IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 May 2023, accepted 27 May 2023, date of publication 30 May 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3281337

RESEARCH ARTICLE

Fast Approximate Convex Hull Construction in Networks via Node Embedding

DMITRII GAVRILEV[1] **AND ILYA MAKAROV**[2,3]
[1]Skolkovo Institute of Science and Technology, 121205 Moscow, Russia
[2]AI Center, NUST MISiS, 119991 Moscow, Russia
[3]Artificial Intelligence Research Institute (AIRI), 105064 Moscow, Russia

Corresponding author: Dmitrii Gavrilev (dmitrygavrilyev@gmail.com)

The work of Ilya Makarov was supported by the Strategic Project Digital Business within the Framework of the Strategic Academic Leadership Program "Priority 2030" with National University of Science and Technology (NUST) MISiS.

ABSTRACT Geodesic convexity in networks is an intrinsic property of graphs. It aids in distinguishing between real-world networks and random graphs. One possible application is recommending new connections in a collaborative network by searching for them in the so-called convex hull, which is a minimal subgraph containing all the shortest paths between its nodes. However, the existing algorithms for constructing convex hulls from subsets of nodes involve extensive search over subgraphs and have poor scalability. Thus, they become inapplicable to large graphs such as social networks. 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 apply the well-known convexity concept in embedding space to a similar problem for geometric learning on a graph, optimizing the process of finding all the shortest paths in the induced subgraph. To preserve the metric characteristics of a network, we train a graph neural network with an L1-distance loss. As a result, the trained model enables us to approximately verify the convexity of subgraphs in linear time, contrary to the previous approaches, which have cubic complexity.

INDEX TERMS Network convexity, graph neural networks, node embedding, shortest path, network science.

I. 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. In a metric space (X, d), a set S is convex if for any $x_1, x_2 \in S$, all elements $y \in X$ satisfying $d(x_1, y) + d(y, x_2) = d(x_1, x_2)$ belong to S [1]. Any undirected connected graph without self-loops can be seen as a metric space by setting X as the vertex set V and $d : V \times V \to \mathbb{R}$ as the shortest paths (geodesics) between the vertices. Consequently, convexity can be extended to subgraphs. Consider a connected graph G and a subgraph on a subset of nodes S. A subgraph is called convex if all the geodesics between the nodes in S are entirely included into S [2]. Further, convex hull in the context of graphs can also be defined similarly to the one in a metric space: the convex hull of $S \subseteq V$ is the minimal convex subgraph comprising S [3]. Although there are different types of graph convexity, we focus on *geodesic* convexity throughout this paper.

Unfortunately, classical graph algorithms are computationally expensive, especially when applied to real-world networks with millions of nodes and links. For example, to measure the distance between two nodes in an unweighted graph, we have to solve the problem of finding the shortest

The associate editor coordinating the review of this manuscript and approving it for publication was Seifedine Kadry.

path, which depends on the number of vertices and edges ($O(|V| + |E|)$ with applying breadth-first search [4]). 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, and hence cannot be applied to 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 the geodesic paths between its nodes; therefore, it is convex. It is also minimal, meaning that the exclusion of node 4 would destroy its convexity. The same problem could be approached via node embedding. 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 the node embedding indeed preserves convexity, then the problem of searching for the convex hull is reduced to constructing one in a metric space.

The aim of the paper is to research and devise a method for fast approximate 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 that have lower computational costs. Since convexity is a property inherent to real networks, the research may prove useful in various applications of network classification as well as in distinguishing the networks from random graphs [5]. Another direction for research involves community detection in networks based on their convex skeleton [6], which can be viewed as a generalization of a spanning tree. The research results might also be applied to the problem of finding the optimal subgraph of roads in transportation networks [5], [6], active and online node classification [7], [8].

We reduce the initial problem of finding convex subgraphs to finding convex hulls in a node embedding space [9]. We leverage the properties of the L1 embedding space to decrease the computational complexity. To that end, we use the framework of Graph Neural Networks (GNN) to find the optimal embeddings which preserve the geodesics in a graph, similar to link prediction [10], [11] and link regression [12], [13] tasks but in more constrained geometric setting. We design an efficient loss function suited for mini-batched learning. Our proposed method is linear w.r.t the number of nodes, contrary to the existing graph-based algorithms which have the cubic complexity in the worst-case scenario. The exact complexity of querying convex hulls in the L1 metric space is $O(|V| \cdot d)$, where d is the embedding dimension.

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

II. RELATED WORK

The earliest instance of convex hull algorithm for graphs has the time complexity of $O(|\operatorname{conv}(S)| \cdot |E|)$, where $\operatorname{conv}(S)$ is the convex hull of set $S \subseteq V$ [14]. An algorithm that does not depend on the number of edges was described in [7], and has the complexity of $O(|V|^3)$. A heuristic algorithm proposed in [15] approximates the dense core of a graph by first spanning p random outerplanar [16] subgraphs. Then, it computes a convex hull in each of the p subgraphs in $O(|V| \cdot f)$, where f is the number of interior faces in the outerplanar graph. Finally, a node from the original graph is considered to be in the convex hull iff it frequently appears in the outerplanar convex hulls. A shortcoming of this approach is the lack of control over the expected runtime as f implicitly depends on the graph. In the worst case, $f = O(|V|)$ [15], which makes the overall complexity quadratic. On the other hand, the runtime of our proposed method can be controlled in advance.

Convexity on graphs was studied in three distinct forms: a global one, which refers to a tree of cliques and grows at a noticeably low pace; a local one, which is more typical for random graphs than for real networks; and a regional one, which refers to any partly convex heterogeneous network [5]. Furthermore, the authors show that convexity is a structural property inherent to real-world networks. According to their research, random graph models like Erdos-Renyi, Barabasi-Albert, and Watts-Strogatz fail to recreate convexity in networks (see [17] for random graphs description).

Moreover, convexity in networks can be measured by observing the growth of convex subgraphs. The procedure starts with initializing subset S with a random node. On each next step, a random node out of the neighboring nodes of S is added to the subset. Then, S is expanded by the nodes of its convex hull. Thus, on each step, the induced subgraph on S is its convex hull. This process of expansion is repeated until S covers the entire network. Finally, network convexity X is measured as

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

where n is the number of nodes in a network and $\Delta s(t)$ is the average increase in the size of S at step t. Due to the stochasticity of the procedure, the expansion should be run several times. Hence, $\Delta s(t)$ is the average over the

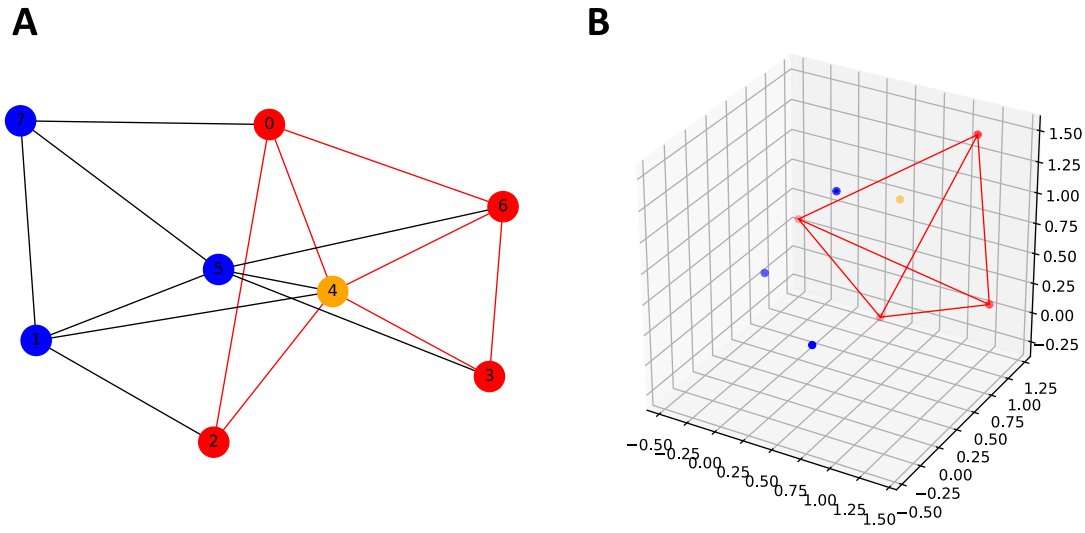

FIGURE 1. An example of a graph (A) and its 3-dimensional embedding (B). The base nodes of the convex hull are colored red. An orange node denotes another node that belongs to the convex hull. The edges that are induced by the convex hull are colored red. The induced subgraph is convex since it contains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

independent runs of the procedure. Convexity $X \in [0, 1]$, with the higher values corresponding to globally convex networks such as trees of cliques. The lower value of X indicates that there is a dramatic expansion of the convex hulls in a network.

In [9] and [18], the authors present a survey on the graph embedding algorithms and methods. The overview includes 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.). 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 the 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's embedding is assigned as a combination of the aggregated neighborhood and an embedding corresponding to the previous layer.Â

For instance, a variation of GNNs, GraphSAGE [19], uses an aggregation rule in the form of

$$z_v^{(l)} = \sigma\left(\left[\mathbf{W}^{(l)} \cdot \mathrm{AGG}\left(\left\{z_u^{(l-1)}, \forall u \in N(v)\right\}\right), \mathbf{B}^{(l)} z_v^{(l-1)}\right]\right), \tag{2}$$

where z_v^l is the embedding for node v, $\mathbf{W}^{(l)}$ and $\mathbf{B}^{(l)}$ are trainable matrices in the l-th layer. The concatenation operator $[\cdot, \cdot]$ is applied to the weighted aggregation of the neighbors and previous representation of v, which is then followed by a (non-linear) activation function σ. The aggregation function AGG can be viewed as a convolutional kernel and may differ from model to model. In particular, [19] propose to use permutation invariant aggregators such as mean, LSTM, and max-pooling. Stacking convolutional layers allows information to propagate not only from the local neighborhood but also from the distant neighbor nodes. In addition, methods for joint node and edge embedding allow to train semi-supervised [20], [21] and self-supervised [22] models for certain sparse graphs.

III. METHODS

In the following section, we discuss methods for convex set creation in graphs and vector spaces. We review the overall approach to the construction and cover the computational complexity of the algorithms. First, we present traditional convex hull algorithms in graphs. Next, we examine convex set methods in a taxicab space induced by the $L1$ norm, and introduce a relaxation for the convex hull operator. Finally, we present the training procedure for learning the $L1$ embeddings.

A. EXACT CONVEX HULL CONSTRUCTION

To construct $H = \mathrm{conv}(S)$, [14] first initialize it with $H = S$. Next, they run bread-first search from each node of H, and include all the new nodes along the geodesics if they cross H. The resulting time complexity is $O(|\,\mathrm{conv}(S)| \cdot |E|)$.

Alternatively, as presented in [7] (see Algorithm 1), we can precompute the distance matrix and search the triplets for which the triangle inequality holds with equality. In the case of unweighted networks, the distance matrix can be constructed by running breadth-first search from each node.

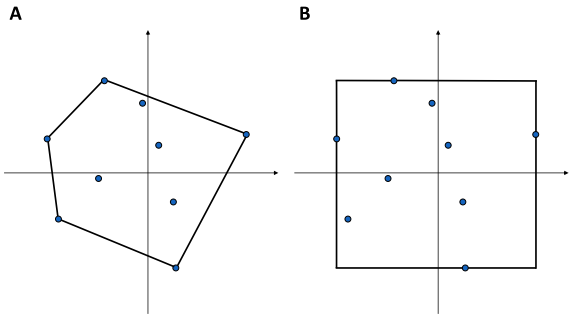

FIGURE 2. The difference between convex hulls in a Euclidean space (A) and a taxicab space (B).

Despite the $O(|V| \cdot (|V| + |E|))$ complexity, we only have to calculate it once.

Algorithm 1 Exact Convex Hull Construction

Data: subset S
Result: the vertex set of the convex hull H
$H \leftarrow S$ **while** *new node is added to* H **do**
  for *all pairs* a, b *in* H **do**
    for *all nodes* c *except nodes in* H **do**
      if $dist(a, b) > 1$ *and* $dist(a, c) + dist(c, b) = dist(a, b)$ **then**
        $H \leftarrow H \cup \{c\}$
      end
    end
  end
end

Both exact algorithms are asymptotically equivalent in the sense that they are $O(|V|^3)$ in the worst-case scenario.

B. CONVEX SETS IN A TAXICAB SPACE

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

$$\|z\|_1 = \sum_{k=1}^{d} |z^k|. \tag{3}$$

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

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_1 = \sum_{k=1}^{d} |x^k - y^k|. \tag{4}$$

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

$$\begin{aligned}\mathrm{conv}(S) = \{&\min \pi_1(S) \le x^1 \le \max \pi_1(S), \dots, \\ &\min \pi_d(S) \le x^d \le \max \pi_d(S)\}.\end{aligned} \tag{5}$$

To construct the convex hull in a taxicab space, it is sufficient to find the coordinate-wise minima and maxima of a given set. Hence, the computational complexity is $O(n \cdot d)$, where n is the total number of points. 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, whose complexities explode with the increase in the number of dimensions [23].

Another advantage of a taxicab space lies in the fact that there are multiple geodesics between two points, whereas the geodesics are always unique in a space with the L_2 distance. Moreover, we are not limited by the dimension of a 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 a set S has positive volume unless S is affinely dependent. Figure 2 illustrates the difference between convex hulls in Euclidean and taxicab spaces. Note that a convex taxicab set tends to encompass more space.

In practice, the nodes may not be perfectly embedded into a taxicab space, and they are likely to be distorted, either theoretically or numerically. To address this issue, we introduce a simple yet effective extension to our algorithm. Instead of directly operating with $\mathrm{conv}(S)$, we approximate the convex hull in a graph with $\mathrm{conv}_\epsilon(S)$, a weakened version of $\mathrm{conv}(S)$:

$$\begin{aligned}\mathrm{conv}_\epsilon(S) = \{&\min \pi_1(S) - \epsilon \le x^1 \le \max \pi_1(S) + \epsilon, \dots, \\ &\min \pi_d(S) - \epsilon \le x^d \le \max \pi_d(S) + \epsilon\},\end{aligned} \tag{6}$$

where ϵ is the thresholding hyperparameter. Note that at $\epsilon = 0$, $\mathrm{conv}_\epsilon(S)$ is reduced to $\mathrm{conv}(S)$.

C. LEARNING THE DISTANCES

We consider the problem of mapping a graph to a taxicab space as the problem of learning the distance matrix. As in Algorithm 1, we precompute the distance matrix, which we further leverage in our target objective. We want pairwise distances in a taxicab space to approximate the original distances in a graph. Let $\mathbf{D}$ and enc denote the distance matrix and learnable node encoder, respectively. Then, the objective is given in the form of

$$L = \sum_{\substack{u,v \in V \\ u \neq v}} \left(\|\,\mathrm{enc}(u) - \mathrm{enc}(v)\|_1 - D_{u,v} \right)^2. \tag{7}$$

We choose enc as a GNN to leverage the graph structure. However, in practice, we can not afford the full-batch learning on large graphs. Therefore, we propose the following mini-batching scheme to reduce the memory consumption. First, we compute the node embeddings using randomly initialized enc and denote them $\hat{z}$. The embeddings $\hat{z}$ are then freezed, meaning that we detach them from the computational graph. Next, we sample a mini-batch of target nodes B. Then, the loss is computed as

$$L = \sum_{\substack{u \in B, v \in V \\ u \neq v}} \left(\|\,\mathrm{enc}(u) - \hat{z}_v\|_1 - D_{u,v} \right)^2, \tag{8}$$

and one step of gradient descent is performed. Finally, we update the frozen embeddings of the nodes from B:

$$\hat{z}_u \leftarrow \text{enc}(u), \quad \forall u \in B. \tag{9}$$

The process is then repeated, starting with sampling a new mini-batch. Note that we only need to update the embeddings for a small subset of nodes, so we employ GraphSAGE as our encoder to reduce the overall computational graph needed to retrieve the mini-batch representation [19].

Alternatively, we propose to substitute the distances in Equation 8 with their logarithmic values:

$$L = \sum_{\substack{u \in B, v \in V \\ u \neq v}} \left(\ln \| \text{enc}(u) - \hat{z}_v \|_1 - \ln D_{u,v} \right)^2. \tag{10}$$

IV. EXPERIMENTS

In this section, we describe the experiments on which we test our methods. First, we provide a full list of the datasets and their statistics, as well as a description of preprocessing. Next, we introduce the metrics that help us assess the quality of the embeddings. Finally, we describe the pipeline of our experiments and discuss the main findings.

A. DATASETS

In what follows, we describe the datasets on which we conduct our experiments and their statistics. The following networks are used: the Little Rock Lake food web [24], coauthorships in network science [25], European road network [26], connections between the US political blogs [27], flights between the US airports [28] and the Western US power grid [29]. The datasets are downloaded from KONECT [28], the website of the project oriented on collecting network data, and Mark Newman's personal page, which contains a compilation of networks [30]. The largest graph we use is Arxiv ASTRO-PH [31], [32] (17903 nodes), a scientific collaboration network from arXiv.

We transform each of the networks mentioned above into a simple graph. In other words, we remove all multiple edges, self-loops, and edge direction. Besides, if a graph is unconnected, we consider only its largest connected component. In Table 1, we present the following statistics for the preprocessed graphs: the number of nodes $|V|$, number of edges $|E|$, and convexity X, which is introduced in Eq. 1. For measuring X, we use the original implementation of the expansion algorithm from [5] and report the average values over 100 repeats.

B. METRICS

We propose two ways of measuring embedding accuracy. One compares two independently built convex sets, the other assesses the similarity of the convex hull of the convex set built on the graph after projecting it onto the vector space. The former measures the ability of the embeddings to recreate convexity, whereas the latter estimates how well they preserve convexity.

1) 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:

$$J(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}. \tag{11}$$

2) PROJECTION

For this method, in contrast to the previous one, we first build only one convex set on the graph. The vector representations of the corresponding nodes are gathered from the embedding matrix once the ground-truth convex set has been obtained. Following that, we build the convex hull S of those points, search for the additional points that lie inside this hull, and call them the *error set*. Then we measure the accuracy of the set's recreation with the following formula:

$$\text{precision}(S, error) = \frac{|S| - |error|}{|S|}, \tag{12}$$

which essentially evaluates the fraction of the correct nodes.

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

C. RESULTS

In Table 1, we present the training settings for the datasets: batch size, final layer dimension, and number of epochs. Before training, we normalize the distance matrix D such that it has a unit standard deviation. In all cases, we train the 3-layer GraphSAGE with a mean aggregator. Each subsequent layer has its embedding dimension get reduced by half. The non-linearity σ is chosen as Swish [33]. The initial embedding is set to the identity matrix of size $|V|$. We use PyTorch [34] and DGL (Deep Graph Library [35]) in our implementation. The optimization is done using Adam optimizer [36]. Additionally, while learning we employ the gradient clipping. We share our code at https://github.com/realfolkcode/fast-network-convexity.

What follows is a description of the pipeline on which our approach is 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 that belong to the hull and calculate the metric. As for the projection method, we build the convex hull in the embedding space from the convex hull obtained on the graph. Next, we find the error set and calculate the metric. Then, for each method,

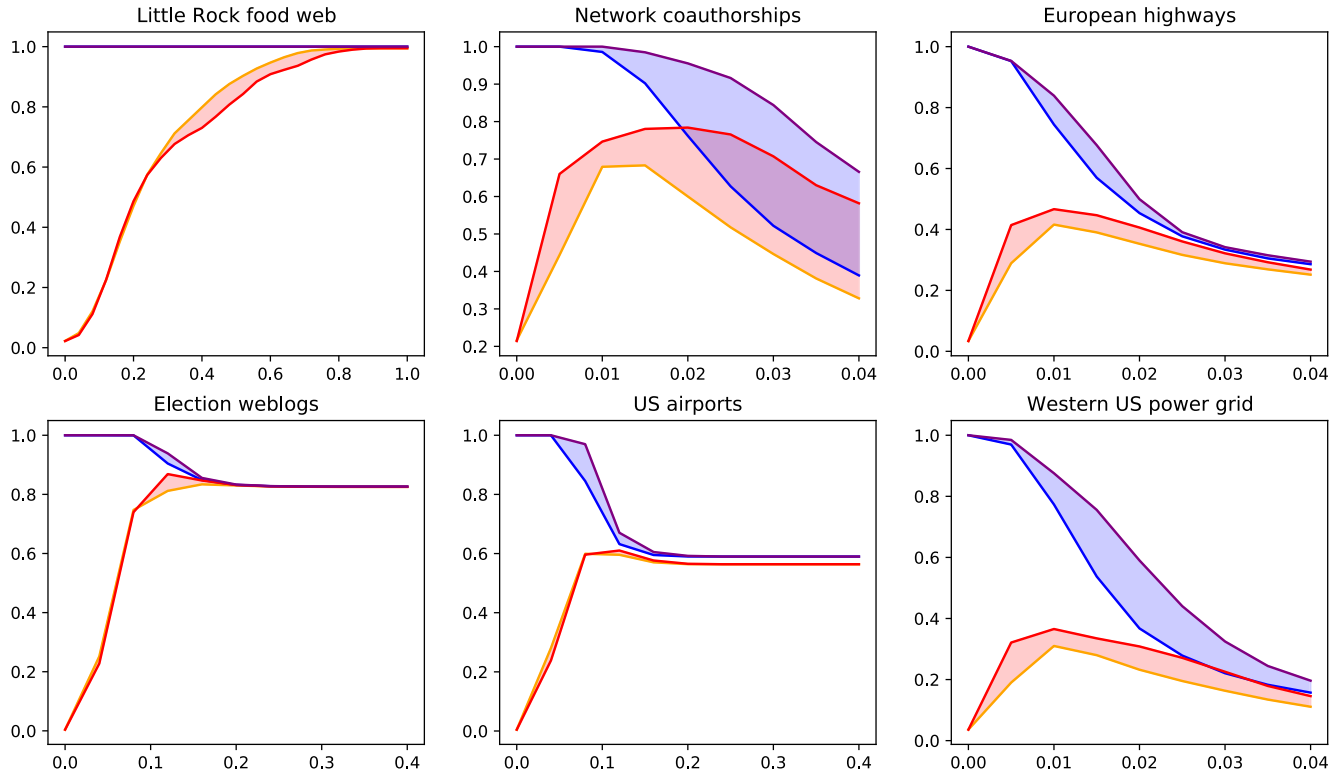

FIGURE 3. Scores versus (y-axis) ϵ-threshold (x-axis). The orange and blue lines present the comparison and projection scores respectively, for the models trained with the standard loss. The results for the logarithmic loss are depicted in red (comparison) and purple (projection).

TABLE 1. Graph statistics and training settings.

Dataset	$\|V\|$	$\|E\|$	X	Batch size	Final layer dim.	Num. of epochs
Little Rock food web	183	2452	0.02	64	32	2000
Network coauthorships	379	914	0.85	128	64	2000
European highways	1039	1305	0.45	256	128	1500
Election weblogs	1222	16717	0.17	256	128	1500
US airports	1572	17214	0.42	512	128	2000
Western US power grid	4941	6594	0.52	256	128	300
Arxiv Astro Physics	17903	197031	0.46	128	128	100

TABLE 2. Metrics and optimal threshold.

Dataset	Loss	Threshold	Comparison	Projection	Loss (log)	Threshold (log)	Comparison (log)	Projection (log)
Little Rock food web	0.1792	1.000	0.99	1.00	0.0289	1.000	0.99	1.00
Network coauthorships	0.0134	0.015	0.65	0.92	0.0008	0.020	0.80	0.98
European highways	0.0085	0.010	0.42	0.74	0.0026	0.010	0.47	0.84
Election weblogs	0.1377	0.160	0.84	0.85	0.0169	0.120	0.87	0.94
US airports	0.0785	0.080	0.60	0.87	0.0108	0.120	0.61	0.67
Western US power grid	0.0111	0.010	0.31	0.77	0.0013	0.010	0.37	0.88
Arxiv Astro Physics	0.1387	0.100	0.58	0.62	0.0106	0.100	0.58	0.62

we evaluate the average score over all samples. Our method is substantially faster than the standard algorithm described in Section III-A: on the Arxiv Astro Physics graph, the construction of 50 random convex hulls takes 10 seconds with the former, whereas it takes more than 20 minutes with the latter.

Figure 3 shows our main results, as it illustrates the evolution of the comparison and projection scores with the increase of the ϵ-threshold introduced in Eq. 6. For each dataset, at $\epsilon = 0$, the comparison is close to zero. This validates our choice of the ϵ-thresholding, as conv$(\cdot)$ completely fails to recreate convexity in networks. With the increase of ϵ, the comparison score increases until it reaches its optimal value, after which it starts decreasing. On the other hand, the projection score starts from approximately 1, then it monotonously decays. This indicates that our embeddings are able to preserve convexity. The monotonous decay is explained by the overall increase of $\epsilon-$convex hulls that leads to inevitable inclusion of additional nodes. Therefore, there is a tradeoff between recreating and preserving convexity. In Table 2, we present

the values of ϵ that maximize the comparison score and the metrics. Additionally, we report the corresponding loss values. Furthermore, we investigate the effect of the logarithmic distances that we introduce in Equation 10. As can be seen from Figure 3 and Table 2, in each network except the Little Rock food web, substituting the distances with their logarithmic values boosts the overall quality.

We empirically observe that the optimal value of ϵ correlates with the loss values and convexity measure of a network. The model fails to isometrically embed the networks with low convexity (Little Rock food web and Election weblogs), which results in a higher error in the reconstructed distances. Compared to the networks with high convexity, these networks benefit from a much higher ϵ. As a simple heuristic, the corresponding orders of magnitude would hint towards the choice of the threshold. Despite the high error in the distances, $\text{conv}_\epsilon(\cdot)$ leverages the low convexity by including more nodes. In particular, we notice that in the Little Rock food web, the convex hulls of random subsets of size 4 cover almost the entire graph (182 nodes).

V. CONCLUSION

To our knowledge, this paper is the first attempt at speeding up the construction of convex hulls in graphs by employing a deep learning approach. Compared to the standard algorithms on graphs, our method leverages the properties of a taxicab space and reduces the computational complexity of the construction from $O(|V|^3)$ to $O(|V| \cdot d)$ after training and obtaining the node embeddings. Moreover, we introduce the thresholding technique that helps us alleviate the errors in distances. To that end, our approach might be useful in scenarios where the number of queries is large.

Future ways to develop the current research include investigating the connection between the convexity measure and threshold, deriving a theoretical estimate of the optimal threshold, and incorporating other objective functions, such as the Laplacian embedding loss [37]. Besides, a more detailed investigation of vector spaces equipped with the ℓ_p norm can be fruitful. A handy property of ℓ_1 norm is the non-uniqueness of geodesics. The only other norm with such a property is the ℓ_∞ norm [38]. Another possible direction of research is to develop an algorithm for the fast construction of convex skeletons, a lightweight abstraction of networks [6].

ACKNOWLEDGMENT

The authors are grateful to Dr. Lovro Šubelj for fruitful discussions and original problem formulation. Author Contributions: Dmitrii Gavrilev: paper preparation, conducting experiments, implementation and design of methods; and Ilya Makarov: paper revision, help with experiment and model design, research supervision.

REFERENCES

[1] I. M. Pelayo, *Geodesic Convexity in Graphs*. Cham, Switzerland: Springer, 2013.
[2] H.-J. Bandelt and V. Chepoi, "Metric graph theory and geometry: A survey," *Contemp. Math.*, vol. 453, pp. 49–86, Jan. 2008.
[3] F. Harary and J. Nieminen, "Convexity in graphs," *J. Differ. Geometry*, vol. 16, no. 2, pp. 185–190, Jan. 1981.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2022.
[5] T. Marc and L. Šubelj, "Convexity in complex networks," *Netw. Sci.*, vol. 6, no. 2, pp. 176–203, Jun. 2018.
[6] L. Šubelj, "Convex skeletons of complex networks," *J. Roy. Soc. Interface*, vol. 15, no. 145, Aug. 2018, Art. no. 20180422.
[7] M. Thiessen and T. Gärtner, "Active learning of convex halfspaces on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 23413–23425, 2021.
[8] M. Thiessen and T. Gürtner, "Online learning of convex sets on graphs," in *Proc. Mach. Learn. Knowl. Discovery Databases, Eur. Conf.* Grenoble, France: Springer, Sep. 2023, pp. 349–364.
[9] I. Makarov, D. Kiselev, N. Nikitinsky, and L. Subelj, "Survey on graph embeddings and their applications to machine learning problems on graphs," *PeerJ Comput. Sci.*, vol. 7, p. e357, Feb. 2021.
[10] I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Recommending co-authorship via network embeddings and feature engineering: The case of national research university higher school of economics," in *Proc. 18th ACM/IEEE Joint Conf. Digit. Libraries* New York, NY, USA: ACM, May 2018, pp. 365–366.
[11] I. Makarov, M. Makarov, and D. Kiselev, "Fusion of text and graph information for machine learning problems on networks," *PeerJ Comput. Sci.*, vol. 7, p. e526, May 2021.
[12] I. Makarov and O. Gerasimova, "Predicting collaborations in co-authorship network," in *Proc. 14th Int. Workshop Semantic Social Media Adaptation Personalization (SMAP)*, New York, NY, USA, Jun. 2019, pp. 1–6.
[13] I. Makarov and O. Gerasimova, "Link prediction regression for weighted co-authorship networks," in *Proc. 15th Int. Work-Conf. Artif. Neural Netw. (IWANN)*. Berlin, Germany: Springer, Universitat Politecnica de Catalunya, Jul. 2019, pp. 667–677.
[14] M. C. Dourado, J. G. Gimbel, J. Kratochvíl, F. Protti, and J. L. Szwarcfiter, "On the computation of the hull number of a graph," *Discrete Math.*, vol. 309, no. 18, pp. 5668–5674, Sep. 2009.
[15] F. Seiffarth, T. Horváth, and S. Wrobel, "A fast heuristic for computing geodesic cores in large networks," 2022, *arXiv:2206.07350*.
[16] G. Chartrand and F. Harary, "Planar permutation graphs," *Annales de l'IHP Probabilités et Statistiques*, vol. 3, no. 4, 1967, pp. 433–438.
[17] L. Rooney, "Random graph models and matchings," 2019, *arXiv:1909.01723*.
[18] W. L. Hamilton, "Graph representation learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 14, no. 3, pp. 1–159, 2020.
[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1025–1035.
[20] I. Makarov, O. Gerasimova, P. Sulimov, K. Korovina, and L. E. Zhukov, "Joint node-edge network embedding for link prediction," in *Proc. 7th Int. Conf. Anal. Images, Social Netw. Texts (AIST)*. Berlin, Germany: Springer, Polytechnic University, Jul. 2018, pp. 20–31.
[21] I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Co-authorship network embedding and recommending collaborators via network embedding," in *Proc. 7th Int. Conf. Anal. Images, Social Netw. Texts (AIST)*. Berlin, Germany: Springer, Polytechnic University, Jul. 2018, pp. 32–38.
[22] I. Makarov, K. Korovina, and D. Kiselev, "JONNEE: Joint network nodes and edges embedding," *IEEE Access*, vol. 9, pp. 144646–144659, 2021.
[23] D. Avis, D. Bremner, and R. Seidel, "How good are convex hull algorithms?" *Comput. Geometry*, vol. 7, nos. 5–6, pp. 265–301, Apr. 1997.
[24] R. J. Williams and N. D. Martinez, "Simple rules yield complex food webs," *Nature*, vol. 404, no. 6774, pp. 180–183, Mar. 2000.
[25] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 74, no. 3, Sep. 2006, Art. no. 036104, doi: 10.1103/PHYSREVE.74.036104.
[26] L. Šubelj and M. Bajec, "Robust network community detection using balanced propagation," *Eur. Phys. J. B*, vol. 81, no. 3, pp. 353–362, Jun. 2011.

[27] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 US election: Divided they blog," in *Proc. 3rd Int. Workshop Link Discovery*, Aug. 2005, pp. 36–43.
[28] J. Kunegis, "KONECT: The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web*, May 2013, pp. 1343–1350.
[29] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[30] M. Newman. (2022). *Mark Newman's Compilation of Networks*. [Online]. Available: http://www-personal.umich.edu/~mejn/netdata/
[31] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discovery Data*, vol. 1, no. 1, p. 2, Mar. 2007.
[32] J. Leskovec and R. Sosič, "SNAP: A general-purpose network analysis and graph-mining library," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 1, pp. 1–20, Jan. 2017.
[33] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.
[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8026–8037.
[35] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *Proc. ICLR Workshop Represent. Learn. Graphs Manifolds*, 2019, pp. 1–7.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[37] Z. Wu, X. Lin, Z. Lin, Z. Chen, Y. Bai, and S. Wang, "Interpretable graph convolutional network for multi-view semi-supervised learning," *IEEE Trans. Multimedia*, early access, Mar. 23, 2023, doi: 10.1109/TMM.2023.3260649.
[38] M. R. Bridson and A. Haefliger, *Metric Spaces of Non-Positive Curvature*, vol. 319. Berlin, Germany: Springer, 2013.

DMITRII GAVRILEV received the bachelor's degree in applied mathematics from HSE University, Moscow, Russia. He is currently pursuing the master's degree with Skoltech, Moscow.

In 2022, he was an Intern with the Huawei Moscow Research Center. He is also an Intern Researcher with Skoltech. His research interests include graph ML and generative modeling.

ILYA MAKAROV received the Specialist degree in mathematics from Lomonosov Moscow State University, Moscow, Russia, and the Ph.D. degree in computer science from the University of Ljubljana, Ljubljana, Slovenia.

Since 2011, he has been a Lecturer with the School of Data Analysis and Artificial Intelligence, HSE University, where he was the School Deputy Head, from 2012 to 2016, and he is currently an Associate Professor and a Senior Research Fellow. He was also the Program Director of the Bigdata Academy MADE, VK, and a Researcher with the Samsung-PDMI Joint AI Center, St. Petersburg Department, V.A. Steklov Mathematical Institute, Russian Academy of Sciences, Saint Petersburg, Russia. He is also a Senior Research Fellow with the Artificial Intelligence Research Institute (AIRI), Moscow, where he leads the research in industrial AI. He became the Head of the AI Center and Data Science Tech Master Program in NLP, National University of Science and Technology MISIS.