GPU-embedding of kNN-graph representing large and high-dimensional data

Bartosz Minch, Mateusz Nowak, Rafał Wcisło, and Witold Dzwinel

AGH University of Science and Technology, Kraków, Poland
{minch, wcislo, dzwinel}@agh.edu.pl, mtsznowak9@gmail.com

Abstract. Interactive visual exploration of large and multidimensional data still needs more efficient $ND \rightarrow 2D$ data embedding (DE) algorithms. We claim that the visualization of very high-dimensional data is equivalent to the problem of 2D embedding of undirected kNN-graphs. We demonstrate that high quality embeddings can be produced with minimal time&memory complexity. A very efficient GPU version of IVHD (interactive visualization of high-dimensional data) algorithm is presented, and we compare it to the state-of-the-art GPU-implemented DE methods: BH-SNE-CUDA and AtSNE-CUDA. We show that memory and time requirements for IVHD-CUDA are radically lower than those for the baseline codes. For example, IVHD-CUDA is almost 30 times faster in embedding (without the procedure of kNN graph generation, which is the same for all the methods) of the largest ($M = 1.4 \cdot 10^6$) YAHOO dataset than AtSNE-CUDA. We conclude that in the expense of minor deterioration of embedding quality, compared to the baseline algorithms, IVHD well preserves the main structural properties of ND data in $2D$ for radically lower computational budget. Thus, our method can be a good candidate for a truly big data ($M = 10^{8+}$) visualization.

Keywords: high-dimensional data · data embedding · kNN graph visualization · GPU implementation

1 Introduction

In the age of data science, interactive visualization of large high-dimensional data is an essential tool in knowledge extraction. It allows for both the insight into data structure and its interactive exploration by direct manipulation on the whole or a fragment of a dataset. This way, it is possible to observe the shapes and mutual location of classes, as well as remove irrelevant data samples and identify the outliers. The multiscale structure can be explored visually by changing data embedding strategies and visualization modes (e.g., the type of the loss function), and zooming-in and out selected fragments of 2D (3D) data mappings. Summarizing, interactive visualization allows for: 1) instant verification of a number of hypotheses, 2) precise matching of data mining tools to the properties of data investigated, 3) adapting optimal parameters to machine learning algorithms, and 4) selecting the best data representation. Herein, we focus on application of data embedding (DE) methods in the interactive visualization of large ($M{\sim}10^5$-10^6) and high-dimensional $N{\sim}10^{2+}$ data.

Data embedding (DE) is defined as a transformation $\mathbf{B}{:}Y{\rightarrow}X$ of N-dimensional (ND) dataset $\Re^N \ni Y = \{y_i\}_{i=1,...M}$ into its n-dimensional (nD) representation $\Re^n \ni X =$

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

$\{x_i\}_{i=1,...M}$, where $N>>n$ and M is the number of ND feature vectors y_i and corresponding nD embeddings x_i. The mapping $\mathbf{B}$ can be perceived as a lossy compression of data. It is performed by minimizing a loss function $E(\|Y - X\|)$, where $\|.\|$ is a measure of topological dissimilarity between Y and X. Due to the high complexity of the low-dimensional manifold, immersed in the ND feature space and occupied by data samples Y, perfect embedding of Y in the nD space is possible only for trivial cases.

In the context of high-dimensional data visualization, we assume that $n = 2$. Data embedding to 3D can be processed in a similar way. As shown in many papers (see, e.g., [9, 17]), DE of large data that is both sufficiently precise in reconstruction of ND data topology, and simultaneously, computationally affordable, is the algorithmic challenge. To preserve topological properties of Y in X both the classical MDS (multidimensional scaling) methods (e.g., [5]) and the state-of-the-art (SOTA) clones of the stochastic neighbor embedding (SNE) concept (e.g., [11, 14, 19]) require computing and storing two O(M^2) arrays: (1) the dissimilarities between data vectors Y and (2) the Euclidean distances between their 2D embeddings X. That is why, the SOTA visualization algorithms, such as t-SNE [16] and its clones, suffer from high $O(M^2)$ time&memory complexity. The time complexity can be decreased to $O(MlogM)$ and even $O(M)$, by using the approximated versions of t-SNE, such as BH-SNE [15] and other its variants and approximations [14, 19]. Meanwhile, the memory complexity remains $O(M^2)$ which considerably limits its use for truly big data and new parallel computer architectures. Summing up, for many of the SOTA embeddings: [1, 17, 22, 23]:

1. The time complexity of the DE procedure is dominated by the construction of the kNN-graph, which is generally $O(MlogM)$ complex (for exact kNN search algorithms).
2. The computational efficiency of the DE process vastly depends on the loss function and optimization procedure applied for its minimization.
3. Calculating gradient of the loss function is $O(N{\cdot}M)$ complex, but with large proportionality coefficient, which is dominated by a relatively large value of k.

Interactive visual data exploration involves very strict time&memory performance regimes while the SOTA DE algorithms are still too time&memory consuming. The main contribution of this paper is developing a method to overcome this flaw and optimize the data embedding process. To this end, for DE, we adapt the same idea we used previously for large unweighted and undirected graphs visualization (GV) [6]. Furthermore, we clearly demonstrate that the DE of very high-dimensional data can be treated as a subproblem of GV what has not been said explicitly before (maybe except of recently published [13]). Consequently, the visualization of high-dimensional data comes down to the visualization of the kNN graphs (kNN-graphs) where each of the nodes represents a feature vector. However, unlike the other DE algorithms which also employ this trick (e.g., BH-SNE [15], LargeVis [22], UMAP [17]), we show that the value of k can be much smaller, such as $k{\sim}2, 3$ (compare it to BH-SNE [15] and LargeVis [22] where $k{\sim}10^2$). Moreover, for truly high-dimensional data ($N \gtrapprox 30$), i.e., when the effects of "curse of dimensionality" on the ND space topology become evident, the floating point dissimilarities used for kNN-graph construction can be discarded. Instead, the integer indices to only a few nn nearest neighbors (instead of index k, we use nn to

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

be consistent with the notation we use in the rest of this paper) have to be kept in the computer memory. We show here that if for each ND feature vector we:

1. store only indices of $nn < 6$ nearest neighbors ($nn = 2$ in the most cases is sufficient),
2. select very few (often just one) random neighbors rn (similar to *negative sampling* procedure) during calculations,
3. define binary distances (0 and 1, respectively) to the nearest and random neighbors,

it is possible to radically simplify the loss function and, consequently, decrease the CPU time required for its minimization. Thus, we are able to reconstruct the ND data structure in 2(3)D space in a very efficient way both in terms of computational time and storage.

Summarizing, in the paper, we demonstrate that the data embedding (DE) is equivalent to the visualization of unweighted kNN-graphs, constructed for the source Y data. The principal contribution of this paper is the essential improvement of the time&memory complexity of DE at the expense of a minor deterioration of embedding quality. Consequently, the proposed data mapping methodology allows for interactive visualization of much larger data than the state-of-the-art DE algorithms. Moreover, we demonstrate that our IVHD algorithm can be implemented in an efficient way in GPU/CUDA environment. We compare this implementation to the fastest publicly available GPU data embeddings: BH-SNE-CUDA [4] and Anchor-t-SNE [10].

2 Methods

Despite that there are many algorithms for visualization of high-dimensional data and that this topic has been extensively studied for years, to the best of our knowledge, there are only a few implementations of modern data embedding algorithms in GPU/CUDA environment [4, 10, 18]. We will focus on the publicly available ones, which generate the best embeddings of large datasets: BH-SNE-CUDA [4] and Anchor-t-SNE [10] (At-SNE). These algorithms base on the well known t-SNE (t-distributed Stochastic Neighbor Embedding) concept [16], and consist of two stages: (1) generate of a weighed kNN-graph; (2) run a proper embedding procedure which is based on: (2a) definition of a loss function and (2b) its minimization.

2.1 *k*NN graph generation

kNN-graph approximates a nD non-Cartesian manifold immersed in $\mathbb{R}^N$ sampled by the feature vectors y_i. We consider here the kNN-graph construction procedure shipped by the FAISS library [12]. Its authors claim that it is currently the fastest available kNN search algorithm implemented on GPU. The FAISS kNN-search procedure merges a very efficient and well-parallelized exact kNN algorithm and indexing structures that allow to perform approximate search. To achieve a high-efficiency search with a good accuracy we employ the IVFADC [2] indexing structure. It uses two levels of quantization combined with the vector encoding for compressing high-dimensional vectors. The main idea behind the index usage is to split the input space and divide all input samples

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

into a number of clusters represented by their centroids. To handle a query, the algorithm compares it with the centroid centers. It picks the centroid that is the most similar to the query vector and performs an exact search within the set of samples belonging to this centroid. Apart from the efficient kNN algorithm, its CUDA implementation is extremely well optimized [12]. We use the same FAISS kNN-graph generation procedure in both the baseline and IVHD algorithms.

2.2 Loss functions

BH-SNE-CUDA The group of methods based on the SNE concept [16] defines the similarity of two samples i and j in terms of probabilities p_{ij} (in Y) and q_{ij} (in X), that i would pick j as its neighbor and vice versa. These probabilities are functions of distances between samples in Y and their embeddings in X, respectively. Let $\mathbf{D} = [D_{ij}]$ is the distance table in Y and D_{ij} are the distances between i and j feature vectors y_i and y_j, while $\mathbf{d} = [d_{ij}]$ is the respective distance array in X. Then, the loss function $C{=}E(\mathbf{D}, \mathbf{d})$ is defined by the Kullback–Leibler (KL) divergence:

$$C(.) = E(\mathbf{D}, \mathbf{d}) = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{1}$$

where, for t-SNE algorithm, p_{ij} is approximated by the Gaussian $\mathcal{N}(y_i, \sigma)$, while q_{ij} is defined by the Cauchy distribution [16]. The p_{ij} and q_{ij} are defined as follows:

$$p_{ij} = \frac{exp(-D_{ij}^2/2\sigma_i^2)}{\sum_{k \neq l} exp(-D_{kl}^2/2\sigma_i^2)} \tag{2}$$

$$q_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \neq l}(1 + d_{kl}^2)^{-1}} \tag{3}$$

The BH-SNE (Barnes-Hut t-SNE [15]) is an approximation of the t-SNE method that can reduce computational complexity of the DE from $O(M^2)$ to $O(MlogM)$ by using Barnes-Hut approximation but at the cost of increasing the algorithmic complexity and, consequently, decrease of the parallelization efficiency [15]. Its GPU version - the BH-SNE-CUDA algorithm - does not introduce any changes to the BH-SNE [15] algorithm, and just matches the instructions and data flows to the GPU architecture [4].

AtSNE-CUDA Recently published AtSNE-CUDA (Anchor-t-SNE [10]) algorithm was created to deal with the t-SNE issues such as sensitivity to initial conditions and inadequate reconstruction of the global data structure. The Authors of the AtSNE-CUDA method claim that their method is 50% faster than BH-SNE-CUDA and has lower memory requirements. As shown in [10], it still generates good quality embeddings though some kNN quality scores are better for the classic t-SNE or the LargeVis algorithms [22].

Similar to all t-SNE clones, AtSNE minimizes the regularized KL divergence. Information about a local structure of data can be acquired from the set of approximated kNN neighbors of each y_i. Meanwhile, to reconstruct the global structure of data, x_i points receive "pulling forces" from, so called, *anchor points* generated from the original data. This way the *anchor points* are responsible for maintaining the mutual positions and

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

shape of classes in X. Consequently, the hierarchical embedding [10] optimizes both the positions of the *anchor points* and regular data samples.

Let A represent the set of *anchor points* in the high-dimensional space, and B its low-dimensional embedding. The probabilities $P(Y)$ and $Q(X)$ denote the high and low-dimensional distributions of the *anchor points*, respectively. To preserve both the global and local information, the following loss function are minimized:

$$E(.) = \sum_i KL(P(Y)||Q(X)) + \sum_i KL(P(A)||Q(B)) + \sum_i ||b_i - \frac{\sum_{y_k \in C_{b_i}} y_k}{|C_{b_i}|}||, \quad b_i \in C_{b_i} \quad (4)$$

where C_{b_i} denotes the set of points whose K-means cluster center is b_i. The last term is a regularization term explained in details in [10].

IVHD Unlike t-SNE based BH-SNE-CUDA and AtSNE-CUDA, IVHD-CUDA utilizes classical MDS stress function. However, the number and sort of distances IVHD employs, are radically different than those in classical MDS and the baseline algorithms.

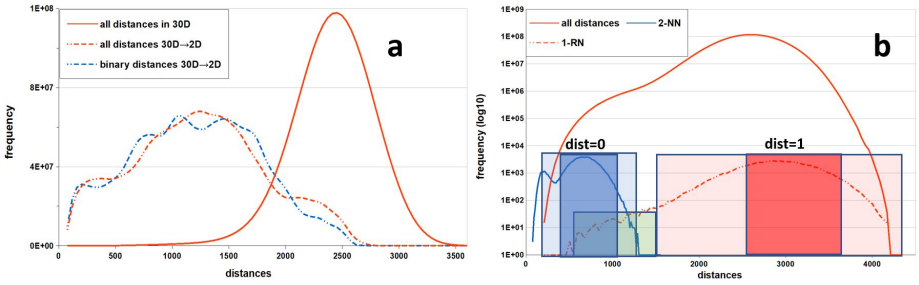


Fig. 1. The envelopes of histograms for MNIST dataset (after PCA transformation $784D \rightarrow 30D$). *Red solid line*: all **D** distances (a - linear, b - logarithmic scale); a) *dashed lines*: all **d** distances; b) *blue solid line*: **D** distances only between samples and their 2-NNs, and *red dashed line*: **D** distances only between samples and one random neighbor.

In Fig.1 we display the envelopes of histograms of both **D** and **d** distances for MNIST dataset before and after IVHD embedding. Although MNIST dataset has a varied structure, the envelope of **D** histogram in linear coordinates (Fig.1a) is perfectly bell-shaped, while that for **d** is more deformed but still resembles the Cauchy distribution. To increase distances diversification (see Fig.1b), instead of all $M(M-1)/2$ floating point distances we can consider binary distances to only a few (nn) nearest neighbors and just one (rn) randomly selected neighbor. This is because, the most of real distances (95%) to the nearest and the random neighbors are located in separated and rather distant intervals (darker blue and red boxes in Fig.1b). For higher dimensions, the random neighbors are getting almost equidistant from y_i due to the "curse of dimensionality" effect. Whereas, the overlapping region (green) contains only 0.3% of distances. So, we can assume additionally that $O_{nn}(i) \cap O_{rn}(i) = \emptyset$. However, this assumption is superfluous because the probability of picking the nearest neighbor as a

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

random neighbor is negligibly small for large N. As shown in Fig.1a, for non-binarized and binarized source distances, their histograms for respective 2D embeddings are very similar. Thus, let $O_{nn}(i)$ and $O_{rn}(i)$ will be the sets of indices of nn nearest (connected) neighbors and rn (unconnected) random neighbors of a feature vector y_i in kNN-graph, respectively. We define binary dissimilarity measure as follows (see Fig.1b):

$$\forall y_i \in Y : D_{ij} = \begin{cases} 0 \ if \ j \in O_{nn}(i) \\ 1 \ if \ j \in O_{rn}(i) \end{cases} . \tag{5}$$

Thus, unlike in the baseline algorithms, we are not interested even in an approximated ordering of kNNs for each $y_i \in Y$. This is justified for small nn, because distances to a few first NN, in general, cannot differ too much (see blue plot in Fig.1b) and the ordering of NN can results from measurement errors. We assume, that the number of nn neighbors has to fulfill two conditions. Firstly, the kNN-graph should be fully (or approximately - i.e., the size of the largest component should be comparable to the size of the full graph) connected. For example, for MNIST and FMNIST datasets (see Table 1) and $k = nn = 2$ the largest components consist of 99% of nodes, and respectively: SmallNorb ($nn = 5$ and 82%), RCV-Reuters ($nn = 3$ and 95%). Secondly, the kNN-graph augmented with approximately rn edges should be at least a minimal n-rigid graph (in 2D: 2-rigid). The term "rigidity" can be understood as a property of a nD structure made of rods (distances) and joints (data vectors) that it does not bend or flex under an applied force. The lower band of the number of connections L, required for making the augmented kNN-graph 2-rigid, is $L{\sim}2{\cdot}M$. Meanwhile, the augmented kNN-graph has approximately $L{\sim}n_{vi}{\cdot}M$ edges, where $n_{vi}{=}nn{+}rn > 2$ [7]. As our experience shows, the probability that the largest connected component is rigid (or approximately rigid) is very high. Summarizing, to obtain the largest connected component approximately equal to the full kNN-graph, the number of the nearest neighbors nn can be very low (mostly $nn = 2$ but for some specific datasets with very similar samples it can be a bit larger). Assuming additionally that $rn = 1$, we can obtain stable and rigid 2-D embedding of the kNN-graph.

This way, instead of the $O(M^2)$ floating point $\mathbf{D}$ matrix, we have as the input data $O(nn \cdot M)$ integers - the list of kNN-graph edges. The indices of rn random neighbors can be generated *ad hoc* during embedding process. Thus the embedding of high-dimensional data reduces to the embedding of the corresponding sparse kNN-graph. To this end, we minimize the following stress function:

$$E(\|\mathbf{D} - \mathbf{d}\|) = \sum_i \sum_{j \in O_{nn}(i) \cup O_{rn}(i)} \begin{cases} d_{ij}^2 & if \ j \in O_{nn}(i) \\ c \cdot (1 - d_{ij})^2 & if \ j \in O_{rn}(i) \end{cases} , \tag{6}$$

which represents the error between dissimilarities $D_{ij} \in \{0, 1\}$ and corresponding Euclidean distances d_{ij}, where: $i, j = 1, \ldots, M$, and $c \in (0, 1)$ is the scaling factor for random neighbors. Thus, IVHD uses the stress function, which is much easier to optimize than a KL-based loss function employed in the baseline algorithms.

2.3 Optimization

BH-SNE-CUDA To minimize KL divergence, t-SNE and its clones employ the optimal, matched by the Authors, modern gradient descent optimization schemes [21]. The

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

gradient of the loss function $C(.)$ (Eq. 1) is as follows:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij}) q_{ij} (y_i - y_j). \tag{7}$$

AtSNE-CUDA The AtSNE-CUDA algorithm preserves the global information by minimizing the term $KL(P(A)||Q(B))$ in Eq.4, while the first term of the loss function is responsible for preserving the local structure of data. Because, the gradient of the loss function is more complicated than this defined by Eq. 7, AtSNE-CUDA uses the Hierarchical Optimization Algorithm [10]. This algorithm consists of two optimization layers: the global and the local ones. Optimization proceeds in a top-down manner. The main idea is to optimize the two-layer layout alternately: 1) fix the layout of the ordinary points and optimize the layout of the anchor points; 2) fix the anchor point layout and optimize the layout of ordinary points.

IVHD A few state-of-the-art optimization schemes such as: Nesterov, Adagrad, Adadelta, RMSprop, NAG, Adam, [21] and force-directed (F-D) method [7] have been implemented to calculate minimum of the loss function (Eq. 6). As shown in Table 1, where the best GPU implementations of the optimization methods are collected, the force-directed (F-D) approach and synchronized Nestorov scheme appear to be the best in terms of accuracy (i.e., a better minimum of the loss function reached).

Table 1. Results of cf_5 and cf_{10} accuracies (see Eq.8) achieved by IVHD-CUDA implementation for various optimizers and baseline datasets.

		MNIST			**Fashion-MNIST**			**Small NORB**			**RCV**		**YAHOO**	
		1s	3s	5s	1s	3s	5s	1s	3s	5s	15s	25s	15s	25s
F-D	cf_5	0.501	0.852	**0.89**	0.472	0.671	**0.68**	0.87	0.946	**0.954**	0.666	**0.672**	0.562	**0.618**
	cf_{10}	0.499	0.851	**0.889**	0.469	0.667	**0.676**	0.846	0.933	**0.945**	0.666	**0.671**	0.562	**0.618**
Nest.	cf_5	0.522	0.846	**0.888**	0.473	0.671	**0.688**	0.86	0.942	**0.957**	0.432	0.471	0.102	0.252
	cf_{10}	0.52	0.844	**0.887**	0.47	0.667	**0.684**	0.835	0.929	**0.948**	0.432	0.47	0.102	0.252
Adad.	cf_5	0.87	0.868	0.866	0.687	0.679	0.672	0.936	0.925	0.914	0.581	0.626	0.261	0.57
	cf_{10}	0.869	0.867	0.866	0.685	0.676	0.67	0.927	0.916	0.904	0.58	0.625	0.261	0.57
Adam	cf_5	0.64	0.872	0.867	0.551	0.687	0.686	0.835	0.958	0.951	0.37	0.437	0.102	0.145
	cf_{10}	0.638	0.871	0.866	0.548	0.684	0.682	0.807	0.947	0.941	0.369	0.436	0.102	0.145

3 GPU implementation

The advantage of using GPU accelerator is a huge number of computational units that can perform calculations in parallel. In a typical GPU, there are a few thousands of cores. GPUs boards are Single Instruction, Multiple Data (SIMD) computers. It means that multiple threads are executed using the same instruction set on various data. In the case of the Nvidia CUDA framework, it means that all threads inside a warp will be calculated by using the same instruction set. In the case of instructions branching, the execution of two alternatives will be carried out separately and one of the two will be selected after evaluation of the *if* condition. This process is inefficient and if it occurs frequently (*branch divergence*), it might lead to a poor utilization of GPU. That is why the optimization of the code needs to develop special GPU-dedicated algorithms.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

Optimization schemes on GPU On a CPU unit, subsequent iterations for the classical stochastic gradient descent (SGD) based optimization schemes are executed sequentially and asynchronously. Meanwhile, to achieve the best GPU performance, we had to execute the iterations synchronously, such as in the force-directed approach [7], i.e., all the updated variables of the loss function are calculated using values from the previous iteration what helps to avoid race conditions between CUDA threads. It appears that (see Table 1) synchronous version of the Nesterov method produces almost identical results as the F-D scheme.

IVHD-CUDA In IVHD-CUDA we employ the force directed optimization scheme, similar in spirit to the synchronous *momentum* and *Nesterov* methods, described in detail in [7]. Let *connection* mean two "particles" x_i and x_j (the nearest or random neighbors) in 2D embedded X space joined by an edge. The main IVHD-CUDA implementation loop consists of the following steps: 1) calculating "forces" for each *connection* where the "force" is proportional to the gradient dependent on the two "particles'" positions and their velocities; 2) summing up the forces and 3) updating positions and velocities for all the particles simultaneously. The simplified pseudo-code is presented below as Algorithm 1.

Algorithm 1: Simplified IVHD-CUDA scheme.

```
Input: kNN-graph precalculated with FAISS library.
initialization;
while running do
    allocate subsets of force components to threads;
    for thread in threads do
        calculate force components for each connection;
    end
    join threads;
    allocate subsets of samples to threads;
    for thread in threads do
        sum force/gradient components;
        calculate new velocities;
        update particle positions;
    end
end
```

Because the steps are executed sequentially, it allows to avoid the race condition between CUDA threads. Each connection receives two different memory addresses where calculated values can be stored - one *positive* and one *negative*. If a particle i attracts particle j (*positive* value), then at the same time particle j attracts particle i (*negative* value). In the first step, each thread calculates some subset of the inter-particle force/gradient components. Simultaneously, they execute exactly the same instructions and thus, any branching is not required. However, this approach has a minor bottleneck. To save the result for each *connection*, a given thread has to execute *write* instructions

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

to completely different memory addresses representing two samples. Meanwhile, minimizing the number of *read*/*write* operations is always beneficial as they are very time consuming.

In the second step. each thread is assigned with a set of particles to process. As a precaution against processing the particles that interact with different number of particles (what might cause branch divergence) we sort the particles before the first iteration by considering the number of force components to calculate. After sorting it is guaranteed that all the threads in a warp will process samples with the same number of *connections*. The percentage share of each component of DE procedure (without *k*NN graph generation), e.g., for 70.000 feature vectors with approximately three neighbors each (nn=2 and rn=1), is as follows: (1) force components calculation (78.97%), (2) positions update (20.6%), initialization (3) (0.43%). In IVHD-CUDA implementation the global, register and constant memory are used. In the constant memory, the algorithm parameters are stored. While, the *connections*, force components and positions are stored in the global memory. A register memory is used for auxiliary and temporary calculations.

4 Computational environment

Herein, we present: the baseline datasets used in experiments, evaluation metrics and computational environment.

Datasets The main properties of each baseline dataset are: the number of samples M, dimensionality N, and the number of classes K. Here, we consider datasets with (1) a huge number of samples and relatively low dimensionality, (2) a smaller number of samples but larger number of features, (3) highly imbalanced data (RCV-Reuters), and (4) skewed data (Small NORB). Datasets used in the experiments are described in Table2.

Table 2. The list of baseline datasets.

Dataset	N	M	K	Short description
MNIST	784	70 000	10	Well balanced set of grayscale images of handwritten digits.
Fashion-MNIST	784	70 000	10	More difficult MNIST version. Instead of handwritten digits it consists of apparel images.
Small NORB	2048	48 600	5	It contains stereo image pairs of 50 uniform-colored toys under 18 azimuths, 9 elevations, and 6 lighting conditions.
RCV-Reuters	30	804 409	8	Corpus of press articles preprocessed to 30D by PCA.
YAHOO	100	1.4 million	10	Questions and answers from YAHOO. The answers service preprocessed with FastText [20].

Evaluation of embedding quality Because of the high computational load required for computing precision/recall coefficients, to compare data separability and class purity, we define the following simple metrics:

$$cf_{nn} = \frac{\sum_{i=1}^{M} nn(i)}{nn \cdot M} \quad \text{and} \quad cf = \frac{\sum_{nn=1}^{nn_{\max}} cf_{nn}}{nn_{\max}}, \tag{8}$$

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

where $nn(i)$ is the number of the nearest neighbors of x_i in X space, which belong to the same class as y_i. The value of cf is computed for arbitrarily defined value of $nn_{\max}$ dependent on the number of feature vectors in classes. To reflect a wide range of embedding properties, we use $nn_{\max}$=100. The value of $cf \sim 1$ for well separated and pure classes, while $cf \sim 1/K$ for random points from K classes. These simple metrics allow for evaluating the quality of the embeddings by calculating several cf_{nn} values for small, medium, and greater number of nn. The differences in this criterion for confronted methods allow for inferring their embedding qualities for very local (nn=2), local (nn=10) and medium (nn=100) reconstruction depth. The stability of cf_{nn} for increasing nn means more compact and circular shape of classes. For elongated and mixed classes, the values of cf_{nn} decrease faster with nn.

Hardware All GPU/CUDA implementations of the baseline embedding methods were executed on the separated remote server with: CPU Intel Xeon E5-2620 v3, GPU Nvidia GeForce GTX 1070 (1920 CUDA cores, 8GB GDDR5), 252 GB RAM, OS: Ubuntu 18.04.3, architecture x86, 64. The codes were compiled using GCC-7.4 and CUDA Toolkit 10.0. IVHD-CUDA code was tested on the GPU device with capability 2.7 and CUDA toolkit V8.0, and no issues were observed. The source code used in this paper can be found at `https://github.com/mtsznowak/ivhd-cuda`.

5 Results and comparisons

In this section we compare our IVHD-CUDA implementation to the most efficient and robust publicly available GPU-implemented data embedding codes: BH-SNE-CUDA and AtSNE-CUDA. The most important parameters of the methods are collected in Table 3. For the baseline methods we use default parameters (appx. 15 parameters) proposed in [10, 15] and submitted to the GitHub repositories with respective GPU codes [3, 4]. IVHD parameters are also selected by default, e.g., $rn = 1$ while nn is fitted automatically as a minimal value producing the largest connected component which approximates (with a given high accuracy) kNN-graph. The value of $c \in [0.01, 0.1]$ from Eq. 6 is the only parameter, which can be adjusted interactively.

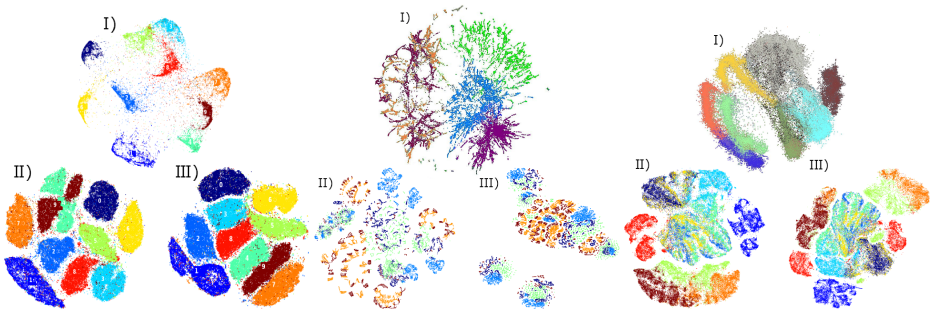
Table 3. The glossary of parameters of DE algorithms used in experiments. All IVHD parameters are displayed. The baseline algorithms require matching a considerably higher amount of parameters (about 10) [4, 3]), so due to the brevity only the most important ones, i.e., *perplexity* and *nn* are given.

Method	Dataset	Perplexity	nn	rn	c	Metric
IVHD-CUDA	MNIST	-	2	1	0.01	Euclidean
IVHD-CUDA	Fashion-MNIST	-	2	1	0.01	Cosine
IVHD-CUDA	Small Norb	-	5	1	0.01	Cosine
IVHD-CUDA	RCV-Reuters	-	3	1	0.1	Cosine
IVHD-CUDA	YAHOO	-	2	1	0.1	Cosine
BH-SNE-CUDA	All	50	32	-	-	Euclidean
AtSNE-CUDA	All	50	100	-	-	Euclidean

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

Based on visualizations presented in Figs. 2, 3 one can make the following observations:

1. In Fig. 2a, IVHD clearly reflects the global structure of the MNIST classes and their separation. However, due to the "crowding effect", the classes become very dense in the centers, and sparse between them. On the other hand, the results of graph visualization presented in [6], demonstrate that the "crowding effect" can be controlled by tuning IVHD-CUDA parameters. Consequently, even the fine-grained neighborhood can be preserved with an amazingly high accuracy (see [8]).
2. For the Small NORB dataset (see Fig.2b), IVHD-CUDA was able to visualize clearly separable three big clusters with fine-grained data structure. For BH-SNE-CUDA and AtSNE-CUDA the quality of embeddings is much worse and more fragmented. The changes of the parameter values of the baseline algorithms (*perplexity*) do not improve the visualization quality.
3. IVHD applied to the Fashion-MNIST (see Fig.2c) creates separate clusters of various, mainly elongated shapes. Moreover, the generated mapping is fuzzy. It is clearly reflected by decreasing values of cf_{nn} particularly for nn=100. On the other hand, both AtSNE-CUDA and BH-SNE-CUDA are much better creating oblate and clearly separated clusters. As shown in Table 4, the values cf_{nn} are more stable than those for the IVHD method. Nevertheless, unlike the IVHD result, some classes reproduced by the baseline methods are mixed and fragmented.
4. Similar conclusions can be drawn by visualizing RCV and YAHOO 2D embeddings (see Fig.3a,3b, respectively). IVHD-CUDA generates more fuzzy output than AtSNE-CUDA but the samples from the same class are closer together than those generated by the baseline method. Meanwhile, AtSNE-CUDA is able to separate clearly visible but fragmented clusters. BH-SNE-CUDA was too slow to visualize these big datasets in a reasonable time budget.



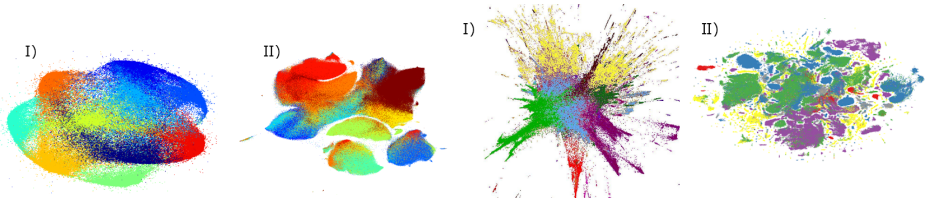
(a) MNIST. (b) Small NORB. (c) Fashion MNIST.

Fig. 2. Visualization of datasets using: I) IVHD-CUDA, II) BH-SNE-CUDA, III) AtSNE.

As shown in Table 4, IVHD-CUDA is the fastest method for all the baseline datasets. Unfortunately, we were not able to control the run-times of AtSNE-CUDA and BH-SNE-CUDA implementations, thus the comparisons for various time budgets are not possible. As shown above, the quality and fidelity of embeddings strongly depends on the structure of a specific dataset, user expectations, and their data visualization requirements. In general, the local structure of the *source* data is better reconstructed by the

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

two baseline SNE algorithms. This is not a surprise because the IVHD algorithm does not use the correct ordering of kNN neighbors and original distances between a sample y_i and its kNN neighbors. Moreover, the value of k (nn in this paper) is extremely small compared to AtSNE-CUDA and BH-SNE. Nevertheless, despite of this drastic approximation, IVHD properly preserves the class structure and its relative locations. For fine-grained structures of classes (such as in the Small NORB dataset), IVHD outperforms its competitors in both the efficiency and - slightly - in the cf accuracy. Moreover, unlike AtSNE-CUDA and BH-SNE-CUDA, IVHD is able to visualize the separated and not fragmented classes. The same can be observed for highly imbalanced dataset RCV. For MNIST and Fashion MNIST datasets, the dominance of t-SNE based methods in reproducing the local - "microscopic" - data structure is visible, mainly due to the strong "crowding effect" seen for IVHD embeddings. However, the "macroscopic" view of the classes is more convincing for IVHD visualization, though due to the high $perplexity$ parameter ($perplexity = 50$), the baseline algorithms are tuned also for better coarse-grained visualization.



(a) YAHOO (b) RCV-Reuters

Fig. 3. Visualization of datasets using: I) IVHD-CUDA, II) AtSNE-CUDA.

Table 4. The values of cf_{nn} accuracy for various datasets and embedding methods. The timings show: the overall embedding time ($time$), kNN-graph generation time ($time_gg$), net embedding time ($time_emb$). The results are the averages over 10 simulations.

Dataset	Algorithm	$time$ [s]	$time_{gg}$ [s]	$time_{emb}$ [s]	cf_2	cf_{10}	cf_{100}
MNIST	BH-SNE-CUDA	32.588	**5.813**	26.775	0.94	0.938	0.933
	AtSNE	15.980		10.167	0.944	**0.943**	**0.938**
	IVHD-CUDA	**7.326**		**1.261**	**0.946**	0.936	0.924
FMNIST	BH-SNE-CUDA	32.913	**6.734**	26.179	0.757	0.755	**0.738**
	AtSNE	17.453		10.719	0.76	**0.757**	0.737
	IVHD-CUDA	**8.177**		**1.443**	**0.767**	0.726	0.670
Small NORB	BH-SNE-CUDA	38.673	**15.517**	23.161	0.944	0.919	0.745
	AtSNE	20.521		5.009	**0.97**	**0.94**	0.73
	IVHD-CUDA	**16.151**		**0.634**	0.936	0.921	**0.828**
RCV-Reuters	BH-SNE-CUDA	-	**45.302**	-	-	-	-
	AtSNE	220.39		175.088	0.82	0.82	**0.818**
	IVHD-CUDA	**60.72**		**15.418**	**0.835**	**0.828**	0.803
YAHOO	BH-SNE-CUDA	-	**52.930**	-	-	-	-
	AtSNE	628.63		575.7	**0.686**	**0.686**	**0.686**
	IVHD-CUDA	**70.12**		**18.930**	0.668	0.662	0.653

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

The main advantage of IVHD is that after storing the kNN-graph in the disc cache, and neglecting the computational time required for its generation, IVHD embedding can be more than one order of magnitude faster than the baseline methods. This allows for a very detailed interactive exploration of multi-scale data structure by employing broad spectrum of parameter values and various versions of the stress function without a need for the kNN-graph recalculation.

6 Conclusions

In this paper, we have compared GPU/CUDA implementations of data embedding algorithms in the context of interactive visualization of large and high-dimensional data. We demonstrate that in comparison to the baseline algorithms and their GPU implementations, IVHD-CUDA data embedding is the fastest and the most storage saving GPU-implemented DE algorithm. We can demonstrate that it allows for immediate response on interactive requests during exploration of large datasets.

Surprisingly, despite radical simplifications, IVHD still properly reconstructs the main structural properties of large ND datasets at the cost of rather minor, and incomparable to the scale of these radical approximations deterioration of embedding quality. In our opinion, it is the most important result of this research, which shows how robust is the "backbone" of high-dimensional data represented by its very sparse (k is small) kNN-graph.

It is interesting, how robustness of this "backbone" is correlated to the complexity of a low-dimensional manifold occupied by data samples and embedded in a very high-dimensional feature space. The algorithms based on t-SNE still assume that Euclidean distances are responsible for the structure of data. But this is not true at all for very complex manifolds resulting from strong feature interdependence. As a result, they too often produce very fragmented visualizations. Just the sparse kNN-graph (small k) is the most appropriate structure, which is able to approximate such manifolds [17]. Therefore, in the future work, we would like to concentrate on the robustness of the IVHD algorithm depending on data complexity and also on the noise and errors in data (e.g. erroneous labels). The method is planned to be tested on really big datasets $M = 10^7+$, too demanding computationally for the SOTA DE algorithms. To this end, the most powerful multi-GPU boards will be used what involves substantial revision of GPU-code. Summarizing, we believe that our method would be very helpful for visualization of truly big and complex data, where low storage and high computational speed of DE algorithm are the crucial issues.

Acknowledgments The research presented in this paper was supported by the funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education. The authors used PL-Grid Infrastructure and computing resources of ACK Cyfronet.

References

1. Amid, E., Warmuth, M.K.: TriMap: Large-scale Dimensionality Reduction Using Triplets. arXiv preprint arXiv:1910.00204 (2019)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24

2. Cofaru, C.: Inverted file system with asymmetric distance computation for billion-scale approximate nearest neighbor search. https://github.com/zgornel/IVFADC.jl/ (2019), [Online; accessed 20-November-2019]
3. Cong Fu, Yonghui Zhang, D.C., Ren, X.: Anchor-t-sne for large-scale and high-dimension vector visualization. (2019), https://github.com/ZJULearning/AtSNE, [Online; accessed 20-November-2019]
4. David M. Chan, Roshan Rao, F.H., Canny, J.F.: Gpu accelerated t-distributed stochastic neighbor embedding. Journal of Parallel and Distributed Computing **131**, 1–13 (2019), https://github.com/CannyLab/tsne-cuda/, [Online; accessed 20-November-2019]
5. Dzwinel, W., Blasiak, J.: Method of particles in visual clustering of multi-dimensional and large data sets. Future Generation Comp. Syst. **15**, 365–379 (1999)
6. Dzwinel, W., Wcisło, R., Czech, W.: ivga: A fast force-directed method for interactive visualization of complex networks. Journal of Computational Science **21**, 448–459 (2017)
7. Dzwinel, W., Wcislo, R., Matwin, S.: 2-d embedding of large and high-dimensional data with minimal memory and computational time requirements. arXiv preprint arXiv:1902.01108 (2019)
8. Dzwinel, W., Wcislo, R., Strzoda, M.: ivga: Visualization of the network of historical events. In: Proceedings of the 1st International Conference on Internet of Things and Machine Learning. ACM (2017)
9. France, S.L., Carroll, J.D.: Two-way multidimensional scaling: A review. IEEE Transactions on Systems, Man, and Cybernetics **41**(5), 644–661 (2011)
10. Fu, C., Zhang, Y., Cai, D., Ren, X.: Atsne: Efficient and robust visualization on gpu through hierarchical optimization. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 176–186 (2019)
11. Hinton, G.E., Roweis, S.T.: Stochastic neighbor embedding. In: Advances in neural information processing systems. 857–864 (2003)
12. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. arXiv preprint arXiv:1702.08734 (2017)
13. Kumari, N., Rupela, A., Gupta, P., Krishnamurthy, B.: Shapevis: High-dimensional data visualization at scale. arXiv preprint arXiv:2001.05166 (2020)
14. Linderman, G., Rachh, M., Hoskins, J.: Efficient algorithms for t-distributed stochastic neighborhood embedding. arXiv preprint arXiv:1712.09005 (2017)
15. van der Maaten, L.: Accelerating t-sne using tree-based algorithms. Journal of Machine Learning Research **15**, 3221–3245 (2014)
16. van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research **9**, 2579–2605 (2008)
17. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426 (2018)
18. Pawliczek, P., Dzwinel, W., Yuen, D.: Visual exploration of data by using multidimensional scaling on multicore cpu, gpu, and mpi cluster. Concurrency and Computation Practice and Experience **3**, 1–21 (2014)
19. Pezzotti, N., Höllt, T., Lelieveldt, B., Eisemann, E., Vilanova, A.: Hierarchical stochastic neighbor embedding. Computer Graphics Forum **35**, 21–30 (2016)
20. Research, F.A.: Library for fast text representation and classification. (2018), [Online; accessed 20-November-2019]
21. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2017)
22. Tang, J., Liu, J., Zhang, M., Mei, Q.: Visualizing large-scale and high-dimensional data. In: Proceedings of the 25th International Conference on World Wide Web. 287–297 (2016)
23. Wang, R., Zhang, X.: Capacity preserving mapping for high-dimensional data visualization. arXiv preprint arXiv:1909.13322 (2019)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50417-5_24