set of size 8.9 on average, whereas in the skeleton it is of size 4.5. Therefore, we assume that our methods of embedding nodes might work well for networks with high convexity.

Another possible explanation for the improvement of the scores may lie in the network's community structure. According to [26], convex skeletons tend to emphasize communities. As depicted in Figure 9, the nodes of the inverse image are located in the middle of two communities. With the extraction of the convex skeleton, the communities become more isolated. It forces the inverse image to split into two parts and breaks the relation graph depicted on the left.

6 Conclusion

We propose a method for building an embedding which preserves original convexity and distance. We have tested several encoder-decoder models that were visioned to approximate one of the similarity matrices defined on networks: the adjacency and distance matrices. For this purpose, we introduced two metrics for assessing the quality of embeddings on convexity problems. Furthermore, we observed that the best results were achieved by the high-dimensional Floyd-Warshall approach with L1 distance as the decoder function. The embedding we obtained is able to preserve convexity due to the high projection score but is comparatively poor at reconstructing the convex sets within its space, since the comparison scores are lower. We observed that the model shows better results on the network's convex skeleton.

We also developed an improved algorithm for the construction of convex sets on graphs which proved to be more efficient than the baseline solution. Time complexity of our algorithm is $O(n^3)$ in the worst case, as opposed to $O(n^5)$ complexity of a naive convex hull algorithm, and works on par with an algorithm proposed in [4].

The future ways to develop the current research include devising a stable and robust algorithm for finding convex hulls in networks with low convexity. We should also note that the impact of communities on the structure of convex sets might be essential to our task and needs further investigation. In addition, the convexity problem may be generalized to searching for convex set for the purpose of existing data augmentation in metric space, useful in zero-shot learning and AutoML.

Declarations

There were no competing interests and no conflict of interests when preparing the article.

The code and datasets generated and analyzed during the current study are available in the GitHub repository

`https://github.com/realfolkcode/convexity-graph-embeddings`.

References

1. Avis, D., Bremner, D., Seidel, R.: How good are convex hull algorithms? Computational Geometry **7**(5-6), 265–301 (1997)
2. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS) **22**(4), 469–483 (1996)
3. Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
4. Ciglarič, T.: Network convexity (2017). URL `https://github.com/t4c1/Graph-Convexity`
5. Cook, W., Webster, R.: Carathéodory's theorem. Canadian Mathematical Bulletin **15**(2), 293–293 (1972)
6. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness ii: On completeness for w [1]. Theoretical Computer Science **141**(1-2), 109–131 (1995)
7. Erdős, P., Rényi, A.: Publicationes mathematicae. On random graphs I **6**, 290–297 (1959)
8. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the national academy of sciences **99**(12), 7821–7826 (2002)
9. Graham, R.L.: An efficient algorithm for determining the convex hull of a finite planar set. Info. Pro. Lett. **1**, 132–133 (1972)

10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 855–864 (2016)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017)
12. Hochstenbach, M.: A jacobi–davidson type method for the generalized singular value problem. Linear Algebra and its Applications **431**(3-4), 471–487 (2009). DOI 10.1016/j.laa.2009.03.003. URL `https://doi.org/10.1016/j.laa.2009.03.003`
13. Hornus, S., Boissonnat, J.D.: An efficient implementation of delaunay triangulations in medium dimensions. [Research Report] RR-6743, INRIA. 2008, ffinria-00343188f (2008)
14. Jarvis, R.A.: On the identification of the convex hull of a finite set of points in the plane. Information processing letters **2**(1), 18–21 (1973)
15. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953). DOI 10.1007/bf02289026. URL `https://doi.org/10.1007/bf02289026`
16. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (2014)
17. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
18. Klee, V.: Convex polytopes and linear programming. Tech. rep., BOEING SCIENTIFIC RESEARCH LABS SEATTLE WASH (1964)
19. Korovin, S.: Application of graph embeddings to network convexity problems (2019). URL `https://www.hse.ru/edu/vkr/296302923`
20. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)
21. Makarov, I., Kiselev, D., Nikitinsky, N., Subelj, L.: Survey on graph embeddings and their applications to machine learning problems on graphs. PeerJ Computer Science **7** (2021)
22. Marc, T., Šubelj, L.: Convexity in complex networks. Network Science **6**(2), 176–203 (2018)
23. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, p. 1105–1114. Association for Computing Machinery, New York, NY, USA (2016)
24. Rooney, L.: Random graph models and matchings. arXiv preprint arXiv:1909.01723 (2019)
25. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015). URL `http://networkrepository.com`
26. Šubelj, L.: Convex skeletons of complex networks. Journal of The Royal Society Interface **15**(145), 20180422 (2018)
27. Šubelj, L., Fiala, D., Ciglarič, T., Kronegger, L.: Convexity in scientific collaboration networks. Journal of Informetrics **13**(1), 10–31 (2019)

Appendix. Convexity algorithms in a Euclidean space

Gift wrapping The algorithm starts at the leftmost point p_0 of the plane and chooses the next point p_i such that all of the points are to the right of the line going through p_{i-1} and p_i. Overall complexity: $O(nh)$, where h is the number of points on the hull, therefore $O(n^2)$ in the worst case as shown in [14].

Graham scan For this algorithm, designed for the 2-D plane, we start at the lowest point, sort the points by the angle they and the starting point make with the X-axis. Then, we iteratively search for the point with a clockwise turn using a stack to eliminate concavities [9]. The time complexity is $O(n \log n)$.

Quickhull The 2-D variant of this algorithm can be described as follows: choose two points with the biggest and the least coordinate of your choice, divide the points with the line segment induced by the two base points, find the farthest from the segment and add it to the set to make a triangle. The nodes inside the triangle are eliminated. Repeat finding the farthest and eliminating the points inside until no points are left. The average complexity is $O(n \log n)$.

However, this algorithm, unlike the aforementioned ones, can be generalized to an arbitrary number of dimensions as shown in [2]. We start off by creating a simplex of $d+1$ points. Then, for each facet, we assign points to the "outside set" if it is above the facet. Following that, for each facet with the non-empty outside set, we choose the farthest point from the facet p. Next, we find the set of *horizon ridges* H_p, which is the boundary between the visible and non-visible facets from the point's perspective. Having found it, we create new facets from the point p and the ridges of H_p. Then, we calculate the outside sets for the newly obtained facets and eliminate the old facets.

The supposed complexity of the algorithm is $O(n \log r)$ for $d \leq 3$ and $O(\frac{nf_r}{r})$ for $d \geq 4$, where r is the number of processed nodes and f_r is the maximum number of facets of r vertices and $f_r = O(\frac{r^{d/2}}{(d/2)!})$ [18]. However, as presented in [1], the algorithm's triangulation complexity explodes with the increase in the number of the dimensions and causes instability in the performance.

Besides, we employ Delaunay triangulation with an overall $O(n \log n)$ time complexity for the construction when $d \leq 3$ and $O(n^{d/2})$ in the worst case otherwise [13].