International Journal of Data Science and Analytics
https://doi.org/10.1007/s41060-018-0100-7

REGULAR PAPER

Parallel edge-based visual assessment of cluster tendency on GPU

Tao Meng[1] · **Bo Yuan**[1]

Received: 12 May 2017 / Accepted: 11 December 2017

Abstract
The visual assessment of (cluster) tendency (VAT) algorithm is an effective tool for investigating cluster tendency, which produces an intuitive image of matrix as the representation of complex datasets. The *improved* VAT (iVAT) incorporates a path-based distance metric into VAT to improve its effectiveness on complex-shaped datasets. The efficient formulation of the iVAT algorithm (efiVAT) further reduces the computational complexity of iVAT from $O(N^3)$ to $O(N^2)$. In this paper, we propose eVAT, an edge-based algorithm that can replicate the output of efiVAT but is more efficient and more suitable for parallelism. We also propose a parallel scheme to accelerate eVAT using NVIDIA GPU and CUDA architecture. We show that, on a range of datasets, the GPU-based eVAT features good scalability and can achieve significant speedups.

Keywords Cluster analysis · Cluster tendency · VAT · EfiVAT · GPU

1 Introduction

Cluster analysis is an important task in pattern recognition and data mining. In general, it consists of three steps: (1) assessing the cluster tendency (e.g., how many groups to seek); (2) partitioning the data into groups; (3) validating the clusters discovered [27]. For data that can be directly projected onto a 2D or 3D Euclidean space (e.g., with a scatter plot), direct observation can provide good insight into the appropriate number of clusters. However, for high-dimensional data, or when only the pairwise relationship between objects is available, advanced techniques are necessary.

Visual assessment of (cluster) tendency (VAT) [1] is one of the popular methods widely used to assess the cluster tendency. Given the dissimilarity matrix D of a set of n objects, VAT represents D as an $n \times n$ image $I(D^*)$ where the objects are reordered to reveal the hidden cluster structure as dark blocks along the diagonal of the image. As an extension to VAT, iVAT [29] first transforms D using a graph-theoretic distance metric and the original VAT is applied on the transformed dissimilarity matrix. iVAT images can clearly reveal the number of clusters and their approximate sizes for datasets with highly complex cluster structures. However, the complexity of iVAT is $O(N^3)$ and efiVAT [12] is proposed to reduce the complexity of iVAT. Different from iVAT, efiVAT first applies VAT to the input dissimilarity matrix and then transforms the VAT-reordered dissimilarity data into iVAT images. It is clear that both iVAT and efiVAT are all closely based on the principle of VAT.

VAT typically works well on relatively small datasets (e.g., 500 or fewer objects). However, for datasets of moderate sizes (e.g., 20,000 data points), the computing time of VAT with time complexity $O(N^2)$ may become intolerable. In view of the high computing time of VAT, several extensions such as reVAT [16], bigVAT [17] and sVAT [11] have been proposed. reVAT performs quasi-ordering of the objects based on a threshold parameter and replaces the intensity image with a series of one-dimensional profile graphs. However, the profile graphs are not as interpretable as the images produced by VAT. To address this problem, bigVAT uses the profile graphs to select a sample of objects and displays the quasi-ordered dissimilarity data of the sampled objects as a VAT-like intensity image. However, the resulting image may not be as descriptive as the VAT-ordered image. sVAT selects a sample of (approximately) size n from the full set of objects

This paper is an extension Version of the PAKDD'2017 Long Presentation paper "Parallel Visual Assessment of Cluster Tendency on GPU" [20].

✉ Bo Yuan
yuanb@sz.tsinghua.edu.cn

Tao Meng
zdhmengtao@163.com

[1] Graduate School at Shenzhen, Tsinghua University, Tsinghua Campus, The University Town, Shenzhen 518055, People's Republic of China

Published online: 30 January 2018

$O = \{o_1, o_2, \ldots, o_N\}$ and performs VAT on the sample. The sample is chosen so that it consists of similar cluster structure as the original dataset. However, if the original dataset consists of many clusters, the value of n needs to increase accordingly.

Graphics processing unit (GPU) is an inexpensive, energy-efficient and highly efficient single instruction, multiple-thread (SIMT) parallel computing device, which can be found in many mainstream desktop computers and workstations. In our previous work, we have shown that the computational efficiency of VAT can be greatly improved by exploiting CUDA-enabled GPUs and the parallelism of VAT [20].

In this paper, we propose a novel edge-based VAT (eVAT) algorithm with good potential for parallelism, which can replicate the iVAT image without preprocessing the input matrix as iVAT or post-processing the output matrix of VAT as efiVAT. Its time complexity is similar to VAT, but the memory usage can be reduced by half. To improve the computational efficiency of eVAT, we also implement its GPU version.

This paper is organized as follows. Section 2 gives a brief review of VAT, iVAT, efiVAT as well as GPU computing. Section 3 describes the proposed eVAT algorithm. Section 4 presents the details of the parallel eVAT algorithm based on GPU. The main experimental studies are reported in Sect. 5, focusing on the comparison of CPU-based efiVAT, CPU-based eVAT and GPU-based eVAT. This paper is concluded in Sect. 6 with some discussions on future work.

2 Related work

2.1 VAT

Let $O = \{o_1, o_2, \ldots, o_n\}$ denote n objects in the dataset and D denote a matrix of pairwise dissimilarities between objects each element of which $d_{ij} = d(o_i, o_j)$ is the dissimilarity between objects o_i and o_j, with $0 \leq d_{ij} \leq 1$; $d_{ij} = d_{ji}$; $d_{ii} = 0$, for $1 \leq i, j \leq n$. Let K be the permutation of $1, 2, \ldots, n$ such that $K(i)$ is the index of the ith element in the list. The reordered list is represented as: $\{o_{K(1)}, o_{K(2)}, \ldots, o_{K(n)}\}$. Let P be the permutation matrix with $p_{ij} = 1$ if $j = K(i)$ and 0 otherwise. The matrix D^* for the reordered list is a similarity transform of D by P: $D^* = P^T DP$.

The key idea is to find P so that D^* is as close to a block diagonal form as possible. VAT reorders the row and columns of D using a modified version of Prims minimal spanning tree (MST) algorithm [24] and displays D^* as a gray-scale image. The main difference is that VAT does not form a MST. Instead, it identifies the order in which vertices are added and the initial vertex is selected based on the maximum edge weight in the underlying complete graph [22]. The general procedure of VAT is shown in Algorithm 1, and an example is shown in Fig. 1. Figure 1a shows the scatter plot of 2,000 data points in $2D$. The five visually apparent clusters are reflected by the five distinct dark blocks along the main diagonal in Fig. 1c, which is the VAT image of the data. Compared to the image of D in the original order as shown in Fig. 1b, it is evident that reordering is necessary to reveal the underlying cluster structure of the data.

Algorithm 1 The VAT Algorithm

Input: An $N \times N$ dissimilarity matrix D

1: Set $I = \varnothing$, $J = \{1, 2, \ldots, N\}$ and $K = (0, 0, \ldots, 0)$.
2: Select $(i, j) \in \arg_{p \in I, q \in J} \max\{d_{pq}\}$.
3: Set $K(1) = i$, $I \leftarrow \{i\}$ and $J \leftarrow J - \{i\}$.
4: **for** $t = 2 : N$ **do**
5: Select $(i, j) \in \arg_{p \in I, q \in J} \min\{d_{pq}\}$.
6: Set $K(t) = j$, update $I \leftarrow I \cup \{j\}$ and $J \leftarrow J - \{j\}$.
7: Form the reordered matrix $D^* = [d^*_{ij}] = [d_{K(i)K(j)}]$, for $1 \leq i, j \leq n$.

Output: A gray-scale image $I(D^*)$ with $\max\{d^*_{ij}\}$: white and $\min\{d^*_{ij}\}$: black.

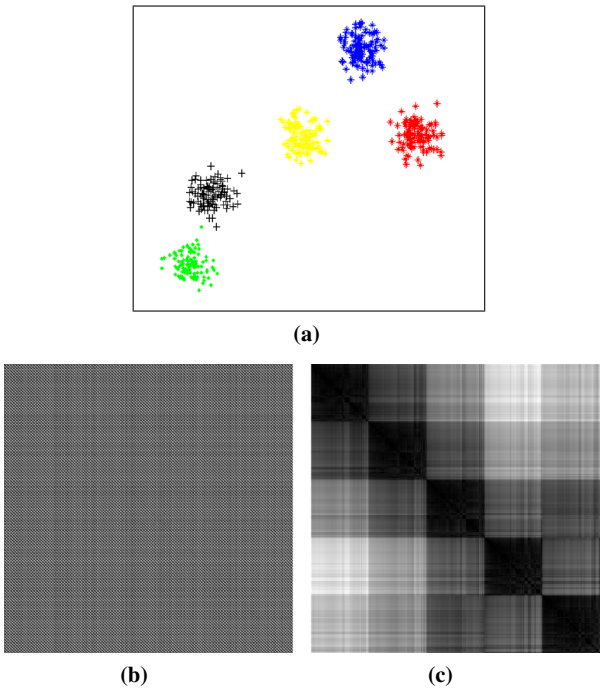

Fig. 1 An example of VAT: **a** the scatter plot of the $2D$ dataset; **b** the original dissimilarity image $I(D)$ and **c** the reordered VAT image $I(D^*)$

2.2 iVAT

For regular-shaped datasets, VAT can often work reasonably well. However, for complex-shaped datasets with irregular geometries, the resulting VAT images may fail to produce dark blocks even when cluster structure is clearly present. Wang et al. [29] proposed an improved VAT (iVAT) algorithm

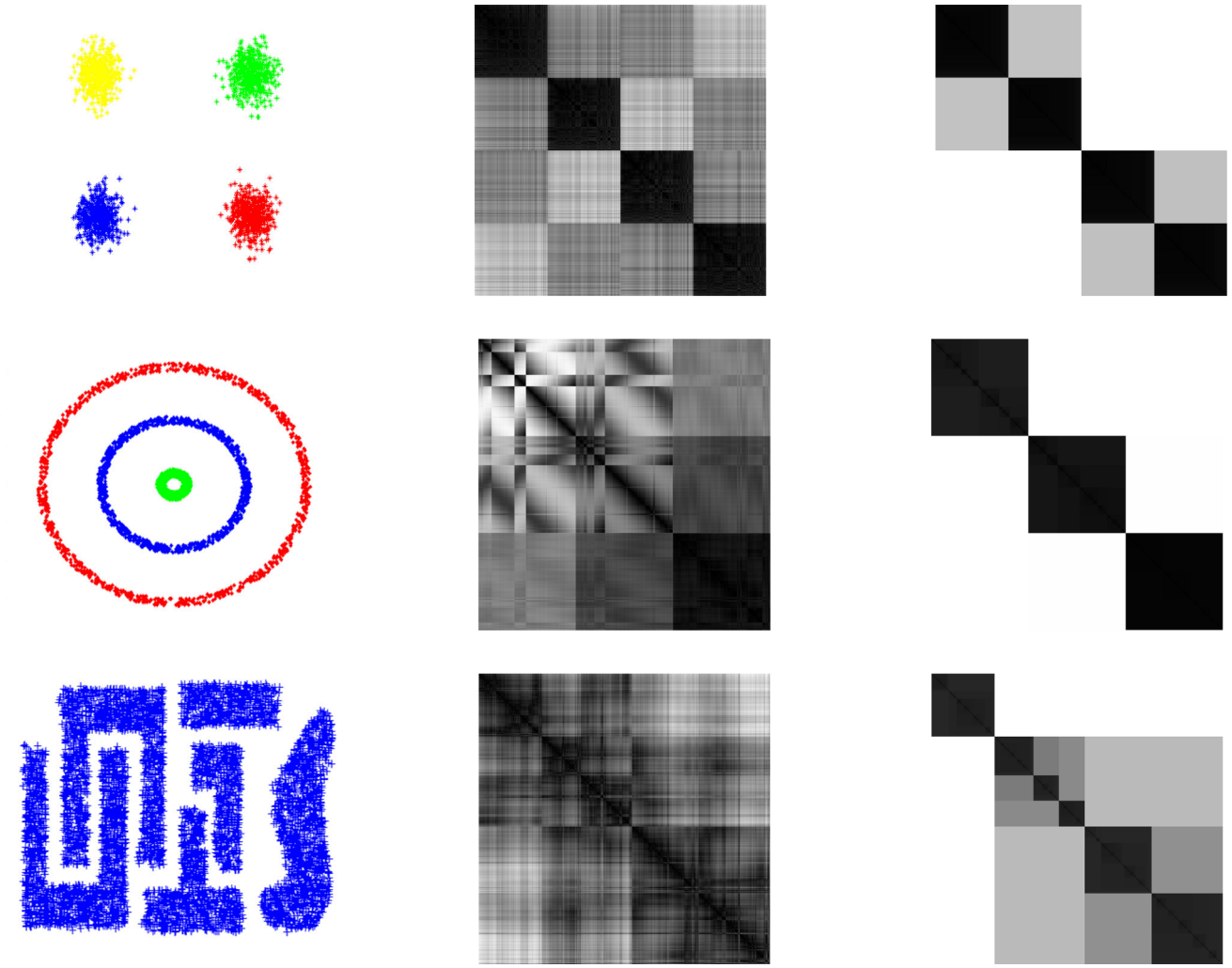

Fig. 2 VAT and iVAT images on complex-shaped datasets: the scatter plot of the 2D dataset (left); the VAT image (middle) and the iVAT image (right)

that uses a path-based distance metric. Suppose D represents the weights of the edges of a fully connected graph. The path-based distance is defined as

$$D'_{ij} = \min_{p \in P_{ij}} \max_{1 \le h < |p|} D_{p[h]p[h+1]}. \tag{1}$$

where $p \in P_{ij}$ is an acyclic path in the set of all acyclic paths between vertex $i(o_i)$ and vertex $j(o_j)$, $p[h]$ is the index of the hth vertex along path p, and $|p|$ is the number of vertices along the path. Hence, $D_{p[h]p[h+1]}$ is the weight of the hth edge along path p. Essentially, the cost of each path p is the maximum weight of its $|p|$ edges. The distance between i and j is the minimum-cost path in P_{ij}. Similar to other algorithms such as the dimensionality reduction method that perform the distance transform as the preprocessing step, iVAT employs a shortest path algorithm to transform the input matrix D into a new matrix D', where the cost is computed by Eq. (1); then the original VAT is applied on matrix D'. iVAT algorithm can not only handle complex-shaped datasets but also produce better quality images on ordinary-shaped datasets than VAT. Figure 2 shows the comparison of VAT and iVAT on both complex-shaped and ordinary-shaped datasets where the effectiveness of iVAT is evident.

2.3 efiVAT

Although iVAT can work well on complex-shaped datasets, the complexity of the preprocessing step, transforming the input matrix D into matrix D', is $O(N^3)$. The reason is that computing D' directly using Eq. (1) can be regarded as a shortest path problem and the Floyd–Warshall algorithm [4] is an algorithm that solves this problem for all N^2 pairs of lowest-cost paths in a connected graph with N nodes, with complexity of $O(N^3)$. Since the complexity of VAT is $O(N^2)$, the total complexity of iVAT is $O(N^3) + O(N^2) = O(N^3)$ [29]. In view of this, Timothy et al. [12] presented an

Algorithm 2 Efficient calculation of iVAT image

Input: D^* - VAT-reordered dissimilarity matrix.
1: $D' = [0]^{N\times N}$.
2: **for** $r = 2 : N$ **do**
3: $\quad j = \arg_{k=1,\ldots,r-1} \min\{D^*_{rk}\}$.
4: $\quad D'_{rc} = D^*_{rc}, c = j$.
5: $\quad D'_{rc} = \max\{D^*_{rj}, D'_{jc}\}, c = 1, , r-1, c \neq j$.
6: D' is symmetric, thus $D'_{cr} = D'_{rc}$.
Output: A gray-scale image $I(D')$ with $\max\{d'_{ij}\}$: white and $\min\{d'_{ij}\}$: black.

efficient formulation of iVAT (efiVAT) algorithm by applying VAT to the original input dissimilarity matrix and then transforming the VAT-reordered dissimilarity data into the iVAT image using the algorithm shown in Algorithm 2. The total number of operations in Algorithm 2 is $(2N^2 - 3N)$, resulting in $O(N^2)$ complexity [12]. Although efiVAT is more efficient than iVAT, according to step 5 in Algorithm 2, it is unfortunate that before computing D'_{rc}, D'_{jc} should be computed, where $0 < j < r$. So, due to this inherent dependence, efiVAT is not suitable for parallel computation.

2.4 GPU high-performance computing

In recent years, GPUs have evolved into highly parallel, multi-threaded, many-core processors and are widely used for general-purpose high-performance computing [5]. Compared with CPU-based distributed systems, GPU-based parallel computing systems are more lightweight, portable and energy efficient. GPUs are well suited to problems that can be represented as data-parallel tasks where the same instruction is executed on massive data elements in parallel. It is also highly desirable that the arithmetic intensity is high, which is the ratio between the number of arithmetic operations and the number of memory operations.

Compute Unified Device Architecture (CUDA) is a general-purpose parallel computing platform and programming model that leverages the parallel computing engine in NVIDIA GPUs to solve challenging computational problems in a more efficient way than CPUs. It was introduced by NVIDIA in November 2006, which significantly reduces the difficulty faced by programmers for developing flexible parallel programs based on NVIDIA GPUs. Threads and kernels are the core concepts in CUDA. Threads are lightweight processes executed on independent processors in GPU, and they are easy to be created and synchronized. Kernels are functions executed on the GPU in parallel by massive threads organized into blocks and grids [3].

There are different types of memory in GPUs, which can significantly affect the performance of GPU programs. Each thread has its private local memory called register, which is the fastest type of memory. Each thread block features shared memory accessible by all threads within the same block, which can be as fast as registers if accessed properly. All threads have access to the same global memory, which is the largest and slowest storage and the only memory visible to CPU. Constant memory and texture memory are two read-only memory spaces accessible by all threads [7]. The global, constant and texture memory spaces are persistent across kernel launches by the same application. In data science, examples of successful GPU applications include matrix multiplication [18], databases [8,9,15], data stream mining [10], FIMI mining [6], subsequence search [25] and GPU-based primitives for database applications [14,15].

3 Edge-based VAT Algorithm

The VAT algorithm consists of three parts: (1) finding the maximum dissimilarity value and the objects involved; (2) generating the new order; (3) reordering the matrix. The proposed edge-based VAT (eVAT) algorithm shown in Algorithm 3 bears some similarity with VAT but features key differences. In VAT, a MST algorithm is applied to generate the new order and reorder the matrix according to the new order. In eVAT, a MST algorithm is applied to sequentially obtain edge weights of the minimal spanning tree and generate a new matrix solely based on the stored edge weights. By doing so, the output of eVAT is already an iVAT image. In short, eVAT has the same complexity as VAT and produces the same result as efiVAT. So eVAT is expected to be more efficient than efiVAT. Furthermore, once the edge weights are obtained, the original input matrix is no longer required and its space can be released, reducing the memory requirement of efiVAT by half.

Algorithm 3 Edge-based VAT Algorithm

Input: An $N \times N$ dissimilarity matrix D
1: Set $K = \{1, 2, \ldots, N\}$, $E = (0, 0, \ldots, 0)$, $J = (0, 0, \ldots, 0)$ and $L(0, 0, \ldots, 0)$.
2: Select $(i, j) \in \arg_{p\in I, q\in J} \max\{d_{pq}\}$.
3: Set $L(k) = \infty$, for $1 \leq k \leq N$.
4: Set $J(i) = 1$ and $E(1) = 0$.
5: **for** $t = 2 : N$ **do**
6: $\quad \arg_{J(k)=0} L(k) = \min\{D_{ik}, L(k)\}$, for $1 \leq k \leq N$.
7: $\quad$ Select $i \in \arg_{J(k)=0} \min\{L(k)\}$.
8: $\quad$ *Set* $J(i) = 1$, and $E(t) = L(i)$.
9: $D' = [0]^{N\times N}$.
10: **for** $r = 2 : N$ **do**
11: $\quad D'_{rc} = \max\{E_r, D'_{[r-1][c]}\}, c = 1, \ldots, r-1$.
12: $\quad D'_{cr} = D'_{rc}$.
Output: A gray-scale image $I(D')$ with $\max\{d'_{ij}\}$: white and $\min\{d'_{ij}\}$: black.

In the next, we prove that the output of eVAT is an iVAT image and is the same as that produced by efiVAT.

In step 4 (Algorithm 2), as D' is symmetric, when $c = j$, there is $D'_{jc} = 0$, and step 4 can be rewritten as $D'_{rc} = \max\{D^*_{rj}, D'_{jc}\}, c = j$. So step 4 and step 5 can be combined as:

$$D'_{rc} = \max\{D^*_{rj}, D'_{jc}\}, c = 1, \ldots, r-1. \tag{2}$$

To prove that the images produced by efiVAT and eVAT are identical, we need to prove:

$$\max\{D^*_{rj}, D'_{jc}\} = \max\{E_r, D'_{[r-1][c]}\}, \quad c = 1, \ldots, r-1. \tag{3}$$

In Eq. (3), E is an array storing the weights of MST edges; E_r is the weight of the $(r-1)$th edge added into the MST according to the Prims algorithm with $E_1 = 0$; r is between 2 and N, where N is the number of rows of D'; j is selected by step 3 in Algorithm 2 and satisfies $0 < j < r$; D' is the resulting matrix and D^* is the VAT-reordered matrix with $D^*_{rj} = E_r$[12]. In order to prove Eq. (3), we just need to show that the following equation holds:

$$\max\{E_r, D'_{jc}\} = \max\{E_r, D'_{[r-1][c]}\}, \quad c = 1, \ldots, r-1. \tag{4}$$

Proof For $r = 2$, as $0 < j < r$, $j = 1$; hence, we have $D'_{[r-1][c]} = D'_{jc}$, and Eq. (4) is true in this case. Assuming the truth of Eq. (4) for some $k > 2$, we obtain the induction hypothesis:

$$D'_{kc} = \max\{E_k, D'_{[k-1][c]}\}, c = 1, \ldots, k-1. \tag{5}$$

According to Eq. (5) , there is:

$$D'_{[k-1][c]} = \max\{E_{k-1}, D'_{[k-2][c]}\}, c = 1, \ldots, k-2. \tag{6}$$

Thus,

$$D'_{kc} = \max\{D'_{jc}, E_{j+1}, \ldots, E_k\}. \tag{7}$$

For $r = k+1$, the analysis is as follows:
For $j = r-1$, it is easy to see that $D'_{[r-1][c]} = D'_{jc}$; hence, Eq. (4) is true.
For $j < r-1$, namely $j < k$, according to Eq. (5) and Eq. (7), there is:

$$\max\{E_{k+1}, D'_{kc}\} = \max\{D'_{jc}, E_{j+1}, \ldots, E_k, E_{k+1}\}, \quad c = 1, \ldots, j. \tag{8}$$

As eVAT incorporates the Prims algorithm that computes the MST of a set of vertices (objects) by sequentially adding the vertex that is closest to the already ordered vertices, E_{k+1} is the weight of the kth edge added into MST. Then:

$$E_{k+1} \geq \max\{E_{j+1}, E_{j+2}, \ldots, E_k\}. \tag{9}$$

Thus,

$$\max\{D'_{jc}, E_{j+1}, \ldots, E_k, E_{k+1}\} = \max\{D'_{jc}, E_{k+1}\}, \quad c = 1, \ldots, j. \tag{10}$$

Hence, when $r = k+1$ and $c = 1, \ldots, j$, Eq. (4) is true. When $r = k+1$ and $c = j+1, \ldots, k$, as D' is symmetric, $D'_{jc} = D'_{cj}$; thus,

$$\max\{E_{k+1}, D'_{jc}\} = \max\{E_{k+1}, D'_{cj}\}. \tag{11}$$

According to Eq. (7):

$$D'_{cj} = \max\{D'_{jj}, E_{j+1}, \ldots, E_c\}. \tag{12}$$

As $D'_{jj} = 0$, according to Eq. (9):

$$\max\{E_{k+1}, D'_{jc}\} = E_{k+1}. \tag{13}$$

From Eqs. (7) and (9), we have:

$$\max\{E_{k+1}, D'_{kc}\} = E_{k+1}. \tag{14}$$

Thus, Eq. (4) is true. □

In the above, we have proved the correctness of the case for $r = k+1$, under the assumption that the case is true for $r = k$. So, Eq. (3) is true and the output of eVAT is the same as the output of efiVAT, which is an iVAT image.

4 GPU-accelerated eVAT

In this section, we present the design and implementation details of the parallel eVAT based on GPU. The eVAT algorithm shown in Algorithm 3 consists of three parts: (1) finding the maximum dissimilarity value and the objects involved; (2) generating edge weight array; (3) regenerating the matrix. Our implementation follows the general workflow of the original algorithm. To make the algorithm more suitable for parallel implementation, we also make some changes.

4.1 Finding the maximum value

The reduction algorithm is a good choice for finding the maximum value of a matrix in GPU. Reduction refers to a class of parallel operations that pass over $O(N)$ input data and generate $O(1)$ result, computed by a binary associative operator

Algorithm 4 eVAT based on GPU

Input: An $N \times N$ dissimilarity matrix D

1: Set $K = \{1, 2, \ldots, N\}$, $E = (0, 0, \ldots, 0)$, $J = (0, 0, \ldots, 0)$ and $L(0, 0, \ldots, 0)$.
2: Select $(i, j) \in \arg_{p \in I, q \in J} \max\{d_{pq}\}$. **//parallel**
3: Set $L(k) = \infty$, for $1 \leq k \leq N$. **//parallel**
4: Set $J(i) = 1$ and $E(1) = 0$.
5: **for** $t = 2 : N$ **do**
6: $\arg_{J(k)=0} L(k) = \min\{D_{ik}, L(k)\}$, for $1 \leq k \leq N$. **//parallel**
7: Select $i \in \arg_{J(k)=0} \min\{L(k)\}$. **//parallel**
8: Set $J(i) = 1$, and $E(t) = L(i)$.
9: $D' = [0]^{N \times N}$.
10: Normalize E to E' as $E'_k \in [0, 255]$,for $1 \leq k \leq N$. **//parallel**
11: **for** $r = 2 : N$ **do //parallel**
12: $D'_{rc} = \max\{E_r, D'_{[r-1][c]}\}, c = 1, \ldots, r-1$.
13: $D'_{cr} = D'_{rc}$.

Output: A gray-scale image $I(D')$ with $\max\{d'_{ij}\}$: white and $\min\{d'_{ij}\}$: black.

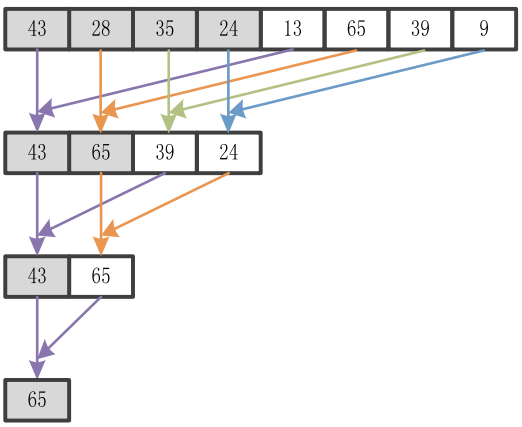

Fig. 3 An example of parallel reduction: finding the maximum value of a vector

$\oplus$. Examples of such operations include minimum, maximum, sum, sum of squares, AND, OR, and the dot product of two vectors. Unless the operator $\oplus$ is extremely expensive to evaluate, reduction tends to be bandwidth-bound [30]. Figure 3 shows an example of parallel reduction that computes the maximum of an 8-element array. There are four threads in use, which are marked in different colors.

Although thrust, a popular library in CUDA, can find the maximum value efficiently, we employ a special reduction method as the index of the object with the maximum value is required. Furthermore, the input matrix itself is symmetric, which means that only half of the matrix needs to be processed. In this paper, we apply the two-pass reduction algorithm [30] to find the maximum value and its index in the matrix. The two-pass reduction operates in two stages, as shown in Fig. 4. A kernel performs *NumBlocks* reductions in parallel, where *NumBlocks* is the number of blocks used to invoke the kernel. Then, the results are stored in an intermediate array. The final result is generated by invoking the same kernel to perform a second pass on the intermediate

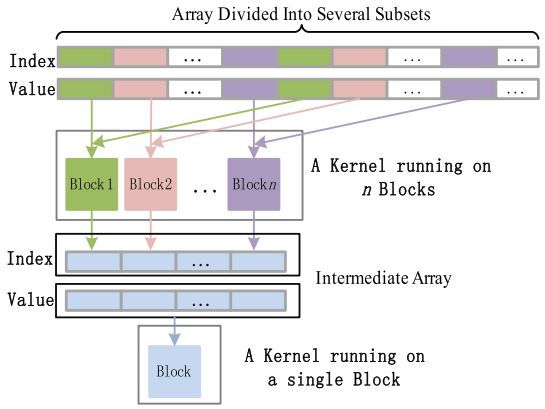

Fig. 4 An example of the two-pass reduction algorithm

array using a single block. Note that, this method imposes no strict requirement on the compute capability of GPUs, making it applicable to a wide range of GPU facilitates.

4.2 Generating edge weight array

Generating edge weight array takes most of the time in eVAT and its degree of parallelism has a significant impact on the overall speedup. It can be divided into two steps: computing the elements in L and finding the minimum value and the corresponding index in L.

Although it features a process of finding the minimum value and the corresponding index, we use the Reduction with Atomics algorithm with only a single kernel, instead of the two-pass reduction algorithm. Note that, invoking a kernel, even an empty kernel that performs no operations, involves a certain amount of overhead. In particular, in this step, the time spent on invoking a kernel is large compared to the time spent on the execution of the kernel. Furthermore, each kernel needs to be launched $N - 1$ times, where N is the width of the input matrix. Consequently, reducing the number of kernels in the algorithm is likely to be beneficial in terms of efficiency.

Similar to the two-pass reduction algorithm, the Reduction with Atomics algorithm stores the result in an intermediate array. The difference is that the Reduction with Atomics algorithm uses a flag value for recording the number of exited blocks. As each block exits, it performs the *atomicAdd* function, a type of atomic operation in CUDA, to check whether it is the block that needs to perform the final reduction. Although the atomic operation does cost some extra time, the Reduction with Atomics algorithm is more efficient than two-pass reduction when the size of data to be processed is small. Figure 5 shows the running times of the Reduction with Atomics algorithm and the two-pass reduction algorithm.

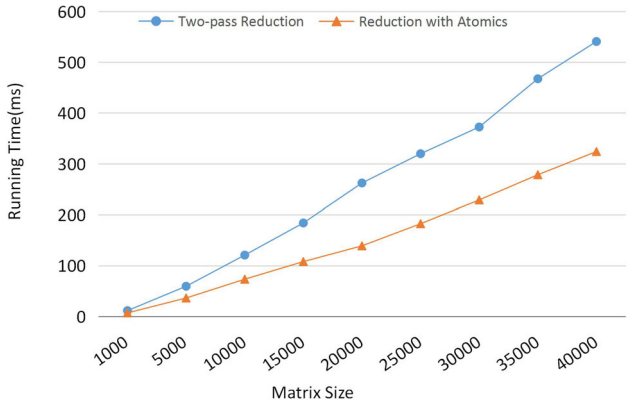

Fig. 5 Running times of the Reduction with Atomics algorithm and the two-pass reduction algorithm on datasets of different sizes

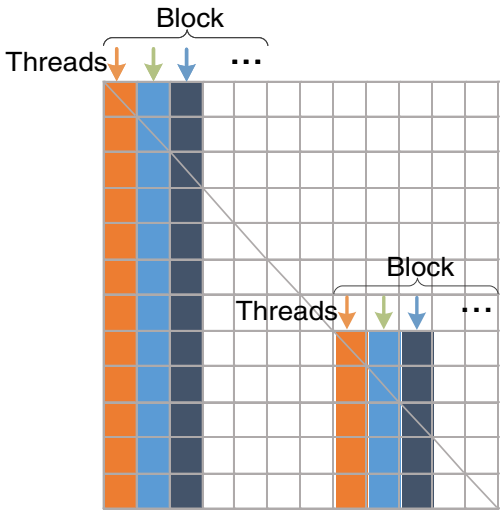

Fig. 6 An illustration of generating D' in parallel

4.3 Regenerating the matrix

The values of D' need to be transformed into $d'_{ij} \in [0, 255]$, to reflect the image density range [0, 255] in openCV [21]. Since D' is generated using E, we normalize E to E' where $E'_k \in [0, 255]$. As the value of d'_{ij} is only associated with $d'_{[i-1][j]}$ and E'_i, it is suited for parallel programming. According to the strategy generating D' in parallel shown in Fig. 6, each thread is in charge of calculating the values in one column recursively. Since D' is symmetric, there is no need to compute the data above the diagonal.

5 Experimental results

We conducted the experiments on a workstation with two Intel Xeon E5-2640 v2 (2.00 GHz, 8 Cores) CPUs, 128 GB RAM and NVIDIA GeForce GTX TITAN X GPU. Powered by NVIDIA Maxwell architecture, the GeForce GTX TITAN X GPU features 3,072 CUDA cores and 12 GB GDDR5 memory. The programming environment was gcc-4.7 with CUDA 7.5 running on Ubuntu 15.04 (64 bit).

5.1 Test datasets

We used a random dataset generator from *scikit-learn* [23]. Four different types of datasets (circles, moons, blobs and random) were generated (Fig. 7) and 10 instances (2D) were created for each type of dataset with 1,000–45,000 objects. We also used a dataset from UCI Machine Learning Repository [19] from which we sampled subsets with different sizes. Since the input of efiVAT is a dissimilarity matrix, once this matrix is given, the efficiency of efiVAT is fully determined, regardless of the dimension of the original dataset.

5.2 Results and analysis

For each dataset, we compared the efficiency of the CPU-based efiVAT, CPU-based eVAT and parallel eVAT base on GPU. For the same data size, our algorithm achieved almost the same speedup rate on different datasets. So, we averaged the results and present the running time and speedup rate.

Figure 8 shows a comparison between CPU-based efiVAT and CPU-based eVAT algorithms. Obviously, eVAT is more efficient than efiVAT on the CPU platform. More specifically, when matrix sizes were relatively small, the speedup rates were around 2.0; when matrix sizes were relatively large, the speedup rates increased to around 3.5. Figure 9 shows

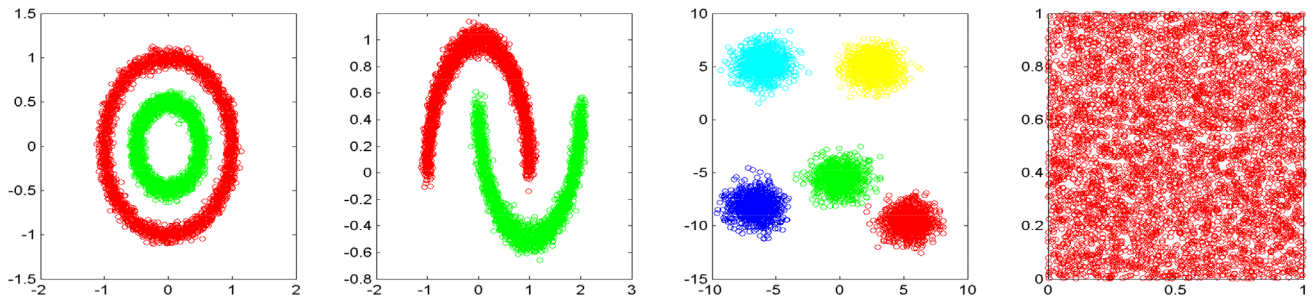

Fig. 7 Four different types of datasets used in the experiments. From left to right: circles, moons, blobs and random

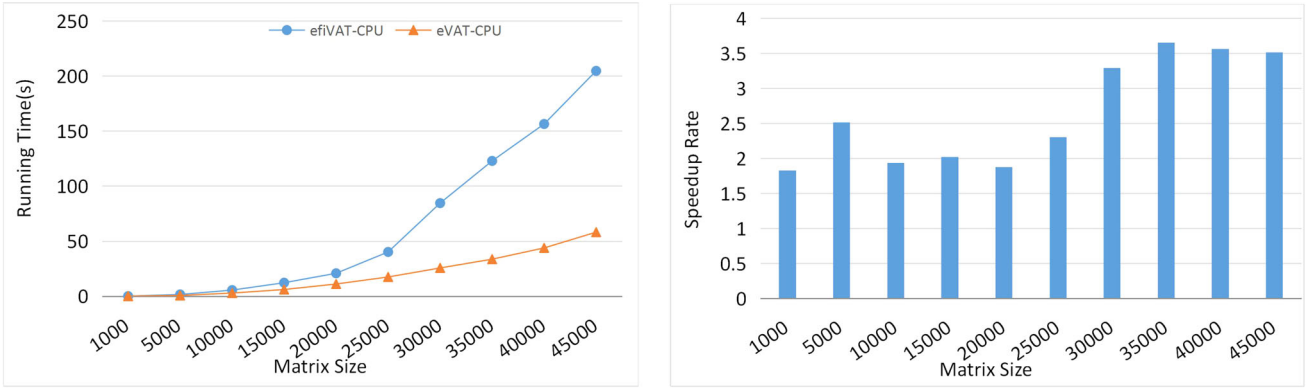

Fig. 8 Average running times (left) and speedup rates (right) of efiVAT-CPU and eVAT-CPU on datasets of different sizes

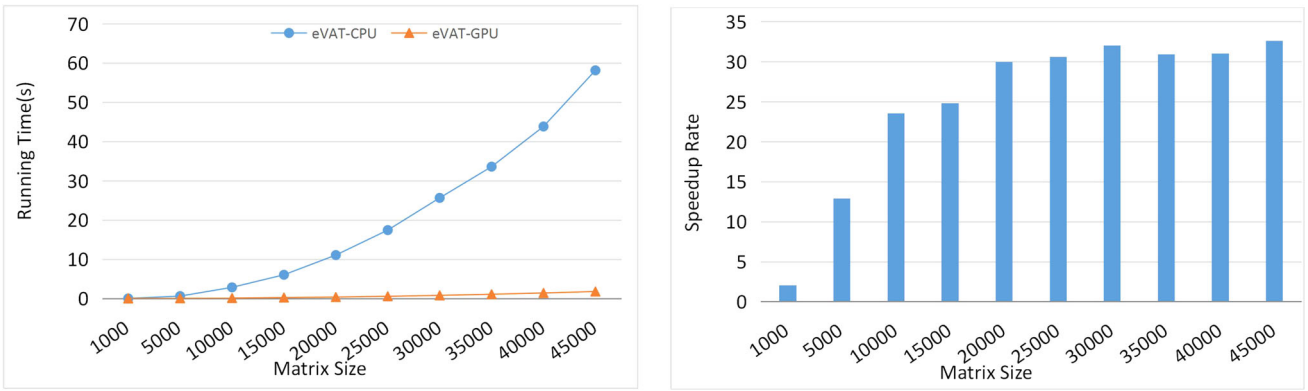

Fig. 9 Average running times (left) and speedup rates (right) of eVAT-CPU and eVAT-GPU on datasets of different sizes

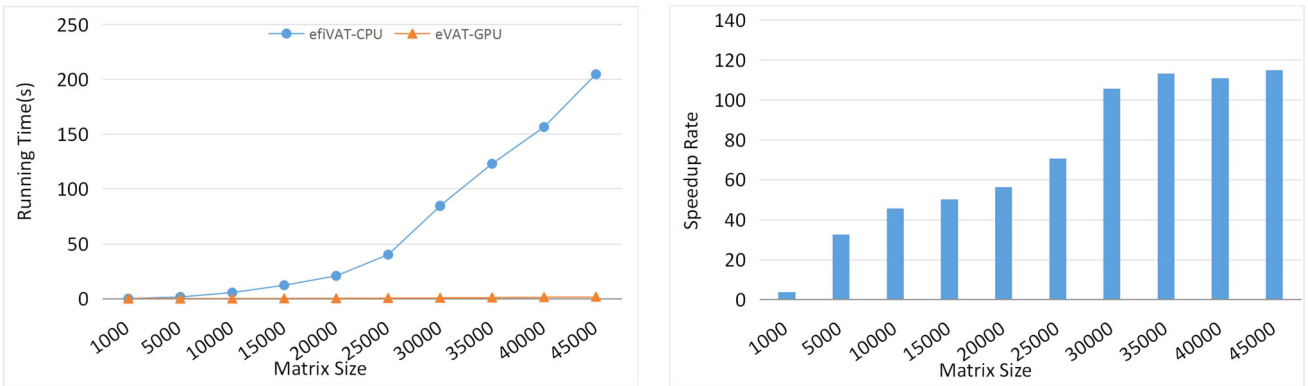

Fig. 10 Average running times (left) and speedup rates (right) of efiVAT-CPU and eVAT-GPU on datasets of different sizes

the comparison between CPU-based eVAT and GPU-based eVAT. It is clear that, the running time of CPU-based eVAT increased rapidly due to its $O(N^2)$ time complexity. Meanwhile, the speedup rate increased steadily as the matrix size increased and reached around 33 for datasets with 45,000 objects. We also compared CPU-based efiVAT and GPU-based eVAT, and the speedup rate reached around 115 for datasets with 45,000 objects as shown in Fig. 10.

6 Conclusion

Visualizing the cluster tendency of datasets is a key step in cluster analysis and is important in both academic research and industrial applications. However, the applicability of efiVAT, one of the most advanced visualization techniques in this domain, has been severely limited by its time complexity. In this paper, we proposed a novel edge-based VAT algorithm, which can produce the same output as efiVAT with higher

efficiency and lower memory usage. Furthermore, eVAT is inherently more suitable for parallel computing compared to efiVAT. We investigated the potential of parallelism of various components in eVAT and designed a GPU-based parallel VAT. Experiments on a variety of test datasets showed that eVAT is more efficient than efiVAT with demonstrated good scalability. The parallel eVAT can also achieve significant speedup rates compared to the CPU versions of eVAT and efiVAT, making it a competent technique for handling large datasets.

In recent years, a number of variations of VAT have been proposed to enhance its capability. For example, Havens et al. [13] performed data clustering in ordered dissimilarity images, and coVAT [2] extends VAT to rectangular dissimilarity data. CCE [26], DBE [28] and aVAT [29] use different schemes to automatically estimate the number of clusters in iVAT images. Most of these VAT-like methods are built on the basic idea of the original VAT and our proposed CPU-based eVAT, and GPU-based parallel eVAT can be potentially extended to these problem domains.

Acknowledgements This work was partially supported by the NVIDIA GPU Education Center awarded to Tsinghua University.

References

1. Bezdek, J.C., Hathaway, R.J.: VAT: a tool for visual assessment of (cluster) tendency. In: International Joint Conference on Neural Networks, pp. 2225–2230 (2002)
2. Bezdek, J.C., Hathaway, R.J., Huband, J.M.: Visual assessment of clustering tendency for rectangular dissimilarity matrices. IEEE Trans. Fuzzy Syst. **15**(5), 890–903 (2007)
3. Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. Elsevier, Hoboken (2012)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
5. CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/index.html
6. Fang, W., Lu, M., Xiao, X., He, B., Luo, Q.: Frequent itemset mining on graphics processors. In: International Workshop on Data Management on New Hardware, Damon 2009, pp. 34–42. Providence June (2009)
7. Farber, R.: CUDA Application Design and Development. Morgan Kaufmann Publishers Inc., Burlington (2011)
8. Govindaraju, N., Gray, J., Kumar, R., Manocha, D.: GPUTeraSort: high performance graphics co-processor sorting for large database management. In: ACM SIGMOD International Conference on Management of Data, pp. 325–336. Chicago June (2006)
9. Govindaraju, N.K., Lloyd, B., Wang, W., Lin, M., Manocha, D.: Fast computation of database operations using graphics processors. In: ACM SIGMOD International Conference on Management of Data, pp. 215–226. Paris June (2004)
10. Govindaraju, N.K., Raghuvanshi, N., Manocha, D.: Fast and approximate stream mining of quantiles and frequencies using graphics processors. In: ACM SIGMOD International Conference on Management of Data, pp. 611–622 (2005)
11. Hathaway, R.J., Bezdek, J.C., Huband, J.M.: Scalable visual assessment of cluster tendency for large data sets. Pattern Recognit. **39**(7), 1315–1324 (2006)
12. Havens, T.C., Bezdek, J.C.: An efficient f of the improved visual assessment of cluster tendency (iVAT) algorithm. IEEE Trans. Knowl. Data Eng. **24**(5), 813–822 (2012)
13. Havens, T.C., Bezdek, J.C., Keller, J.M., Popescu, M.: Clustering in ordered dissimilarity data. Int. J. Intell. Syst. **24**(5), 504–528 (2009)
14. He, B., Govindaraju, N.K., Luo, Q., Smith, B.: Efficient gather and scatter operations on graphics processors. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, p. 46 (2007)
15. He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N., Luo, Q., Sander, P.: Relational joins on graphics processors. In: ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 511–524, Vancouver June (2008)
16. Huband, J.M., Bezdek, J.C., Hathaway, R.J.: Revised visual assessment of (cluster) tendency (reVAT). In: International Conference of the North American Fuzzy Information Processing Society, vol. 1, pp. 101–104 (2004)
17. Huband, J.M., Bezdek, J.C., Hathaway, R.J.: bigVAT: visual assessment of cluster tendency for large data sets. Pattern Recognit. **38**(11), 1875–1886 (2005)
18. Larsen, E.S., Mcallister, D.: Fast matrix multiplies using graphics hardware. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, pp. 43–43 (2001)
19. Lichman, M. UCI Machine Learning Repository. Irvine, University of California, Irvine, School of Information and Computer Sciences. (2013). http://archive.ics.uci.edu/ml
20. Meng, T., Yuan, B.: Parallel visual assessment of cluster tendency on GPU. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 429–440. Springer (2017)
21. OpenCV User Guide. http://docs.opencv.org/2.4.13/doc/user_guide/user_guide.html
22. Pakhira, M.K.: Finding number of clusters before finding clusters. Procedia Technol. **4**(11), 27–37 (2012)
23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**(10), 2825–2830 (2013)
24. Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Tech. J. **36**(6), 1389–1401 (2014)
25. Sart, D., Mueen, A., Najjar, W., Keogh, E., Niennattrakul, V.: Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In: ICDM 2010, The IEEE International Conference on Data Mining, , pp. 14–17. Sydney December (2010)
26. Sledge, I.J., Huband, J.M., Bezdek, J.C.: (Automatic) Cluster count extraction from unlabeled data sets. In: 5th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 3–13 (2008)
27. Wang, L., Geng, X., Bezdek, J., Leckie, C., Kotagiri, R.: SpecVAT: enhanced visual cluster analysis. In: IEEE International Conference on Data Mining, pp. 638–647 (2008)
28. Wang, L., Leckie, C., Ramamohanarao, K., Bezdek, J.: Automatically determining the number of clusters in unlabeled data sets. IEEE Trans. Knowl. Data Eng. **21**(3), 335–350 (2009)
29. Wang, L., Nguyen, U.T., Bezdek, J.C., Leckie, C.A., Ramamohanarao, K.: iVAT and aVAT: enhanced visual analysis for cluster tendency assessment. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 16–27. Springer (2010)
30. Wilt, N.: The CUDA Handbook: A Comprehensive Guide to GPU Programming. Pearson Education, London (2013)