2011 15th International Conference on Information Visualisation

An Interactive Bio-Inspired Approach to Clustering and Visualizing Datasets

Ugo Erra
Dipartimento di Matematica e Informatica
Università della Basilicata
Potenza, Italy
Email: ugo.erra@unibas.it

Bernardino Frola, Vittorio Scarano
Dipartimento di Informatica
Università di Salerno
Fisciano, Italy
Email: frola@dia.unisa.it, vitsca@dia.unisa.it

***Abstract*—In this work, we present an interactive visual clustering approach for the exploration and analysis of datasets using the computational power of Graphics Processor Units (GPUs). The visualization is based on a collective behavioral model that enables cognitive amplification of information visualization. In this way, the workload of understanding the representation of information moves from the cognitive to the perceptual system. The results enable a more intuitive, interactive approach to the discovery of knowledge. The paper illustrates this behavioral model for clustering data, and applies it to the visualization of a number of real and synthetic datasets.**

***Keywords*-visual clustering; behavioral model; GPU; high-dimensional datasets;**

I. Introduction

The human perception system is the result of both evolution and experience of the environment. A direct consequence is the fact that humans have great ability to understand patterns in the natural environment through visual perception. Humans are physiologically receptive to natural shapes and behaviors and such patterns reach deep into our subconscious. For example, humans usually respond in a similar manner to the beauty of a sunset or to the shapes created by a flock of birds. Consequently, visualization of complex interrelationships may be better understood using natural analogues, producing visualizations that are motivated by metaphors inspired by nature. A methodology for the creation of effective visualizations based on our ability to immediately perceive complex information in nature is discussed in [1].

An approach that may be inspired by natural analogues is clustering. Clustering is essentially a data mining approach that addresses the problems of large amounts of data and the scarcity of human attention by discovering groups of similar objects. Each group, called a 'cluster', consists of objects that are similar to one another and dissimilar to objects of other groups. Based on given similarities, data is organized into clusters using an unsupervised learning approach that starts with an unlabeled dataset, from which the aim is to discover how the objects within that set are organized [2]. The main problem of clustering is in the visualization of vast volumes of data which is the first requirement for meaning to emerge and to be understood effectively. Simplistic approaches to visualization lead to cluttered or confusing displays, which require a great deal of cognitive processing on behalf of the user in order to extract meaning from them.

This paper addresses the clustering of large high-dimensional datasets using a bio-inspired visualization technique that improves human understanding. The metaphor is based on a flock of birds. High-dimensional data is mapped as agents' features (we refer to autonomous agents using the word 'agent'). Each agent is assigned a local behavioral model and moves by coordinating its movement with the movement of other agents in a 3D environment. Our approach relies on the natural organization of groups that arises when agents with similar features interact using this local behavioral model.

In addition, we exploit the parallel architecture of Graphics Processor Units (GPUs) to guarantee interactive clustering. We illustrate the model and show how it enables agents to be organized into clusters with similar features. A significant advantage of the proposed approach is that it does not require the number of clusters as input, and data can be introduced interactively. Generally, our approach enables high-performance data analysis processing, and visualization based on an intuitive representation that avoids the projection of high-dimensional data in two- or three-dimensional space. Experimental results show a guaranteed quality of clustering from our algorithm, while implementation using the GPU architecture merely performs well.

The remainder of this paper is organized as follows: in section II, we review previous clustering approaches that are based on GPUs. In section III, we describe the behavioral model that inspired our clustering approach. In section IV, we illustrate our clustering algorithm. In section V, we present a brief description of the application. Section VI illustrates our results in terms of efficiency and performance scalability. Finally, section VII concludes and discusses directions for future work.

II. Related Works

An example of a system that uses visualization techniques for high-dimensional clustering is OPTICS [3]. The authors of OPTICS created a one-dimensional ordering of databases,

1550-6037/11 $26.00 © 2011 IEEE
DOI 10.1109/IV.2011.16

representing the density of clustering structures. Cluster points are close to each other in the one-dimensional ordering generated by OPTICS, and their reachability is found using a distance plot. This visualization system is valuable for understanding and guiding the clustering process. Another approach to high-dimensional clustering is the HD-Eye system [4]. HD-Eye considers clustering as a partitioning problem and enables the user to be directly involved in the clustering process - that is, in choosing the dimensions to be considered, in selecting the clustering paradigms, and in partitioning the datasets.

In the context of clustering, GPUs have demonstrated some interesting results. The k-means clustering algorithm is probably the algorithm most studied on GPUs. The first demonstrations of the use of GPUs to significantly accelerate k-means analysis are [5] [6]. Using an obsolete approach, based on shader languages, the authors of these studies exploit the computational capabilities of GPUs. Today, general purpose languages, like CUDA, offer better support to GPU architectures. The authors of [7] [8] tried to improve the efficiency of k-means using CUDA and optimizations directly targeted at parallel architectures. They obtained an increase in speed that is 14 and 13 times greater, respectively, than that of a CPU's sequential computation.

The authors of [9] used a shader language to implement hierarchical clustering. Their implementation speed was 2-4 times greater than that of a CPU. [10] explored parallel computation of hierarchical clustering with CUDA and obtained a 48-fold increase in speed.

The real-time simulation and visualization of large datasets using a GPU architecture has been proven to outperform CPU implementation in several past papers, for example [11]. In this work, the authors implement the proposed clustering approach using the BehaveRT framework [12]. This framework allows real-time simulation and visualization of large datasets. This enables developers to focus on the design and implementation of behavioral models that exploit the computational power of the GPU. We will show that this is a key aspect to obtaining interactive results.

III. The Behavioral Model

Our clustering approach is inspired by the original behavioral models proposed by Reynolds [13]. In Reynold's model, each agent has a strictly local perception of the space it occupies. None of the group members have full knowledge of the entire group. Hence, agents must base their decisions on what they know of neighbors in their field of vision. Based on each agent's visibility, the synchronized aggregated motion of the group is achieved by calculating a weighted sum of *steering behaviors*. Reynolds defined three steering behaviors.

The first, *separation*, maintains a certain distance from neighbors. This is necessary to prevent collisions. A repulsive force $\vec{f_s}$ is calculated as the difference vector between

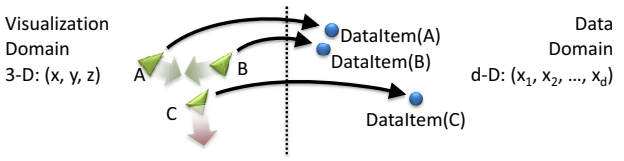


Figure 1: Each agent is associated with a data item in the dataset. Agents move in the 3D space while data items exist in the d-dimensional space. Data items affect the steering force of agents' behavior. For example, agents A and B represent similar data items (using a specific metric) and they move closer to each other. The data item of agent C is quite different to those of agents A and B, thus it moves away from them.

an agent's current position and the position of each of its neighbors, while the steering force is calculated as the average of all the repulsive force vectors. The second, *cohesion* moves the agent toward the center of his local neighborhood. This tends to aggregate the flock. The cohesion force $\vec{f_c}$ is obtained by computing the average position of neighbors. The third steering behavior, *alignment* tends to align the agent with other neighbors through group computing. The alignment force $\vec{f_a}$ is calculated as the difference between the average of the neighbors' forward vectors and the forward vector of the agent itself.

The overall steering force $\vec{f_r}$ of the Reynolds model, for the agent i, is achieved by summing the steering forces produced by all behaviors.

$$\vec{f_r} = w_s\vec{f_s} + w_c\vec{f_c} + w_a\vec{f_a}$$

where, w_s, w_c, and w_a are weights that manage the behavioral impact on the overall steering force.

IV. The Clustering Model Approach

In addition to the behaviors described in the model proposed by Reynolds, we defined two new behaviors called *Cluster-Cohesion* and *Cluster-Alignment*. These behaviors implement the agent-based clustering algorithm.

The cluster-cohesion force $\vec{f_{cc}}$, for a specific agent i, is computed as

$$\vec{f_{cc}} = \sum_{j \in Neighs(i)} sim_{ij}\vec{s_{ij}} + (1 - sim_{ij})\vec{f_{ij}}$$

where, $Neighs(i)$ are the nearest neighbors of the agent i. The vector $\vec{s_{ij}} = (\vec{p_j} - \vec{p_i}) - \vec{v_i}$ is the seeking force between agents i and j, while $\vec{f_{ij}} = -\vec{s_{ij}}$ is the fleeing force. The function sim_{ij} computes a similarity factor between the features vectors associated with agents i and j and must be between 0 and 1. The cluster-alignment force $\vec{f_{ca}}$, for a specific agent i, is computed as

$$\vec{f_{ca}} = \sum_{j \in Neighs(i)} sim_{ij}\vec{d_j}$$

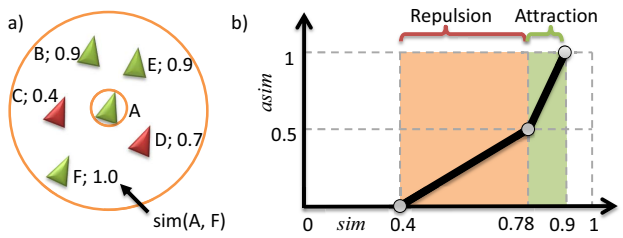


Figure 2: a) A group of 6 agents (triangles). Step I: Agent A computes the linear similarity (sim) of each neighbor. It also computes their minimum (0.4), maximum (0.9) and average (0.78) sim. Step II: Agent A computes the value of $asim$ for each of its neighbors, taking into account the minimum, maximum and average sim computed in the previous step. b) Adaptive vs. linear similarity. Agent A is repulsed by agents with $asim < 0.5$ (agents C and D) and attracted by those with $asim > 0.5$ (agents B, E and F).

The steering force $\vec{f}$ used in the flocking clustering algorithm is calculated by summing Reynold's steering forces and these two new forces

$$\vec{f} = \vec{f_r} + w_{cc}\vec{f_{cc}} + w_{ca}\vec{f_{ca}}$$

It should be noted that in this case, we use two weights w_{cc}, and w_{ca} to manage the impact of the clustering algorithm's behavior.

Similarity.: In our model, each agent represents an object in the data set, while the features vector associated with each object defines an agent's character (Figure 1). Agents move in a 3D environment where the most similar agents will be found and grouped. The overall effect is that when an agent finds another agent similar to itself, it stays near this agent but continues to explore the 3D environment, looking for groups of agents that are similar to its group. The purpose of using a 3D environment as the search space is twofold. First, the 3D environment enables clustering of high-dimensional datasets without feature loss. Second, the clustering process is visualized in an intuitive and natural fashion, irrespective of dataset dimensionality.

The similarity of two agents is computed using the values of their associated features. The implementation described in this paper defines the angular separation between agent i and j as

$$sim_{ij} = \frac{\vec{c}_i \cdot \vec{c}_j}{\sqrt{\|\vec{c}_i\| \, \|\vec{c}_j\|}}$$

where, $\vec{c}$ is the agent's features vector. The range of sim_{ij} is [0, 1] but it must be mapped in the range [-1, 1].To achieve this, the similarity is recalculated as $sim_{ij} = (sim_{ij}+1)/2$.

This similarity factor yields poor results when features vectors are not normalized, which is due to the mean value and variance of all the features vectors. This kind of normalization is unfeasible when data represents continuous streams. For this reason, we adopted a dynamic adjustment

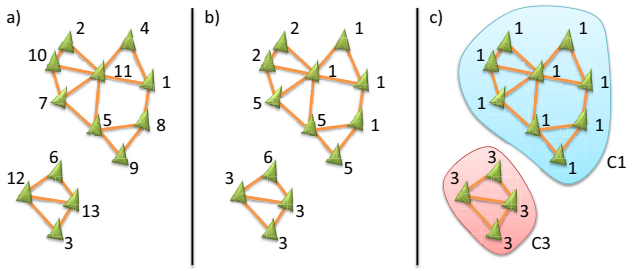


Figure 3: Example of local label propagation for cluster identification (with *LPIterations* equal to 2). a) Step I: Assign a unique label to each agent. b) Step II, iteration 1: Propagate minimum values using neighborhood connections. c) Step II, iteration 2: Again, propagate minimum values. Agents with the same labels represent clusters.

of the similarity value, using the similarities between statistics for agents' neighbors.

At each step of our simulation, each agent collects information about the minimum value (s_{min}), maximum value (s_{max}) and average value (s_{avg}) of its neighbors' similarities. The adaptive similarity $asim$ is then computed as follows:

$$asim_{ij} = \begin{cases} lerp(sim_{ij}, s_{min}, 0.0, s_{avg}, 0.5) & \text{if } sim_{ij} \leq s_{avg} \\ lerp(sim_{ij}, s_{avg}, 0.5, s_{max}, 1.0) & \text{else} \end{cases}$$

where, $lerp(val, x_a, y_a, x_b, y_b)$ represents the linear interpolation of val on the line whose vertexes are (x_a, y_a) and (x_b, y_b). Figure 2 shows the relationship between sim and $asim$.

Cluster Identification.: We implemented a simple local label propagation algorithm for cluster identification. The algorithm consists of two steps:

1) Assign a unique label to each agent.
2) Each agent examines each of its neighbors in turn. If its neighbor's label is smaller than its own label, then it replaces its own label with that of its neighbor. Repeat this step *LPIterations* times.

The value of *LPIterations* can be set at run-time by the user. Figure 3 shows an example of local label propagation.

V. The Application

This model requires that in a large environment, neighbors can be identified - neighbors being all other agents that are within the field of view of a particular agent. This is fundamental because each agent must be able to make decisions based on its neighbors, therefore it must be able to pick out these agents efficiently. In order to guarantee interactive performances in clustering and visualization, we use a framework developed in a previous work called BehaveRT [12]. This framework exploits the computational power of modern GPUs and enables the parallel execution of

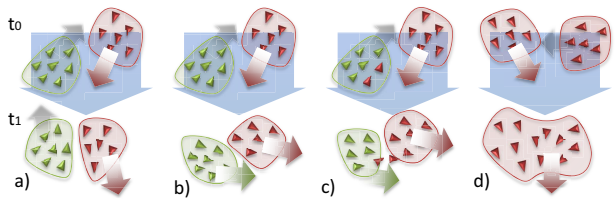


Figure 4: Common experimantal situations were: Collisions between flocks representing well-defined clusters (a). Similar clusters move towards each other (b). Movement of individuals between flocks (c). Flocks mixing (d).

a number of threads equal to the number of simulated agents. In addition, it offers an extensible architecture which enables an efficient implementation of the Reynolds model and of the Cluster-Cohesion and Cluster-Alignment behaviors on the GPU.

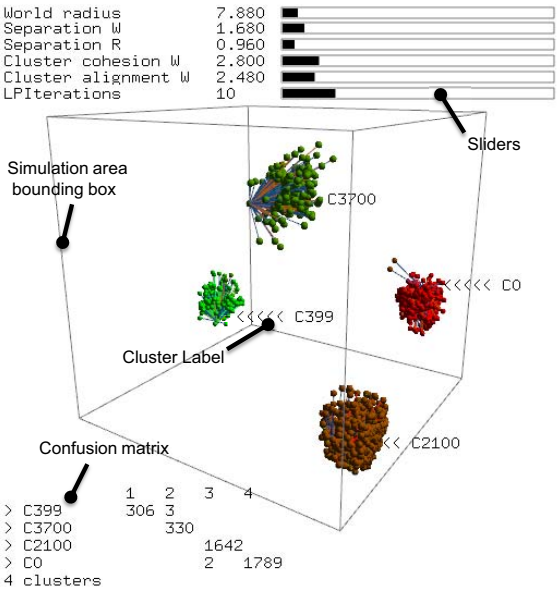
The GPU implementation of the proposed model enables the real-time introduction of agents into the 3D environment, using a sort of agent 'fountain' that eliminates the need to restart the algorithm when new data are available (Figure 5b). The objective is to maintain fast and consistently good clustering of the sequences so far observed. When the data stream of agents enters the environment, it naturally and immediately seeks similar clusters. This feature is implemented by preallocating buffer space in the GPU's memory and using these buffers whenever new data is available.

In the simulation, agents belonging to the same cluster may move together and form flocks. These flocks explore the 3D environment, looking for similar groups to join up with. When flocks (representing well-defined clusters) collide, they bounce off each other and follow different paths (Figure 4a). Flocks representing similar clusters move closer together but do not mix (Figure 4b). Some agents act as cluster bridges, moving between two flocks. These agents change flock membership, depending on whether the cluster of one flock matches its own features vector better than the cluster of its current flock (Figure 4c). Flocks representing the same cluster (according to the similarity function metric) merge into a bigger flock (Figure 4d). Our experiments showed that 2000 iterations were sufficient to reach a stable state, even for flocks consisting of thousands of agents.

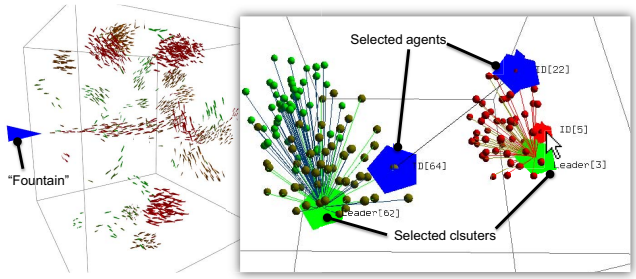
Several parameters influence the formation of clusters. In addition to the weights of the model illustrated in Section IV, we use $worldRadius$, the size of the world; $searchRadius$, the range in question; $separationRadius$, the distance between agents; and $maxNeighbors$, the maximum number of neighbors an agent can have.

The visual interface (Figure 5a) supports the user in the process of classification and verification of output clusters, using the Visual Information Seeking Mantra "Overview, zoom and filter, details-on-demand" [14], described below:

- *Overview.* During cluster creation, the application supports the visualization of the flocking approach. The overview provides the user with a visual summary of clustering results and allows a first evaluation of the number of clusters and relations between clusters. As described in Section IV, each agent is connected to the agent with the lowest unique index in its cluster. The name of the cluster is the label of the lowest unique index in that cluster. While clustering, the user can modify the simulation parameters at run-time, using several sliders. They can also change their point of view in the 3D environment, in order to explore one or more clusters from multiple angles.
- *Zoom and filter.* Because our approach can handle vast volumes of data, the visual interface allows the user



(a) The software Graphical User Interface (GUI)



(b) Agents foutain and Cluster merge action

Figure 5: (a) Each agent is connected to the agent with the lowest label value (leader) amongst the agents that belong to the same cluster. (b) On the left, agents introduced on the fly in any place in a 3D environment. On the right, a screenshot of the merge action. Selected agents and cluster leaders are highlighted. Selected agents are connected with a line.

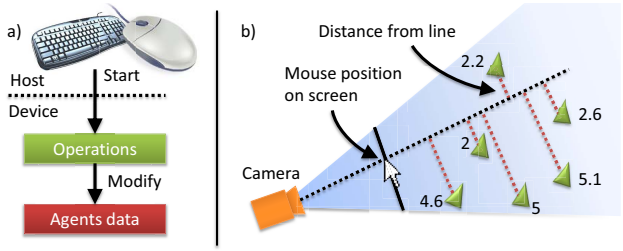


Figure 6: a) Interaction schema: The application forwards input events (generated by the CPU) to the GPU. b) Identification of the nearest agent to the position of the mouse on the screen. The solid line represents the screen. The dotted line indicates the position of the mouse pointer on the screen relative to the camera position. Each agent computes (in parallel) the distance to this line.

to zoom in from the initial overview, and filter information, refining the current view. If the user identifies clusters of interest in the overview, these clusters can be selected individually or removed from the clustering process.

- *Details-on-demand.* The user can select one or more agents and show their properties (position, class membership, etc.). Each input data is labeled with actual class membership, and the application shows detailed information about clusters, using the confusion matrix. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class.

A. User Interaction

Our application allows the user to interactively modify the state of agents at run-time. The purpose of user interaction is to allow the user to improve the quality of the clustering result. Integrating user interaction into our application is not trivial as the data representing the state of agents (positions, directions, etc.) is stored in GPU memory, while the operating system generating input events uses the CPU.

With a large number of agents, we cannot transfer agents' data from GPU to CPU memory, because this operation is too expensive. The solution is to handle input events directly on the device (Figure 6a). This introduces some additional issues because computations are distributed across a number of threads equal to the number of simulated agents. Figure 6 illustrates how to calculate the nearest agent to the mouse position on screen.

The system computes the direction of the line l that starts from the camera position and passes through the 2D position of the mouse on the screen (the dotted line in Figure 6b). Each agent i computes in parallel the distance to l, $dist_i(l)$, that is the perpendicular component of l to the vector $\vec{v_i} = (p_{camera}, p_{agent_i})$ where p_{camera} is the 3D position of the camera and p_{agent_i} is the 3D position of the agent i. $dist_i(l)$

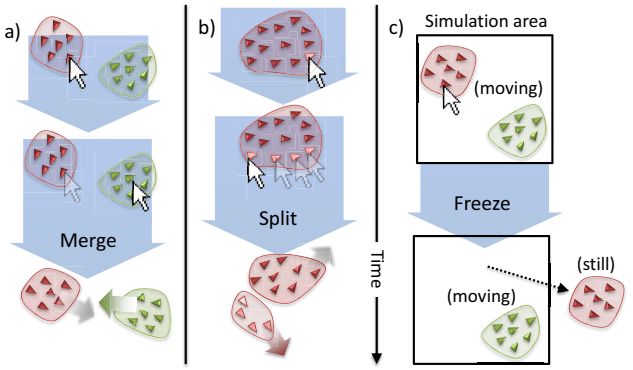


Figure 7: User interaction. The position of the mouse represents a mouse click on an agent. Three actions are possible: Merge (causes collision between clusters), Split (separates agents of the same cluster) and Freeze (moves the selected cluster out of the simulation area).

is the length of the dotted line joining the agent and the line l in Figure 6.

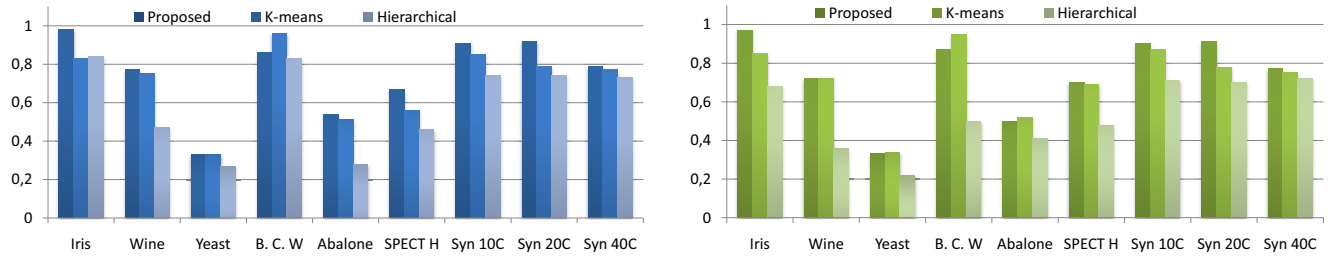
The index of the agent nearest to the mouse position on the screen is equal to the index j such that $dist_j(l) = min_i(dist_i(l))$. The system computes this minimum index performing a parallel reduction on GPU [15] of all the agents' distances in $O(log(n))$ steps.

The application allows the user to interact with agents in several ways. It is composed of two phases: selection and action. During the selection phase the user selects one or more agents by clicking on them with the mouse. Depending on which mode is currently active, the user can select a single agent or a cluster (e.g. when cluster selection is enabled, the user can select a cluster by picking any of its agents). The second phase allows one of the following three operations:

- The *Merge* action (Figure 7a and Figure 5b) causes a collision between two selected clusters. If these two cluster are similar, they merge as shown in Figure 4d.
- The *Split* action (Figure 7b) separates selected cluster agents from all other agents in the same cluster. The two separated clusters will move in opposite directions, and in order to force them to search for similar groups, they will not merged again for a certain number of iterations.
- The *Freeze* action (Figure 7c) moves the selected cluster out of the bounding box. The agents of a frozen cluster are immobilized and cannot interact with other agents.

VI. Experimental Results

In this section, we show the results of two experiments. The first demonstrates the quality of our approach. The second is related to the efficiency of GPU, versus CPU, implementation. All tests were performed on an AMD Athlon 2800+ CPU, 2GB RAM and a NVIDIA GTX 470



(a) Precision

(b) Recall

Figure 8: Quality test results: average values after 500 iterations of (a) precision and (b) recall. The results of the proposed approach are compared with those of k-means and hierarchical clustering.

1280Mb RAM (CUDA compute capability 2.0). Software configuration: CUDA SDK v3.1, Windows 7. Clusters were rendered using OpenGL [16].

Quality.: For the quality tests, we selected six of the most popular datasets from UC Irvine's Machine Learning Repository [17]. The selected datasets are Iris, Wine, Yeast, Breast Cancer Wisconsin, Abalone and SPECT Heart. Below is a brief description of these datasets.

The *Iris* dataset contains information about Iris flowers. There are three classes of Iris flowers - Iris Setosa, Iris Versicolor and Iris Virginica. The Iris dataset consists of 150 examples of Irises that are classified according to 4 attributes. The *Wine* dataset is the result of a chemical analysis of wines grown in a region of Italy but derived from three different cultivars. There are three classes of wines. The dataset consists of 178 examples of wines. The *Yeast* data set contains 1484 records. The data determines the cellular localization sites of proteins. There are ten classes. The *Breast Cancer Wisconsin (B.C.W.)* dataset has 699 records of benign and malignant breast cancer tumors. The goal of this dataset is to explain the difference between the two diagnoses. The *Abalone* (sea snail) dataset has a total of 4177 records. Each record represent an abalone instance. The goal of this dataset is to determine the number of rings using various measurements. The number of rings ranges from 1 to 29. The aim of the Abalone dataset is to divide the number of rings into 3 classes. The *SPECT* Heart dataset has 267 records. In contrast to the other datasets, all of its attributes are binary. The goal of this dataset is to provide a diagnosis using 0 and 1.

In addition, we created three synthetic datasets using the Gaussian cluster generator proposed in [18]. Each contained 4000 records. The first has 10 classes (Synth. 10C), the second has 20 classes (Synth. 20C) and the third 40 classes (Synth. 40C). For each test we split the given dataset into two halves. One was used for training, the other for testing.

The parameters used for quality testing were set to $w_a = 0$, $w_c = 0$, $searchRadius = 4$, $separationRadius = 1.5$, $maxNeighbors = 32$. The training data was used to empirically determine the values of w_s, w_{cc}, and w_{ca} (shown in Table I). The value of $worldRadius$ is caluclated such that agent density in the 3D environment is always 0.05 world units per agent (in order to ensure a good level of interaction among agents).

Table I: Values of parameters

	Iris	Wine	Yeast	B.C.W.	Abalone	SPECT	Synth.
w_s	2.0	3.0	2.0	1.0	2.0	0.5	0.5
w_{cc}	3.0	4.0	4.8	2.0	4.0	1.0	3.0
w_{ca}	2.0	4.0	3.0	4.0	3.8	6.0	2.5

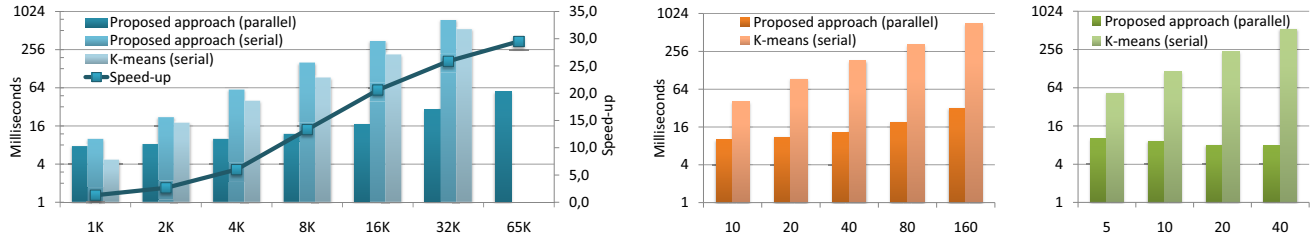
For each dataset, we evaluated the correctness of classification results using $precision$ (P) and $recall$ (R). These measures are defined as:

$$P = \frac{tp}{tp + fp} \qquad R = \frac{tp}{tp + fn}$$

where, tp is the number of true positive patterns, fp the number of false positive patterns, and fn the number of false negative patterns.

Figure 8 shows average values of the proposed clustering algorithm after 500 iterations (these are subsequent to the 2000 iterations necessary to bring the simulation to a stable state). We also compared our results to those of k-means clustering [19] and hierarchical clustering (single-linkage) [20]. The k-means clustering algorithm was executed 500 times for each dataset. For all datasets, results were superior to the those achieved using hierarchical clustering. For Iris, Wine, and SPECT Heart, we achieved better results than with k-means. For Yeast and Abalone, the results were similar and, for Breast Cancer Wisconsin, slightly worse. Tests with the synthetic data show that datasets with high numbers of classes are properly classified.

Performance.: For performance tests, we used Gaussian-based synthetic datasets [18] with different number of instances, features and classes. Parameters are set to $w_s = 0.8$, $w_a = 0$, $w_c = 0$, $w_{cc} = 3.0$, $w_{ca} = 2.5$, $searchRadius = 4$, $separationRadius = 1.5$, $maxNeighbors = 32$, and $worldRadius = 0.05$.



(a) Instances (b) Features (c) Classes

Figure 9: (a) Speed-up is seen between the CPU implementation (proposed approach - serial) and the GPU implementation (proposed approach - parallel). GPU implementation scales better than CPU implementation. GPU implementation is affected by an overhead that dominates overall performance in tests with a low number of instances (up to 1000). (b) The GPU implementation (proposed approach - parallel) does not scale as well as a classical clustering algorithm. This is due to the parallelization scheme chosen. Scalability improves with #instances, compared to #features. (c) The performance of the GPU implementation does not decrease with a high number of classes.

For the evaluation we developed a serial version of the application for the Opteron 252 2.6Ghz CPU with 2GB RAM and based on the OpenSteer steering library [21]. Performance was measured by comparing the number of milliseconds necessary for GPUs and CPUs to implement each algorithm iteration. We also compared the results of our GPU implementation with those of Matlab's k-means serial implementation, in order to have an idea of the results of a classical clustering approach. We executed the k-means clustering algorithm 500 times with each configuration and took the average elapsed time for a single execution.

Figure 9a compares results for GPU, CPU, and Matlab's k-means implementations, using various numbers of instances. With 1000 instances, CPU implementation is more efficient than GPU implementation (due to the data-reordering overhead, as described in [11]), though the latter scales better than the former. We achieved a 30-fold speed-up with a dataset of 65000 agent instances (or size of dataset). Figure 9a also shows that CPU implementation can run up to 2000 instances at interactive frame rates, while GPU implementation can run up to 32000 instances at interactive frame rates.

Figure 9b compares the results of the k-means implentatation with the results of the proposed GPU implementation. The performance of the proposed approach does not scale quite as well as the k-means. This is due the implementation used which launches a new thread for each of the agent's neighbors. This was done to ensure good performance with a high number of instances and a small number of features (up to 40). In future work, a new version of the kernel will address this problem of poor performance. A good solution would be to launch a new thread for each feature of each agent.

Figure 9c illustrates an interesting point. The computation time of the classical k-means implementation increases in proportion to the number of classes. The computation time of the GPU implementation decreases. This is because a large number of classes leads to high agent fragmentation in the 3D environment (one flock for each class). This in turn decreases the average size of the list of agents' neighbors. Thus, when the number of classes is high, the phase of searching for neighbors is slightly more efficient.

VII. Conclusions and Future Works

We proposed a biologically-inspired clustering model for large, high-dimensional datasets using GPUs. Each features vector is represented by an agent. The agent follows the rules developed by Reynolds and two new behaviors (Cluster-Cohesion and Cluster-Alignment) while moving in a 3D environment. Following these simple rules, similar agents gradually merge to form a cluster. GPU implementation is the key to obtaining an interactive visualization as it enables incoming data to cluster without the need to take into account all of the data already processed. Our approach is able to detect evolving input data. It can also detect new data, introduced into the 3D environment which must join old clusters or form new clusters.

Another advantage of our approach is that it does not require *a priori* knowledge of the number of clusters, or of the amount of data that will cluster. As the input data stream evolves during computation, the number of natural clusters changes. This enables the user to interactively introduce data streams into a user-defined 3D space. In addition, we implemented a local label propagation approach to automatically identify clusters. The detection and validation of our results was facilitated by the use of a visualization technique that relies on an interactive interface to improve data interpretation. The approach enables the user to perform several operations on clusters (such as merging, splitting and freezing). Experimental results show that our approach can improve the quality and performance of clustering.

ACKNOWLEDGE

We greatly acknowledge NVIDIA for providing us hardware used during the experiments.

REFERENCES

[1] P. K. Robertson, "A methodology for choosing data representations," *IEEE Comput. Graph. Appl.*, vol. 11, pp. 56–67, May 1991.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS - Ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, 1999.

[4] A. Hinneburg, D. A. Keim, and M. Wawryniuk, "HD-Eye - Visual clustering of high dimensional data: A demonstration," *Data Engineering, International Conference on*, vol. 0, p. 753, 2003.

[5] J. D. Hall and J. C. Hart, "GPU acceleration of iterative clustering," in *ACM Workshop on General Purpose Computing on Graphics Processors*, August 2004.

[6] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," in *DaWaK '08: Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 166–175.

[7] M. Zechner and M. Granitzer, "Accelerating k-means on the graphics processor via CUDA," *Intensive Applications and Services, International Conference on*, vol. 0, pp. 7–15, 2009.

[8] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on GPUs," in *PDPTA*, 2008, pp. 340–345.

[9] Q. Zhang and Y. Zhang, "Hierarchical clustering of gene expression profiles with graphics hardware acceleration," *Pattern Recogn. Lett.*, vol. 27, no. 6, pp. 676–681, 2006.

[10] D.-J. Chang, M. M. Kantardzic, and M. Ouyang, "Hierarchical clustering with CUDA/GPU." in *ISCA PDCCS*, J. H. Graham and A. Skjellum, Eds. ISCA, 2009, pp. 7–12.

[11] U. Erra, B. Frola, V. Scarano, and I. Couzin, "An efficient GPU implementation for large scale individual-based simulation of collective behavior," *High Performance Computational Systems Biology, International Workshop on*, vol. 0, pp. 51–58, 2009.

[12] U. Erra, B. Frola, and V. Scarano, "BehaveRT: A GPU-based library for autonomous characters," in *Motion in Games*, ser. Lecture Notes in Computer Science, R. Boulic, Y. Chrysanthou, and T. Komura, Eds., vol. 6459. Springer Berlin Heidelberg, 2010, pp. 194–205.

[13] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1987, pp. 25–34.

[14] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–.

[15] M. Pharr and R. Fernando, *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

[16] OpenGL ARB, D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005.

[17] http://archive.ics.uci.edu/ml/datasets.html.

[18] http://dbkgroup.org/handl/generators/.

[19] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.

[20] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[21] C. W. Reynolds, "OpenSteer - steering behaviors for autonomous characters," 2004, http://opensteer.sourceforge.net/.