Cleora: A Simple, Strong and Scalable Graph Embedding Scheme

Barbara Rychalska[12], Piotr Bąbel[1], Konrad Gołuchowski[1], Andrzej Michałowski[1], and Jacek Dąbrowski[1]

[1]Synerise
[2]Warsaw University of Technology

***Abstract*—The area of graph embeddings is currently dominated by contrastive learning methods, which demand formulation of an explicit objective function and sampling of positive and negative examples. This creates a conceptual and computational overhead. Simple, classic unsupervised approaches like Multidimensional Scaling (MSD) or the Laplacian eigenmap skip the necessity of tedious objective optimization, directly exploiting data geometry. Unfortunately, their reliance on very costly operations such as matrix eigendecomposition make them unable to scale to large graphs that are common in today's digital world. In this paper we present Cleora: an algorithm which gets the best of two worlds, being both unsupervised and highly scalable. We show that high quality embeddings can be produced without the popular step-wise learning framework with example sampling. An intuitive learning objective of our algorithm is that a node should be similar to its neighbors, without explicitly pushing disconnected nodes apart. The objective is achieved by iterative weighted averaging of node neigbors' embeddings, followed by normalization across dimensions. Thanks to the averaging operation the algorithm makes rapid strides across the embedding space and usually reaches optimal embeddings in just a few iterations. Cleora runs faster than other state-of-the-art CPU algorithms and produces embeddings of competitive quality as measured on downstream tasks: link prediction and node classification. We show that Cleora learns a data abstraction that is similar to contrastive methods, yet at much lower computational cost. We open-source Cleora under the MIT license allowing commercial use under https://github.com/Synerise/cleora.**

I. Introduction

Graphs are data structures which are extremely useful for modeling real-life interaction structures. A graph is represented by sets of nodes and edges, where each node represents an entity from the graph domain, and each edge represents the relationship between two or more nodes. Graph structures are found for example in biology as interconnected sets of amino acids building proteins [17], [30], [45], [46], road networks [43], [48], as well as social networks [4], [47], citation networks [5], [34], or web data [11], [35]. In most machine learning applications it is crucial to represent graph nodes as node embeddings - structures expressing node properties as an input to downstream machine learning algorithms. A simple node adjacency matrix is usually not feasible due to its large size (quadratic with respect to the number of nodes) and lack of easily accessible representation of node properties. A number of elaborate node embedding methods have been proposed, with configurable embedding dimensionality and node similarity defined in terms of a selected distance metric. However, most of these approaches do not scale to the real-world large graphs consisting of millions of nodes and billions of edges. Methods such as Deepwalk [32] or Node2Vec [15], require hours or even days of training even on medium-sized graphs to arrive at a representation of reasonable quality. Scalable models have also been proposed, such as PyTorch BigGraph (PBG) [21] or GOSH [3], however such optimization methods demand significant model complexity to parallelize the computations or/and coarsen the graph into a smaller structure.

In this paper we present Cleora: a very simple yet competitive graph embedding scheme. Cleora relies on multiple iterations of normalized, weighted averaging of each node's neighbor embeddings. Each embedding dimension is optimized independently from other dimensions (with the exception of normalization, which takes into account all dimensions). As such, Cleora can be thought of as an ensemble of 1-D optimizers rather than a single optimizer of N-D data. The operation of averaging can make rapid, substantial changes to embeddings (as compared to step-wise optimization based on an objective), making the algorithm reach optimal embeddings in just four to five iterations on average. Cleora is purely unsupervised. No explicit learning objective is formulated and no sampling of positive or negative examples is performed.

Cleora has only two configurable parameters: the number of iterations and dimensionality of resulting embeddings, which makes it easy to search for an optimal configuration. Contrastive learning methods will usually have many more configurable parameters, for example PBG has at least 18 parameters which directly define the training process. Moreover, some methods (PBG included) define a number of loss functions which optimize embeddings for various purposes separately (e.g. node ranking or classification). With Cleora the obtained embeddings are versatile as no explicit objective is defined.

Cleora scales well to massive graph sizes and can embed various types of edges: undirected, directed and/or weighted. We evaluate Cleora on various real-world datasets of medium to large size, in order to show that the quality of produced embeddings is competitive to recent state-of-the-art methods, without the overhead of complex architecture. We show that Cleora is faster than other recent CPU-based methods, thanks to the simplicity and the ease of parallelization.

Moreover, our algorithm possesses two interesting properties which are a consequence of the weighted averaging approach and may turn up useful in practical scenarios:

- **Additivity.** The embedding procedure can be conducted on chunked graph and the embedded chunks can be merged with a single Cleora step, without any extra solution. As the method is based on simple weighted averaging, an average of all chunks' embeddings produces the final embeddings. This simply repeats the operation which would have happened on the full graph if embedded directly. Cleora can embed massive graphs which do not fit into RAM/disk space at once.

- **Inductivity.** Embedding of nodes added after the computation of full graph embedding is a natural operation, requiring a single iteration of the algorithm on just the newly added nodes. None of the competitors we evaluate have this ability.

Cleora is an algorithm which processes data from large retailers in our production environment, with graphs comprised of millions of nodes and billions of edges. Embedding times for our biggest e-commerce datasets are below 2 hours on a single Azure virtual machine of type Standard E32s v3 32 vCPUs/16 cores and 256 GB RAM. Cleora has already been in use for 1 year.

We release Cleora as open-source software. We offer an easy to run, highly optimized implementation written in Rust. We release the code under the permissive MIT license, which allows commercial use[1].

II. Related Work

The first attempts at graph embeddings were purely unsupervised. Methods such as classic PCA [18], Laplacian eigenmap [7], MSD [12], and IsoMap [40] typically exploit spectral properties of various matrix representations of graphs, such as the Laplacian and the adjacency matrix. In essence, these methods preform dimensionality reduction while preserving distance relations between nodes (viewed as path distances in graphs). Unfortunately, their complexity is at least quadratic to the number of nodes, which makes these methods far too slow for today's volumes of graph data.

A subsequent, essential line of work was based on random walks between graph nodes. The classic DeepWalk model [32] uses random walks to create a set of node sequences fed to a skip-gram model, taking an inspiration from learning representations of tokens in Word2Vec [28] from the area of natural language processing. Node2vec [15] is a variation of DeepWalk where the level of random walk exploration (depth-first search) versus exploitation (breadth-first search) is controlled with parameters. Other more recent approaches make the random walk strategy more flexible at the cost of increased complexity [10], [33]. These models perform well in practice, but cannot scale to graphs with millions of nodes.

LINE [39] is yet another approach, which aims to preserve the first-order or second-order proximity separately by the use of KL-divergence minimization. After optimizing the loss functions, it concatenates both representations. LINE is optimized to handle large graphs, but its GPU-based implementation - Graphvite [49] - cannot embed a graph when the total size of the embedding is larger than the total available GPU memory. With GPU memory size usually much smaller than available RAM, this is a serious limitation.

Another established line of work focuses on embedding graphs with the use of graph convolutional neural networks [19]. Methods such as GraphSAGE [16] are designed for cases where the graph nodes are accompanied by additional feature information and incorporation of no-feature nodes might be complicated or demand additional measures. In contrast, our approach focuses on pure graph structure.

Works closest to our aims in terms of scalability and speed are able to embed massive graphs in reasonable time without the requirement of node features or any kind of side information. These fast embedding methods often boil down to older methods implemented in parallelizable and highly efficient architectures. Pytorch BigGraph (PBG) [21] uses ideas such as adjacency matrix partitioning and reusing examples within one batch to train models analogous to successful but less scalable RESCAL [31], DistMult [44], TransE [8],and ComplEx [41], thus making these models applicable to large graphs. GOSH [3] is a GPU-based approach which trains a parallelized version of Verse [42] on a number of smaller, coarsened graphs. Embeddings are computed with Noise Contrastive Estimation between positive and negative samples, first on smaller, coarsened graphs and projected to bigger ones. In contrast to these methods, Cleora is not a reimplementation of an older approach but a conceptually new algorithm. Moreover, the listed methods are complex, demand step-wise training via gradient descent, and do not naturally generalize to unseen data.

III. Cleora Embeddings

A. Preliminaries

A graph G is a pair (V, E) where V denotes a set of nodes (also called vertices) and $E \subseteq (V \times V)$ as the set of edges connecting the nodes. We consider undirected graphs, where an edge is an unordered pair of nodes. An embedding of a graph $G = (V, E)$ is a $|V| \times d$ matrix T, where d is the dimension of the embedding. The i-th row of matrix T (denoted as $T_{i,*}$) corresponds to a node $i \in V$ and each value $j \in \{1, ..., d\}$ in the vector $T_{i,j}$ captures a different feature of node i.

Hypergraphs are a generalization of graphs where the relationships between nodes can be not only pairwise, but also higher-order. For example, the relation of Customer c buying a Product p in Store s is a hyperedge composed of nodes c, p, s. Hyperedge width k is defined as the number of nodes contained in the hyperedge. For example, the hyperedge (c, p, s) has width 3.

B. The Algorithm

Hypergraph Expansion. In order to produce hypergraph embedding, Cleora needs to break down all existing hyperedges into edges as the algorithm relies on the pairwise notion of node transition. Hypergraph expansion to graph is done using two alternative strategies:

- *Clique Expansion.* Each hyperedge is transformed into a clique - a subgraph where each pair of nodes is

[1]https://github.com/Synerise/cleora

connected with an edge. Space/time complexity of the whole embedding procedure in this approach is given by $O(|V| \times d + |E| \times k^2)$ where $|E|$ is the number of hyperedges and k is the maximal width of hyperedge from the hypergraph. With the usage of cliques the number of created edges can be significant but guarantees better fidelity to the original hyperedge relationship. We apply this scheme to smaller graphs.

- *Star Expansion.* An extra node is introduced which links to the original nodes contained by a hyperedge. Space/time complexity of the whole embedding procedure in this approach is given by $O((|V|+|E|)\times d+|E|k)$. Here we must include the time and space needed to embed an extra entity for each hyperedge, but we save on the number of created edges, which would be only k for each hyperedge. This approach is recommended for large graphs or graphs with large hyperedges.

Embedding. With all hyperedges broken down into pairwise edges, we proceed to embed the graph. Given an interaction network (e.g. between users and items) represented as a graph $G = (V, E)$, we define the random walk transition matrix $\mathbf{M} \in \mathbb{R}^{V \times V}$ where $\mathbf{M}_{ab} = \frac{e_{ab}}{deg(v_a)}$ for $ab \in E$, where e_{ab} is the number of edges running from node a to b, and $deg(v_a)$ is the degree of node a. For ab pairs which do not exist in the graph, we set $e_{ab} = 0$. If edges have attached weights, e_{ab} is the sum of the weights of all participating edges.

We initialize the embedding matrix $\mathbf{T}_0 \in \mathbb{R}^{|V| \times d} \sim U(-1, 1)$, where d is the embedding dimensionality. Then for I iterations, we multiply matrix $\mathbf{M}$ and T_i, and normalize rows to keep constant L_2 norm. $\mathbf{T}_I$ is our final embedding matrix. If the graph is too big to be embedded at once due to memory constraints, it can be split and merged in a natural way by weighted averaging of the composing parts' embeddings. The full procedure (including graph splitting and averaging) is show in detail in Algorithm 1.

Initial vectors in matrix $\mathbf{T}_0$ need to 1) be different from each other so that the algorithm does not collapse to the same representation 2) have similar pairwise distances in order to avoid 'false friends' which are accidentally close in the embedding space. Matrix initialization from the uniform distribution creates vectors which fulfill these conditions in high dimensional space.

Cleora works analogously to an ensemble of d models, as each dimension is optimized separately from others (see the column-wise matrix multiplication operation in Figure 2. The only operation which takes into account all dimensions (and as such, allows some information sharing) is the L_2 normalization. The normalization step ensures numeric stability, preventing the norm of embedding vectors from collapsing towards zero during repeated multiplication by the transition matrix, of which the determinant is ≤ 1.

The method can be interpreted as iterated L_2-normalized weighted averaging of neighboring nodes' representations. After just one iteration, nodes with similar 1-hop neighborhoods in the network will have similar representations. Further iterations extend the same principle to q-hop neighborhoods.

```
Data: Graph $G = (V, E)$ with set of nodes $V$ and set
      of edges $E$, iteration number $I$, chunk number
      $Q$, embedding dimensionality $d$
Result: Embedding matrix $\mathbf{T} \in \mathbb{R}^{|V| \times d}$
Divide graph $G$ into $Q$ chunks. Let $G_q$ be the $q$-th
  chunk with edge set $E_q$;
for $q$ from 1 to $Q$ do
    For graph chunk $G_q$ compute random walk
      transition matrix $\mathbf{M}_q = \frac{e_{ab}}{deg(v_a)}$ for $ab \in E_q$,
      where $e_{ab}$ is the number of edges running from
      node $a$ to $b$ ;
    Initialize chunk embedding matrix $\mathbf{T}_0^q \sim U(-1, 1)$
      ;
    for $i$ from 1 to $I$ do
        $\mathbf{T}_i^q = \mathbf{M}_q \cdot \mathbf{T}_{i-1}^q$;
        $L_2$ normalize rows of $\mathbf{T}_i^q$;
    end
    $\mathbf{T}^q = \mathbf{T}_I^q$;
end
for $v$ from 1 to $|V|$ do
    $\mathbf{T}_{\mathbf{v},*} = \sum_{q=1}^{Q} w_{qv} \times \mathbf{T}^{\mathbf{q}}_{\mathbf{v},*}$ where $w_{qv}$ is the node
      weight $w_{qv} = \frac{|n \in V_q : n = n_v|}{\sum_{k=1}^{Q} |n \in V_k : n = n_v|}$
end
```

Algorithm 1: Cleora algorithm.

Thanks to the weighted averaging approach two useful operations become natural:

- Creation of new node embedding by weighted averaging of all neighbors' representations. With I being the maximum iteration number, the representations should ideally come from $(I - 1)$th iteration to mirror the last matrix multiplication and average. Using representations from Ith iteration is not an error either, but caution is advised as the embedding quality can deteriorate quickly with increasing iteration number (see Section x).
- Obtaining node embedding from a large partitioned graph by weighted averaging of the representation from each slice.

IV. Implementation

Cleora is implemented in Rust in a highly parallel architecture, using multithreading and adequate data arrangement for fast CPU access.

A. Modes of Operation

Cleora ingests text files with relational tables of rows representing a heterogeneous hypergraph. The same input will be treated differently depending on parameters, which define the meaning of particular columns, and as such - the actual graph that will be created. Each column of nodes in the input file can be defined with a number of properties represented with column modifiers. Following column modifiers are available:

- `transient` - the field is virtual - it is considered during embedding process, no entity is written for the column

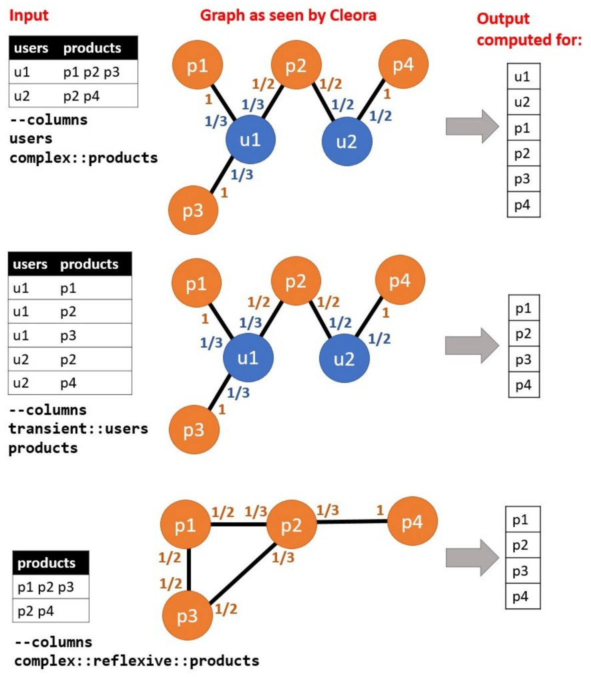


Fig. 1: Examples of various embedding modes and the resulting graph structures and output embeddings. Fractional numbers given next to nodes show values form transition matrix $\mathbf{M}$ for each node.

- `complex` - the field is composite, containing multiple entity identifiers separated by space in TSV or an array in JSON
- `reflexive` - the field is reflexive, which means that it interacts with itself, additional output file is written for every such field
- `ignore` - the field is ignored, no output file is written for the field

Figure 1 shows examples of the most common combinations of column modifiers, with resulting graph structures and the outputs which will be written.

B. *Embedding Computation*

We exemplify the embedding procedure in Figure 2, using a very general example of multiple relations in one graph. Embedding is done in two basic steps: graph construction and training. Such multi-relation configuration is allowed and will result in computation of a separate embedding matrix for each relation pair.

For maximum efficiency we created a custom implementation of a sparse matrix data structure - the `SparseMatrix` struct. It follows the sparse matrix coordinate list format (COO). Its purpose is to save space by holding only the coordinates and values of nonzero entities.

C. *Graph construction*

Graph construction starts with the creation of a helper matrix $\mathbf{P}$ object as a regular 2-D Rust array, which represents the input graph edges according to the selected expansion method (clique, star, or no expansion). An example involving clique expansion is presented in Figure 2 - a Cartesian product (all combinations) of all columns is created. If there are more than 2 relations in the graph, multiple separate $\mathbf{M}$ matrices will be created for each relation pair.

Each entity identifier from the original input file is hashed with `xxhash`[2] - a fast and efficient hashing method. We hash the identifiers to store them in a unified, small data format.

Subsequently, for each relation pair from matrix $\mathbf{P}$ we create a separate matrix $\mathbf{M}$ as a `SparseMatrix` object (the matrices $\mathbf{M}$ will usually hold mostly zeros). Each matrix $\mathbf{M}$ object is produced in a separate thread.

D. *Training*

In this step training proceeds separately for each matrix $\mathbf{M}$, so we will now refer to a single object $\mathbf{M}$. The matrix $\mathbf{M}$ is multiplied by a freshly initialized 2-D array representing matrix $\mathbf{T}_0$ - this array will represent node embeddings. Multiplication is done against each column of matrix $\mathbf{T}_0$ object in a separate thread. The obtained columns of the new matrix $\mathbf{T}_1$ object are subsequently merged into the full matrix. $\mathbf{T}_1$ matrix is L2-normalized, again in a multithreaded fashion across matrix columns. The appropriate matrix representation lets us accelerate memory access taking advantage of CPU caching.

Finally, depending on the target iteration number, the $\mathbf{T}_1$ matrix object is either returned as program output and printed to file, or fed for next iterations of multiplication against the matrix $\mathbf{M}$ object.

E. *Memory Consumption*

Memory consumption is linear to the number of nodes and edges. To be precise, during training we need to allocate space for the following:

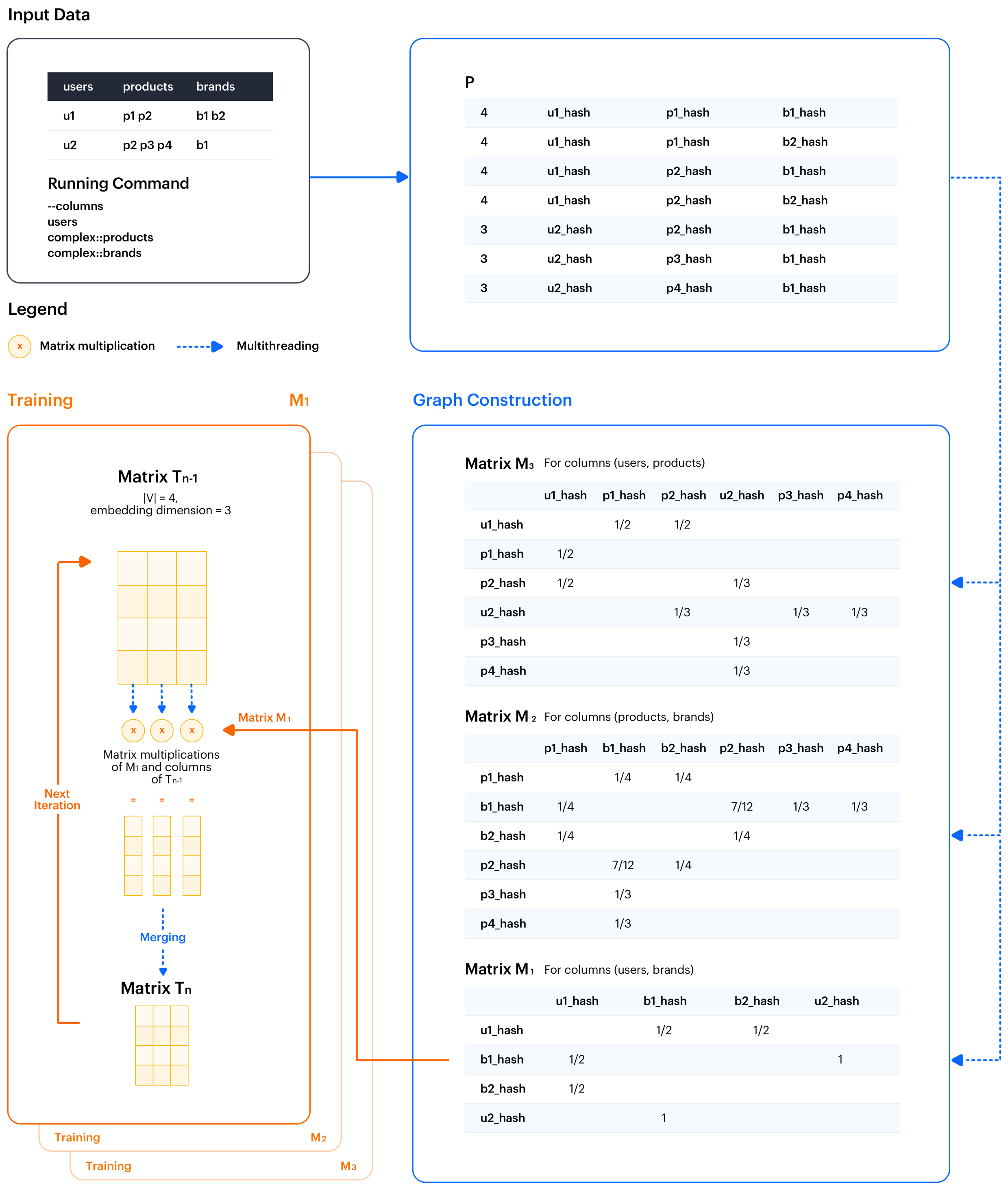
- $|V|$ objects of 40 bytes to store the matrix $\mathbf{P}$;
- $2 \times |E|$ objects of 24 bytes (in undirected graphs we need to count an edge in both directions) to store the matrix $\mathbf{M}$;
- $2 \times d \times |V|$ objects of 4 bytes to store the embedding matrix $\mathbf{T}$.

As such, the total memory complexity is given by $O(|V|(1+2d)+2|E|)$.

V. Experiments

To evaluate the quality of Cleora embeddings, we study their performance on two popular tasks: link prediction [24] and node classification. As competitors we consider recent state-of-the-art methods designed for big data: PBG and GOSH, as well as classic models: LINE and Deepwalk. The details on the competitors' algorithms are described in Section II (Related Work). For each competitor, we use its original author's implementation. We train each model with the best parameter configurations reported in their original papers or

[2]https://cyan4973.github.io/xxHash/



users	products	brands
u1	p1 p2	b1 b2
u2	p2 p3 p4	b1

```
--columns
users
complex::products
complex::brands
```

P

4	u1_hash	p1_hash	b1_hash
4	u1_hash	p1_hash	b2_hash
4	u1_hash	p2_hash	b1_hash
4	u1_hash	p2_hash	b2_hash
3	u2_hash	p2_hash	b1_hash
3	u2_hash	p3_hash	b1_hash
3	u2_hash	p4_hash	b1_hash

Matrix M_3 For columns (users, products)

	u1_hash	p1_hash	p2_hash	u2_hash	p3_hash	p4_hash
u1_hash		1/2	1/2			
p1_hash	1/2					
p2_hash	1/2			1/3		
u2_hash			1/3		1/3	1/3
p3_hash				1/3		
p4_hash				1/3		

Matrix M_2 For columns (products, brands)

	p1_hash	b1_hash	b2_hash	p2_hash	p3_hash	p4_hash
p1_hash		1/4	1/4			
b1_hash	1/4			7/12	1/3	1/3
b2_hash	1/4			1/4		
p2_hash		7/12	1/4			
p3_hash		1/3				
p4_hash		1/3				

Matrix M_1 For columns (users, brands)

	u1_hash	b1_hash	b2_hash	u2_hash
u1_hash		1/2	1/2	
b1_hash	1/2			1
b2_hash	1/2			
u2_hash		1		

Fig. 2: Cleora Architecture.

Name	**Facebook**	**YouTube**	**RoadNet**	**LiveJournal**	**Twitter**
#Nodes	22,470	1,134,890	1,965,206	4,847,571	41,652,230
#Edges	171,002	2,987,624	2,766,607	68,993,773	1,468,365,182
Average Degree	12	5	3	16	36
Density	6×10^{-4}	4.7×10^{-6}	1.4×10^{-6}	3.4×10^{-6}	9.1×10^{-7}
#Classes	4	47	-	-	-
Directed	No	No	No	Yes	Yes

TABLE I: Dataset characteristics.

Algorithm	**Facebook**	**YouTube**	**RoadNet**	**LiveJournal**	**Twitter**
Total embedding time					
Cleora	00:00:43 h	00:12:07 h	00:24:15 h	01:35:40 h	25:34:18 h
PBG	00:04.33 h	00:54:35 h	00:37:41 h	10:38:03 h	-*
Deepwalk	00:36:51 h	28:33:52 h	53:29:13 h	*timeout*	*timeout*
Training time					
Cleora	00:00:25 h	00:11:46 h	00:04:14 h	01:31:42 h	17:14:01 h
PBG	00:02:03 h	00:24:15 h	00:31:11 h	07:10:00 h	-*

TABLE II: Calculation times of CPU-based methods: Cleora, PBG and Deepwalk. Total embedding times encompass the whole training procedure, including data loading and preprocessing. Training times encompass the training procedure itself, excluding data loading and preprocessing. * - training crashed due to excessive resource consumption.

Algorithm	**Facebook**		**YouTube**		**RoadNet**		**LiveJournal**		**Twitter**	
	MRR	HR@10	MRR	HR@10	MRR	HR@10	MRR	HR@10	MRR	HR@10
Scalable methods										
Cleora	0.0724	0.1761	0.0471	0.0618	0.9243	0.9429	0.6079	0.6665	0.0355	0.076
PBG [21]	0.0817*	0.2133*	0.0321*	0.0640*	0.8717*	0.9106*	0.5669*	0.6730*	-**	-**
GOSH [3]	0.0924*	0.2319*	0.0280*	0.0590*	0.8756*	0.8977*	0.2242*	0.4012*	-**	-**
Non-scalable methods										
Deepwalk [32]	0.0803*	0.1451*	0.1045*	0.1805*	0.9626*	0.9715*	*timeout*	*timeout*	*timeout*	*timeout*
LINE [38]	0.0749*	0.1923*	0.1064*	0.1813*	0.9628*	0.9833*	0.5663*	0.6670*	-**	-**

TABLE III: Link prediction performance results. * - results with statistically significant differences to Cleora according to the Wilcoxon two-sided paired test (p-value lower than 0.05). ** - training crashed due to excessive resource consumption/unscalable architecture.

Algorithm	**Facebook**		**YouTube**	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
Scalable methods				
Cleora	0.9165	0.9166	0.3859	0.3077
PBG	0.9258	0.9262	0.3567*	0.2459*
GOSH	0.8312*	0.8305*	0.3166*	0.2245*
Non-scalable methods				
Deepwalk	0.9349*	0.9354*	0.3166*	0.2245*
LINE	0.9442*	0.9446*	0.4008*	0.3338*

TABLE IV: Classification performance results. * - results with statistically significant differences to Cleora according to the Wilcoxon two-sided paired test (p-value lower than 0.05).

repositories per each dataset. For datasets which do not have a per-model best configuration, we perform an experimental grid search.

We conduct all experiments on two machines: 1) Standard E32s v3 Azure (32 vCPUs/16 cores) and 256 GB RAM for CPU-based methods, and 2) 128 GB RAM, 14 core (28 HT threads) Intel Core i9-9940X 3.30GHz CPUs and GeForce RTX 2080 Ti 11GB GPU for GPU-based methods.

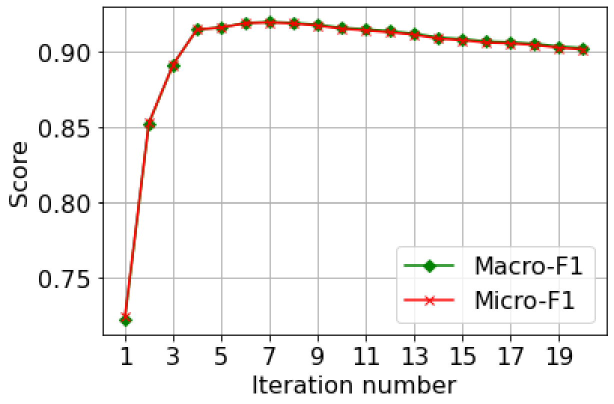
A. Datasets

We use 5 public datasets, summarized in Table I. We focus on picking the most referential datasets, which are already popular benchmarks. Moreover, we take care to include datasets of various sizes, spanning from medium-sized (Facebook) to massive (Twitter). For each experiment we randomly sample out 80% of the edges found in each dataset as trainset (both for the embeddings and the proceeding proxy tasks) and the rest serves as validation/test dataset.

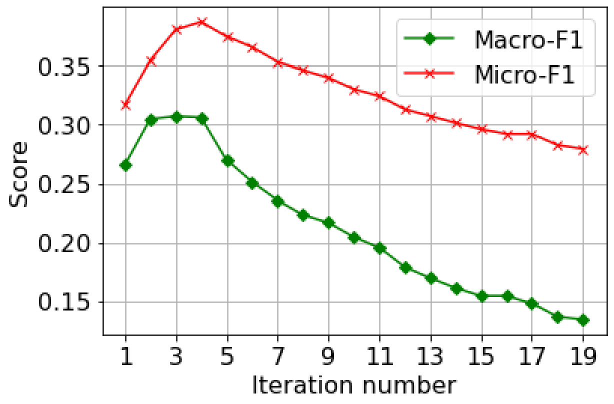
- **Facebook** [36]. In this graph, the nodes represent official Facebook pages while the links are mutual likes between sites. All nodes belong to one of 4 classes defined by Facebook: politicians, governmental organizations, television shows and companies.
- **Youtube** [29]. This graph represents mutual friendships in the Youtube social networks. Nodes are also characterized by class assignments representing membership in groups.
- **RoadNet** [23]. A road network of California. Intersections and endpoints are represented by nodes and the roads connecting these intersections or road endpoints are represented by edges.
- **LiveJournal** [6]. LiveJournal is a free online community with almost 10 million members. The graph represents friendships among members.
- **Twitter** [20]. A subset of the Twitter network with directed follower-following social network. This graph is marked by low reciprocity of relations and a strong presence of influential nodes.

Algorithm	Facebook			YouTube		
	Micro-F1	Macro-F1	MR	Micro-F1	Macro-F1	MR
Original	0.9190	0.9191	457	0.3673	0.3032	3677
1-hop reconstruction	0.8718	0.8586	704	0.3013	0.2046	3751
2-hop reconstruction	0.7856	0.7664	1552	0.2195	0.1136	3943

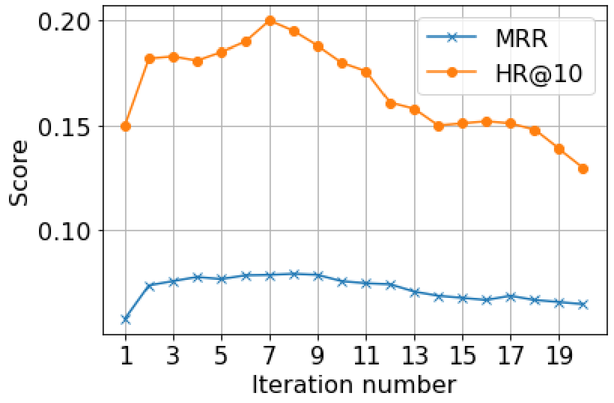
TABLE V: Quality check of reconstructed nodes in a challenging setting where only 30% of all nodes are learned directly and 70% are reconstructed. 1-hop reconstruction nodes are computed from original nodes. 2-hop reconstruction nodes are computed from 1-hop reconstruction nodes.



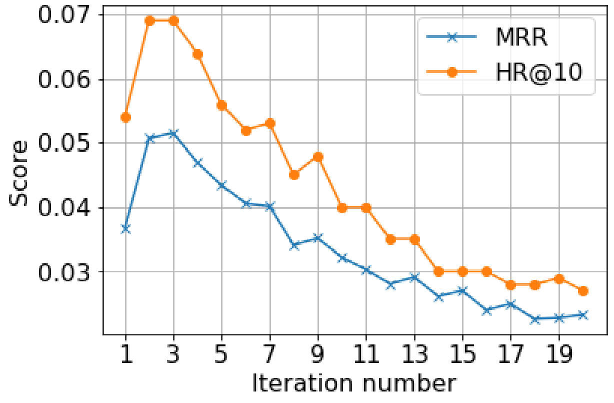
(a) Facebook Dataset - Classification Task.



(b) Youtube Dataset - Classification Task

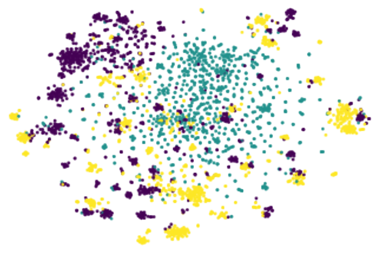


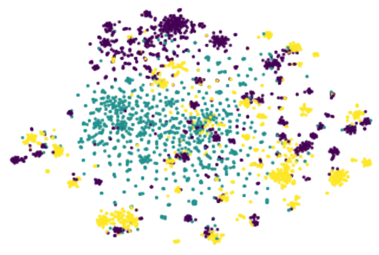
(c) Facebook Dataset - Link Prediction Task.



(d) Youtube Dataset - Link Prediction Task.

Fig. 3: The influence of iteration number on embedding quality.


(a) Cleora


(b) GOSH

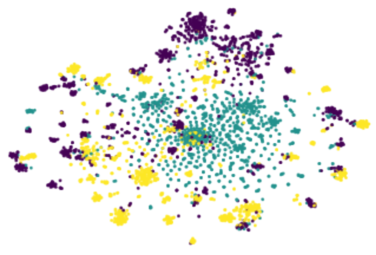

(c) PBG

Fig. 4: t-SNE projections of embeddings learned on Facebook dataset, colored by class assignment. The displayed methods are: a) Cleora b) GOSH c) PBG.

B. Embedding Computation

We use the following configurations for each of the models:

- Cleora: all embeddings are trained with iteration number i = 4 and embedding dimensionality d = 1024. We treat each row of the adjacency matrix as a hyperedge. Hyperedges are expanded with the clique expansion approach in Facebook and RoadNet datasets, and with star expansion in YouTube and LiveJournal datasets (due to the appearance of very large clusters).
- PBG [21]: We use the code from original author repository[3]. We use the following parameter configurations which we find to give the best results:
  - LiveJournal dataset: we reuse the original configuration from the author repository.
  - Facebook, Roadnet datasets:
    `dimension=1024,`
    `global_emb=False,`
    `num_epochs=30,`
    `lr=0.001,`
    `regularization_coef=1e-3,`
    `comparator="dot"`
  - Youtube dataset: same as above, with
    `num_epochs=40`

  For link prediction we use `loss_fn="ranking"` and for classification `loss_fn="softmax"` as suggested in the documentation.
- LINE [38]: We use the code from Graphvite[4], which includes a parallelized GPU-based implementation of the original algorithm. We use the configuration for Youtube dataset from author repository for embedding this dataset. We reuse this configuration for other datasets with epoch number e = 4000. We concatenate the base embeddings with context embeddings as recommended by the authors.
- GOSH [3]: We use the original author code[5]. The model is trained with learning rate lr = 0.045 and epochs e = 1000, defined in the original paper as the optimal configuration for our graph sizes (without any graph coarsening for maximum accuracy).
- Deepwalk [32]: We use the original author code[6]. We reuse the optimal configuration from author paper:
  `-workers 30,`
  `-representation-size 128,`
  `-number-walks 80,`
  `-walk-length 40`

We compare the results of scalable methods (Cleora, PBG, GOSH), contrasting them with unscalable methods (Deepwalk, LINE). We deem LINE unscalable as its implementation cannot embed a graph when the total size of the embedding is larger than the total available GPU memory.

Embedding training times are displayed in Table II. We include only CPU-based methods for fair comparison. PBG is the fastest high-performing CPU-based method we are aware of, and Deepwalk serves as a baseline example of the non-scalable methods. Cleora is shown to be significantly faster than PBG in each case. Training is up to 5 times faster than PBG and over 200 times faster than Deepwalk.

We face technical difficulties with training the embeddings on the Twitter graph. Both GOSH and PBG crash during or just after loading the data into memory due to exhausting the available hardware resources. This demonstrates that Cleora can be trained on hardware which fails for other fast methods (note, though, that our hardware configuration is not small but rather standard). Also, technical errors can stem directly from complex model architecture (e.g. heavy and complicated parallelism in PBG), a problem which does not exist in Cleora due to the simplicity of the algorithm. As a consequence, the only model we evaluate on Twitter is Cleora.

C. Performance Check

Link Prediction. Link prediction is one of the most common proxy tasks for embedding quality evaluation [25]. It consists in determining whether two nodes should be connected with an edge.

We collect the embeddings for all positive pairs (real edges) and for each such pair we generate a negative pair with random sampling. We compute the Hadamard product for each pair, which is a standard procedure observed to work well for performance comparison [13], [15]. Then, we feed the products to a Logistic Regression classifier which predicts the existence of edges. As the graphs we use are mostly large-scale, we resort to step-wise training with the `SGDClassifier` module from `scikit-learn` with a Logistic Regression solver.

For evaluation, we reuse the setting from [21]: for each positive node pair, we pick 10,000 most popular graph nodes, which are then paired with the start node of each positive pair to create negative examples. We rank the pairs, noting the position of the real pair among the 10,001 total examples. In this setting we are able to compute popular ranking-based metrics:

- MRR (mean reciprocal rank). This is the average of the reciprocals of the ranks of all positives. The standard formulation of MRR requires that only the single highest rank be summed for each example i (denoted as $rank_i$) from the whole query set Q:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}.$$

  As we consider all edge pairs separately, this is true by default in our setting. Higher score is better, the best is 1.0.
- HitRate@10. This gives the percentage of valid end nodes which rank in top 10 predictions among their negatives. Higher score is better, the best score is 1.0.

Due to a large size of graphs, we evaluate on 100,000 sampled examples from the testset. The variances of scores among samples drawn with various seeds are very low, on the level of 1e−8 to 1e−12, so the sampling error will have no visible influence on model ranking.

[3]https://github.com/facebookresearch/PyTorch-BigGraph/
[4]https://github.com/DeepGraphLearning/graphvite
[5]https://github.com/SabanciParallelComputing/GOSH
[6]https://github.com/phanein/deepwalk

Results of our evaluation versus the competitors are presented in Table III. The table shows that in spite of its extreme simplicity, the quality of Cleora embeddings is generally on par with the scalable competitors. Cleora performs worse than the competitors on the smallest graph - Facebook, but gets better with increasing graph size. In some cases Cleora reaches results that are better than other fast algorithms aligned for big data - PBG and GOSH. Cleora performs especially well in the MRR metric, which means that it positions the correct node pair high enough in the result list as to avoid punishment with the harmonically declining scores.

Apart from score rankings, another important issue is embedding versatility. This can be problematic in particular for PBG, which defines a number of loss functions linked to the downstream task. Indeed, we note that in order to achieve the top results we observe, it is necessary to train 2 versions of the embeddings: a version for link prediction task with `loss_fn="ranking"` and a version for classification task with `loss_fn="softmax"`. Whenever training with the classification objective, we observe significant drops of performance in link prediction (e.g. up to 5 times lower MRR and HR for Youtube dataset). The other models, including Cleora, do not require separate task-specific parameters.

We are unable to compare our results on the Twitter graph to competitors, thus presenting the results only for Cleora in Table III. Due to large size of the graph and the computed embedding file (over 500 GB, far exceeding our available memory size), we compute the results on a large connected component of the graph, containing 4,182,829 nodes and 8,197,949 edges.

Classification. We evaluate node classification on two datasets which have node labels: Facebook and YouTube. In Facebook, we use the 4 available classes. In YouTube, the number of classes (representing interest group numbers) is on the scale of thousands. We select 47 most numerous classes for our classification objective (following [38]). As performance measures we use micro-F1 and macro-F1 scores from `scikit-learn` library.

The results are displayed in Table IV. The findings confirm the results from link prediction. Cleora is again ranking high, this time usually beating the scalable competitors, and getting close to non-scalable ones.

Figure 4 shows Cleora, GOSH and PBG vectors learned on Facebook and projected to 2-D space using t-SNE [50]. The vectors are colored by their class assignments. It can be observed that Cleora learns an essentially similar data abstraction as the contrastive methods. Additionally, a number of outliers appear in Cleora's visualization, which are nodes from small, strongly interconnected clusters. Such nodes' embeddings get extremely similar to each other and at the same time distant from the rest of the embeddings. As such, Cleora clearly separates strongly connected components of the graph.

Node Reconstruction (Inductivity). We also evaluate the quality of embeddings of new nodes which are added to the graph after the computation of the full graph embedding table. To this end, we split the Facebook and Youtube datasets into a 'seen node' set comprised of 30% of each dataset, and 'new node' set comprised of the remaining 70% of each dataset. This setting is challenging as the vast majority of nodes will need to be reconstructed. At the same time, it gives us enough data to study embedding quality of two kinds of reconstruction: **1-hop reconstruction**, where a new node embedding is created based on 'seen node' embeddings, which have been learned directly from interaction data, and **2-hop reconstruction**, where a new node embedding is created based on reconstructed node embeddings. 2-hop reconstruction is a much harder task, as errors from previous reconstruction step will be accumulated.

Table V shows the results of the reconstruction quality check. We use the previous measures of micro-F1 and macro-F1 in the Node Classification task, also taking into account the MR (Mean Rank) measure, which is a simple average of obtained ranks in the Link Prediction task. The measure is a simplified abstraction of the MRR and HR metrics. 1-hop and 2-hop reconstruction results are computed on the reconstructed embeddings only.

We observe that the task of classification on Facebook dataset notes only a 5% micro-F1 drop in 1-hop node generation, and a 14.5% micro-F1 drop in 2-hop node generation. Youtube faced a stronger drop of 18% and 40%, respectively. A reversed tendency is observed for link prediction: the MR drops are severe for Facebook (MR falling by a half when reconstructed), while being very mild for Youtube (absolute difference in MR being only 266). This suggests that quality of reconstruction is heavily dependent on dataset. Even with 2-hop reconstruction, it is possible to obtain embeddings of high relative quality.

VI. Analysis

A. Optimal iteration number

The iteration number defines the breadth of neighborhood on which a single node is averaged: iteration number i means that nodes with similar i-hop neighborhoods will have similar representations. We show the influence of i on classification performance in Figure 3.

The iteration number is related to the concept of average path length from the area of graph topology. The average path length is defined as the average number of steps along the shortest paths for all possible pairs of network nodes. If the iteration number reaches the average path length, an average node will likely have access to all other nodes. Thus, iteration number slightly exceeding the average path length can be deemed optimal. For example, the average path length for the Facebook graph equals 5.18 and the best i is found to be 5-7 according to Figure 3. In practice however the computation of average path length is significantly slower than the computation of Cleora embeddings. An optimal solution to determining the iteration number is to verify iteration number empirically on a downstream task.

Too large iteration number will make all embeddings gradually more similar to each other, eventually collapsing to an exact same representation. This behavior might be rather slow or abrupt after passing the optimal point depending on the dataset, as evidenced in Figure 3.

	butterfly			**elephant**		
	w2v	Cleora i = 1	Cleora i = 4	w2v	Cleora i = 1	Cleora i = 4
1.	feather	madama	cover	giraffe	dumbo	elephants
2.	shark	wildflower	pictured	orca	elephas	mammoths
3.	unicorn	state	strange	panda	elephantiasis	hunting
4.	eyed	norsemen	photograph	ox	loxodonta	predator
5.	pear	gopal	nice	albino	shrews	covered

	king			**nationalism**		
	w2v	Cleora i = 1	Cleora i = 4	w2v	Cleora i = 1	Cleora i = 4
1.	prince	coretta	iii	imperialism	eurocentrism	colonialism
2.	queen	fahd	conqueror	nazism	chauvinism	political
3.	throne	aleksandar	succeeded	ideology	simmering	domination
4.	antiochus	anshan	queen	fascism	republicanism	imperialism
5.	kings	sihamoni	lombard	conservatism	arabization	establishment

TABLE VI: Results of the homophily vs. structural equivalence experiment. We contrast Cleora with Word2Vec [28] showing varied levels of homophily and structural equivalence depending on iteration number.

B. Homophily and Structural Equivalence

The idea of node similarity can be considered under various formulations. [15] propose to focus on two such aspects: homophily [27] and structural equivalence. Under the homophily criterion, node embeddings should be similar when the nodes are highly interconnected. Structural equivalence, on the other hand, stresses that similarity of node embeddings should be based on their role within a graph (e.g. two nodes which are centers of large clusters should be similar, or two leaf nodes should be similar). Under this definition, Cleora leans strongly towards the homophily criterion. An extreme example of this behavior would be the embedding of a leaf node. Leaf nodes have only one neighbor, so their embedding is always equal to the embedding of their single neighbor from the previous iteration. As such, two leaf nodes will have very different embeddings if their geodesic distance is higher than the iteration number. With rising iteration number, more structural knowledge flows into the embeddings.

We exemplify this property contrasting Cleora with Word2Vec [28] in Table VI. Both models are trained on the `enwik8` dataset[7]. Cleora is trained with clique expansion approach on token sequences with window sizes of 5,7,9. Representations computed with each window size were concatenated.

We query each set of embeddings with selected nouns to find their nearest neighbors. Word2Vec embeddings exhibit strong structural equivalence, usually keeping to the part of speech (PoS) of the query. PoS is an important structural characteristic of a word, defining its role within a sentence. On the other hand, Cleora i = 4 usually returns similar nouns but sometimes also related words of other PoS, e.g `elephant` is related to `hunting`. Cleora i = 1 is an extreme case of homophily in close neighborhoods, returning words which appear in one phrase, e.g. `madama butterfly`, `elephant dumbo`.

C. Complement and Substitute Prediction

Node similarity problem can be cast as complement versus substitute identification [26], which is especially interesting in e-commerce applications. Substitutes are defined as products that can be purchased instead of each other, while complements are products that can be purchased in addition to each other (they often appear together in the same basket). As such, complements are expected to be similar if their 1-hop neighborhoods are similar, boiling down to neighborhood prediction. Substitutes should be close in terms of a broader notion of semantic similarity of nodes.

Thanks to the influence of the iteration number, Cleora can represent either the complementary or the substitute relation. To exemplify this property, we preprocess the "Complete Journey" Dunnhumby dataset[8]. The dataset includes transactions of 2,500 households within all categories in the store, gathered over 2 years. We embed the product baskets with Cleora, expanding each basket with the clique expansion approach. We use two configurations: iteration number i = 1 and i = 4, and compute the nearest neighbors for each product using cosine similarity.

Extracts from our results are displayed in Table VII. With i = 1 the closest embeddings mirror the complement relation. For example, instant Ramen soups seem to be often bought together with automotive parts, dips and sweets which suggests that they might be bought by drivers stopping by. Regarding bread, there seems to be a marked difference in the perception of regular white bread and French bread. White bread is associated with everyday meals, while French bread is bought together with greeting cards, candy and gifts. As such, it is much more strongly connected to special occasions such as celebration of Valentine's Day and parties. In the case of i = 4, products are more closely aligned with similar products (possible substitutes): ramen soups with other ramen soups of various kinds (only the product IDs differ here). Breads are close to cereal, pies, sandwiches and other bakery products but also meats.

Note that nearest neighbor search between i = 1 embeddings does not achieve identical goals as algorithms of association rule mining [1], [2], [14]. Due to the averaging operation over neighbors, embeddings of items appearing in small baskets will be much more similar to each other than if they appeared in bigger baskets. Thus, nearest neigbor search with Cleora strongly promotes smaller, more selective clusters.

[7]https://deepai.org/dataset/enwik8

[8]https://www.dunnhumby.com/source-files/

	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ	
	1 iteration (Complement)	4 iterations (Substitute)
1.	AUTOMOTIVE PRODUCTS 4 CT	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ
2.	PROCESSED DIPS 15.5 OZ	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ
3.	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ
4.	J-HOOKS JHOOK - HOUSEWARE	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ
5.	PACKAGED CANDY BAGS-CHOCOLATE 11 OZ	SOUP RAMEN NOODLES/RAMEN CUPS 3 OZ
	BAKED BREAD/BUNS/ROLLS MAINSTREAM WHITE BREAD 20 OZ	
	1 iteration (Complement)	4 iterations (Substitute)
1.	BAKED BREAD/BUNS/ROLLS DINNER ROLLS 11 OZ	SMOKED MEATS MARINATED
2.	CHIPS&SNACKS MISC 3.5 OZ	PICKLE/RELISH/PKLD VEG PICKLES
3.	SPRING/SUMMER SEASONAL SALLY HANSEN	PNT BTR/JELLY/JAMS JELLY
4.	DRY NOODLES/PASTA SPAGHETTI DRY 16 OZ	COLD CEREAL KIDS CEREAL
5.	BEERS/ALES BEERALEMALT LIQUORS 40 OZ	BREAKFAST SAUSAGE/SANDWICHES PATTIES
	BREAD BREAD:ITALIAN/FRENCH	
	1 iteration (Complement)	4 iterations (Substitute)
1.	LUNCHMEAT PEPPERONI/SALAMI 3 OZ	REFRIGERATED DOUGH PRODUCTS ROLLS
2.	CANDY BAGS-NON CHOCOLATE 4.25 OZ	SEAFOOD - FROZEN SEAFOOD-FRZ-RW-ALL
3.	GREETING CARDS/WRAP/PARTY SPLY PARTY	BAKED SWEET GOODS SNACK CAKE - PACK 5.7 OZ
4.	VALENTINE VALENTINE GIFTWARE/DECOR 5 CT	PIES PIES: PUMPKIN/CUSTARD
5.	CANDY - CHECKLANE CANDY BARS (SINGLES)	LUNCHMEAT HAM 9 OZ

TABLE VII: Examples of complement vs. substitute prediction on shopping baskets from Dunnhumby dataset. We show that low iteration numbers produce embeddings which reflect the complement relation. On the other hand, high iteration numbers produce embeddings which reflect the substitute relation.

VII. Summary

We have presented Cleora - a simple, purely unsupervised embedding algorithm which learns representations analogous to contrastive methods. Cleora is much faster than other CPU-based methods. Moreover, it has useful extra properties, such as node embedding inductivity and the ability to compute partial embeddings on chunked graph, which can be subsequently merged. The iteration number allows to obtain various functionalities, for example complement or substitute prediction. We open-source Cleora to the community in order to aid reproducibility and allow a wide use of our method, including commercial use.

References

[1] Rakesh Agarwal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, 1994.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining associations between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[3] Taha Atahan Akyildiz, Amro Alabsi Aljundi, and Kamer Kaya. Gosh: Embedding big graphs on small hardware. In *49th International Conference on Parallel Processing - ICPP*, ICPP '20, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Nikolaos Aletras and Benjamin Paul Chamberlain. Predicting twitter user socioeconomic attributes with network and language information. In *Proceedings of the 29th on Hypertext and Social Media*, pages 20–24. 2018.

[5] Kimitaka Asatani, Junichiro Mori, Masanao Ochi, and Ichiro Sakata. Detecting trends in academic research from a citation network using network representation learning. *PloS one*, 13(5):e0197260, 2018.

[6] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 44–54, New York, NY, USA, 2006. Association for Computing Machinery.

[7] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press, 2002.

[8] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26:2787–2795, 2013.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 2018.

[11] Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. Biased graph walks for rdf graph embeddings. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, pages 1–12, 2017.

[12] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000.

[13] Ayushi Dalmia, Ganesh J, and Manish Gupta. Towards interpretation of node embeddings. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 945–952, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[14] Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI*, volume 90, page 65, 2003.

[15] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1024–1034. Curran Associates, Inc., 2017.

[17] John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In *Advances in Neural Information Processing Systems*, pages 15820–15831, 2019.

[18] I. T. Jolliffe. *Principal Component Analysis and Factor Analysis*, pages 115–128. Springer New York, New York, NY, 1986.

[19] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

[20] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[21] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[22] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

[23] Jure Leskovec, Kevin Lang, Anirban Dasgupta, and Michael Mahoney.

Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6, 11 2008.

[24] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007.

[25] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[26] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794, 2015.

[27] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[29] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.

[30] Sameh K Mohamed, Vít Nováček, and Aayah Nounu. Discovering protein drug targets using knowledge graph embeddings. *Bioinformatics*, 36(2):603–610, 08 2019.

[31] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA, 2011. Omnipress.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[33] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don't walk, skip! online learning of multi-scale network embeddings. ASONAM '17, page 258–265, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Chanathip Pornprasit, Xin Liu, Natthawut Kertkeidkachorn, Kyoung-Sook Kim, Thanapon Noraset, and Suppawong Tuarob. Convcn: A cnn-based citation network embedding algorithm towards citation recommendation. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, pages 433–436, 2020.

[35] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[36] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.

[37] T. Schumacher, Hinrikus Wolf, Martin Ritzert, Florian Lemmerich, J. Bachmann, Florian Frantzen, Max Klabunde, M. Grohe, and M. Strohmaier. The effects of randomness on the stability of node embeddings. *ArXiv*, abs/2005.10039, 2020.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*. ACM, 2015.

[39] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[40] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[41] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML), 2016.

[42] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. 03 2018.

[43] Ning Wu, Xin Wayne Zhao, Jingyuan Wang, and Dayan Pan. Learning effective road network representation with hierarchical graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 6–14, 2020.

[44] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[45] Heng Yao, Yunjia Shi, Jihong Guan, and Shuigeng Zhou. Accurately detecting protein complexes by graph embedding and combining functions with interactions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(3):777–787, 2019.

[46] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, 2020.

[47] Yuan Zhang, Tianshu Lyu, and Yan Zhang. Cosine: Community-preserving social network embedding from information diffusion cascades. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[48] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.

[49] Zhaocheng Zhu, Shizhen Xu, Meng Qu, and Jian Tang. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. *The World Wide Web Conference*, 2019.

[50] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.