JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 26, Number 5, 2019
© Mary Ann Liebert, Inc.
Pp. 442–456
DOI: 10.1089/cmb.2018.0245

# A Fast Parallel K-Modes Algorithm for Clustering Nucleotide Sequences to Predict Translation Initiation Sites

GUILHERME TORRES CASTRO, LUIS ENRIQUE ZÁRATE,
CRISTIANE NERI NOBRE, and HENRIQUE COTA FREITAS

## ABSTRACT

**Predicting the location of the translation initiation sites (TIS) is an important problem of molecular biology. In this field, the computational cost for balancing non-TIS sequences is substantial and demands high-performance computing. In this article, we present an optimized version of the *K-modes* algorithm to cluster TIS sequences and a comparison with the standard *K-means* clustering. The adapted algorithm uses simple instructions and fewer computational resources to deliver a significant speedup without compromising the sequence clustering results. We also implemented two optimized parallel versions of the algorithm, one for graphics processing units (GPUs) and the other one for general-purpose multicore processors. In our experiments, the GPU *K-modes*'s performance was up to 203 times faster than the respective sequential version for processing *Arabidopsis thaliana* sequence.**

**Keywords:** clustering, *K-means*, *K-modes*, nucleotide sequences, parallel computing, translation initiation site.

## 1. INTRODUCTION

Several research problems required the use of techniques and algorithms that need substantial computational effort. A field with many such problems is Bioinformatics, a multidisciplinary area that employs statistical, mathematical, and computational techniques to solve biology problems. In molecular biology, one problem of great interest that has a high computational effort demand is the prediction of translation initiation sites (TIS) in mRNA sequences. In eukaryote cells, the start of protein translation happens when the ribosome is connected to the mRNA and finds an AUG codon (methionine) with a suitable context for the start of the translation process. So, only a part of the mRNA transcript contains the information that will be translated into proteins. This region is called the CoDing Sequence. When the AUG codon is found in an appropriate context, the ribosome starts to express the codons in the protein by a peptide chain. The protein translation process is finalized when a stop codon (UAG, UAA, or UGA) is found. According to Kozak (1984), the translation usually starts at the first AUG codon of the mRNA sequence. However, the start of a translation can also appear in a different codon (Hatzigeorgiou, 2002). Thus, this process of translation initiation identification is a complex task.

Department of Computer Science, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Brazil.

In this context, machine learning techniques have been employed on the TIS identification task. This is performed by constructing classification or prediction models considering TIS (AUG that starts translation) and non-TIS sequences (AUG that does not start translation) as instances in a data set. Then, the created classification model estimates whether an AUG has the appropriate context to start the translation of the protein. In TIS identification problems, the learning process faces an inherent class imbalance issue of TIS and non-TIS sequences. This happens because, in most cases, each mRNA molecule contains only one AUG that synthesizes protein (TIS). Normally, all other AUGs are not translation initializers (non-TIS). This scenario causes problems in the use of several machine learning algorithms, which are significantly affected by class imbalance.

In general, to address the class imbalancing problem, two approaches are known: *oversampling*, which is based on replicating instances of the minority class, and *undersampling*, wherein the balancing is achieved by eliminating instances of the majority class. This instance replication or elimination can occur randomly or by applying heuristics. The latter is more indicated because replicating data can cause overfitting in classifiers, and removing significant instances can reduce their predictive accuracy, so some care should be employed when performing instance undersampling and oversampling (Batista et al., 2004). To address this problem, Silva et al. (2011) and Rodrigues et al. (2012) have proposed a heuristic undersampling approach based on clustering technique. The authors considered for training the TIS and non-TIS sequences, extracted with 10 and 30 nucleotides in the *upstream* (before AUG codon) and *downstream* (after AUG codon) regions, respectively, along with other conservative features extracted from molecule. Their results showed that the approaches achieved more accurate classifications. However, the methodology adopted requires the use of the *K-means* clustering algorithm for the balancing problem, which did not have an efficient computational performance.

The balancing technique proposed by Silva et al. (2011) and Rodrigues et al. (2012) searches a subset from majority class containing the most representative sequences. The approaches employ the *K-means* algorithm to cluster the majority class, where the number of clusters (*K*) corresponds to the number of sequences of the minority class. To speed up the process, Rodrigues et al. (2012) presented a parallel and distributed solution of *K-means* algorithm using the MPI (message passing interface) and OpenMP (Open Multi-Processing) libraries, whereas Silva et al. (2011) used the sequential version of the algorithm. Although the algorithm presented by Rodrigues et al. (2012) has reached a substantial speedup when compared with the sequential algorithm, the execution time is still considered high. Thus, this work presents an alternative coding using the *K-modes* algorithm to solve the imbalance problem in the TIS context with improved computational efficiency. In accordance with Silva et al. (2011) and Rodrigues et al. (2012), who proposed an undersampling technique, this work proposes efficient clustering algorithms to support the class imbalancing problem.

Therefore, this article presents an optimized version of the *K-modes* clustering algorithm, employed to balance a data set of TIS and non-TIS nucleotide sequences. In addition, we evaluate and compare our results with those obtained by the *K-means* algorithm, used by Rodrigues et al. (2012). Our proposal makes uses of simple instructions and fewer computational resources to get a significant speedup. To improve the performance, we also implemented two parallel versions of the *K-modes* algorithm for graphics processing units (GPUs) and multicore computer clusters. Therefore, our contributions are the following:

- An adapted *K-modes* algorithm to cluster sequences in the context of TIS prediction. For this adapted version, we can highlight: the use of Hamming distance, a binary representation for sequences, and reduction of operations by removing an inner loop.
- Parallel *K-modes* versions[1] for multicore computer clusters and GPU computing.
- A computing performance comparison between the *K-modes* and *K-means* algorithms for the TIS prediction context.

This article is divided into the following sections: Section 2 presents the theoretical foundation of the TIS prediction problem and related work on applications of *K-means* and *K-modes* algorithms. Section 3 presents the parallel techniques used in the clustering algorithms. Sections 4 and 5 present methods and results, respectively, and Section 6 presents our conclusions.

[1] All codes are publicly available at https://github.com/cart-pucminas/HPC-Bioinformatics-Kmodes

## 2. BACKGROUND AND RELATED WORK

### *2.1. Translation initiation site*

The scanning model in eukaryotes starts the link between the ribosome and the mRNA on the 5′ region, reading the codons in the direction of the 3′ region, for translation. The following concepts and definitions about the translation process are relevant to this study: (1) The *upstream* region (nucleotides before the start codon), (2) the *downstream* region (nucleotides after the start codon), and (3) the phase of mRNA molecule reading. The model is shown in Figure 1.

Kozak (1984) identified some conservative positions relative to the TIS. The position –3 presents a purine, where the nucleotide is either an adenine (A) or a guanine (G) in 79% of the cases. In the positions –1, –2, –4, and –5, the nucleotide C (cytosine) is more common. These conservative positions are called the Kozak consensus.

### *2.2. Parallel and distributed algorithms*

In data mining, the *K-means* is one of the most widely used clustering algorithms. Its standard implementation cannot be applied to categorical data because of the distance function used (Euclidean) and the use of the mean to represent the centroids of the groups.

The *K-modes* algorithm modifies the standard *K-means* to enable categorical data clustering, replacing the Euclidean distance function with a simple measure of similarity and using *mode* instead of the *mean* to represent the centers of each group. The iterative clustering process is not essentially altered, thus the efficiency of the clustering is maintained, ensuring convergence to a local minimum.

The *K-means* algorithm is widely used in clustering applications for Bioinformatics problems. For instance, similarity in protein models (Jamroz and Kolinski, 2013), gene prediction (Liu et al., 2013; Díaz et al., 2014), and validation of gene expression in microarrays (Dysvik and Jonassen, 2001; Yeung et al., 2001). These problem domains are characterized by the use of large volumes of data. In this context, parallel implementations of the algorithms become mandatory to achieve high-performance computing. The methodology discussed in the study by Rodrigues et al. (2012) is based on a hybrid approach focused on shared-memory (OpenMP programming) and message-passing (MPI programming) models. However, most successful techniques in the modern implementations of the parallel *K-means* are using GPUs (Wu et al., 2009).

In our work, we present two parallel versions of the *K-modes* algorithm: one using CUDA (Compute Unified Device Architecture) for GPUs and other using MPI/OpenMP for clusters of parallel computers. CUDA is an application programming interface (API) and platform that enables the execution of parallel codes in Nvida GPUs (Ryoo et al., 2008). Despite the fact that the code executed in parallel inside the GPU is ran using shared memory, the data have to transfer between the host and the device (GPU) and vice versa.

The MPI is a message-passing application programmer interface with protocols and semantic specifications for how its features must behave in any implementation (Gropp et al., 1996). Although currently the MPI specifications and some of its implementations offer support to shared memory, MPI programs are frequently written using distributed memory, in which some implementations use shared memory for message transfer if it is available. The OpenMP API uses an ensemble of directives, and the compiler generates parallel code based on those directives. OpenMP applications use shared memory for exchanging data (Dagum and Menon, 1998).

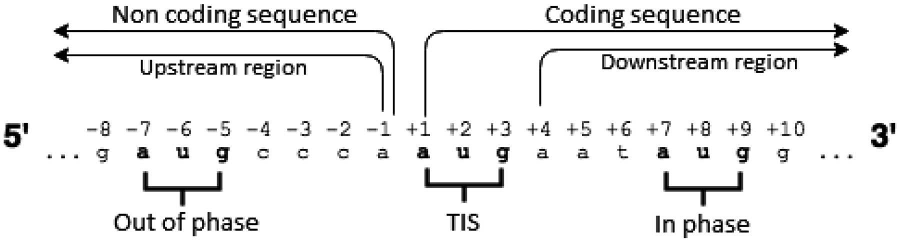


**FIG. 1.** Scanning of mRNA molecule of eukaryotic cell.

### *2.3. Related work*

The clustering process often involves a large volume of data. Due to that and the *K-means* algorithm's popularity, several studies proposed parallel versions for that algorithm, with some implementations using technologies available in the market since the 90s (Kantabutra and Couch, 2000) and others employing more recent technologies (Kang and Park, 2015; Saffran et al., 2017; Baydoun et al., 2018; Souza et al., 2018). Although fewer studies work with the *K-modes* algorithm, their implementations are very similar, but different in the metrics used for distance calculations and in the centroid updating step.

Even recent framework such as the Hadoop has already been employed in a *K-means* implementation. Kang and Park (2015) used *K-means* implementation in Hadoop to group 128,000 instances in 64 clusters, and the algorithm execution took 112 seconds, contrasted with the 1618 seconds of the sequential algorithm. Similarly, the *K-modes* implementation in Hadoop in Tao et al. (2015) reached a six times speedup when compared with the sequential algorithm.

Despite parallel implementations achieving satisfactory results using the Hadoop technology, the greatest speedups found in the recent literature make use of GPUs. In these implementations, CUDA has been frequently used. One of the first such studies, Hong-tao et al. (2009), achieved a 40 times speedup when compared with the sequential implementation of the algorithm. When comparing a GPU implementation to an already optimized parallel *K-means* implementation for central processing unit (CPU), Wu et al. (2009) achieved 10 times faster execution to cluster a billion of instances. In another study, Li et al. (2010) used a GPU cluster capable of optimizing memory access in graphic units, reducing access to the global memory. They reached a six times speedup clustering 51,200 instances compared with CPU processing.

Unlike other recent studies on GPU implementations, Dhanasekaran and Rubin (2011) have developed a *K-means* implementation using the OpenCL (Open Computing Language) library. They used an ATI Radeon® HD 5870 GPU and obtained a 35 times speedup when compared with an OpenMP algorithm version. In the same way, other researches (Cong et al., 2018; Souza et al., 2018; Xu et al., 2018) focus on OpenCL to run *K-means* on GPUs and field programmable gate arrays (FPGAs). Due to the reconfigurable characteristics, FPGA devices present a high speedup, such as the results shown by Xu et al. (2018) that achieved up to 7.76 times more performance than a software version.

Although the MPI library being older than CUDA and OpenMP, it is still widely studied due to its robustness and reliability, often along with other libraries such as the OpenMP (Saffran et al., 2017). For instance, an older work proposed by Kantabutra and Couch (2000) showed that a simple master-slave implementation of *K-means* algorithm ran 11 times faster than its original implementation.

Rao et al. (2009) presented a parallel *K-means* implementation using only OpenMP, and it achieved a 50% reduction in execution time compared with a sequential version. The hybrid MPI/OpenMP *K-means*

Table 1. Related Work Implementations

| *Related work* | *OpenMP* | *MPI* | *MPI/OpenMP* | *CUDA* | *OpenCL* | *OmpSs* | *Hadoop* |
|---|---|---|---|---|---|---|---|
| *K-means* (Kantabutra and Couch, 2000) | | X | | | | | |
| *K-means* (Hong-tao et al., 2009) | | | | X | | | |
| *K-means* (Wu et al., 2009) | | | | X | | | |
| *K-means* (Rao et al., 2009) | X | | | | | | |
| *K-means* (Li et al., 2010) | | | | X | | | |
| *K-means* (Waghmare and Kulkarni, 2010) | X | X | X | | | | |
| *K-means* (Dhanasekaran and Rubin, 2011) | X | | | | X | | |
| *K-means* (Rodrigues et al., 2012) | X | | X | | | | |
| *K-means* (Kang and Park, 2015) | | | | | | | X |
| *K-modes* (Tao et al., 2015) | | | | | | | X |
| *K-means* (Saffran et al., 2017) | X | | X | | | | |
| *K-means* (Souza et al., 2018) | | | | | X | | |
| *K-means* (Baydoun et al., 2018) | X | | | X | | | |
| *K-means* (Cong et al., 2018) | | | | X | X | | |
| *K-means* (Xu et al., 2018) | | | | | | X | |
| *K-modes*—our proposal | | | X | X | | | |

CUDA, Compute Unified Device Architecture; MPI, message passing interface; OpenCL, Open Computing Language; OpenMP, Open Multi-Processing.

implementation presented by Waghmare and Kulkarni (2010) was superior to find the Convex Hull Scan of a data set in comparison to nonhybrid versions that used either MPI or OpenMP individually.

Similarly, Rodrigues et al. (2012) proposed a hybrid version of the *K-means* algorithm for mRNA sequence clustering. Their experiments showed a speedup of up to 23 times when compared with the original sequential implementation. Unlike most studies involving parallel algorithm implementations, the authors worked with a real data set, collecting their data from the RefSeq data repository (Pruitt and Maglott, 2001).

Table 1 presents the cited studies in chronological order, in function of their approach, and includes our own proposal for comparison. As shown in this session, many different approaches can be used to reduce execution times and clustering quality. Our study aims to improve the work proposed by Rodrigues et al. (2012) replicating their experiments in a similar environment and compare their results with *K-modes* implementations in different parallel architectures: CUDA for GPUs and hybrid MPI/OpenMP for computer clusters. Moreover, it is important to notice that our work is the only one to propose *K-modes* to cluster sequences in the context of TIS prediction improving the performance considerably.

## 3. PROPOSAL OF THE K-MODES PARALLEL ALGORITHM

### *3.1. Computational cost*

Both *K-means* and *K-modes* can be defined as shown in Algorithm 1. The difference between them is expressed in lines 6 and 12, where the distance between the instances is calculated using the Hamming distance (line 6) instead of the Euclidean distance. The centroid position is updated based on the mode instead of the mean (line 12), where *tmpC* is an auxiliary struct (an array of integers) that counts the number of active bits in each position of the centroid. To reduce the computational cost in the distance calculation (line 6), we used the hamming distance to measure the dissimilarity between the mRNA sequences. Additionally, the sequences were codified in binary using the method described by Stormo et al. (1982), where A = 1000, C = 0100, G = 0010, and U = 0001. Table 2 shows the binary representation of the ACG and AUG sequences and how the distance between them is calculated.

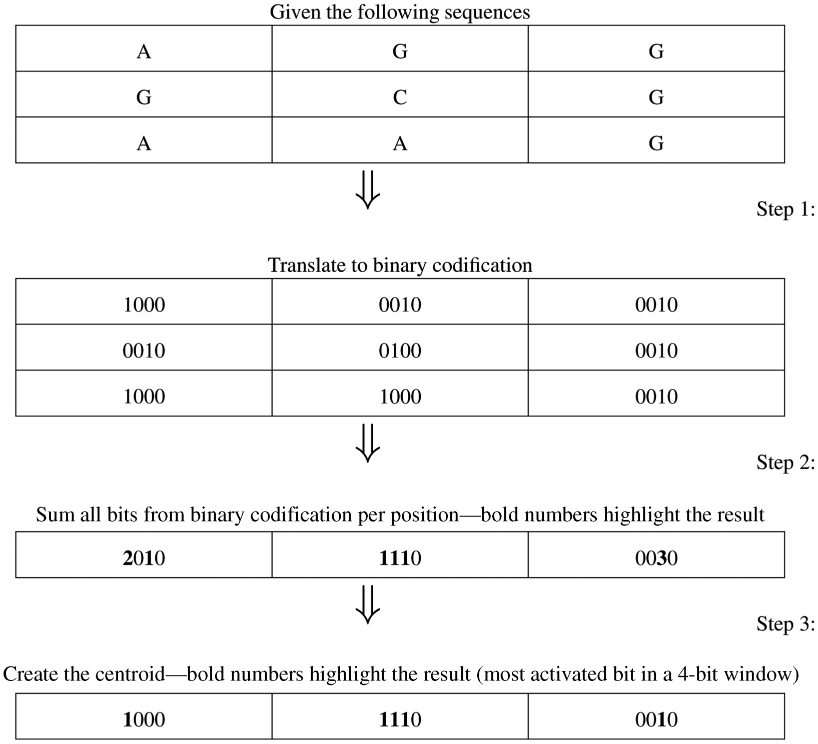
**Algorithm 1:** Pseudocode of *K-modes* and *K-means*

```
    Input: Sequence Set S
    Output: Clusters
 1  Randomly select the initial centroids;
 2  while have sequences changed cluster do
 3      Clear all temporary centroids tmpC;
 4      for each point n in S do
 5          for each centroid c in C do
 6              Calculate the distance between c and n;
 7              if The distance is the nearest found then
 8                  Move n to cluster c;
 9                  Update the number of points in centroid c;
10          update the position of tmpC;
11      for each centroid c do in C do
12          calculate the new value of position c based on the tmpC;
```

TABLE 2. DISTANCE BETWEEN SEQUENCES

| *Sequence* | *Binary representation* |
|---|---|
| ACG | 1000 0**100** 0010 |
| AUG | 1000 0**001** 0010 |
| Distance | 2 |

Bold numbers highlight the different bits used to calculate the distance.



Given the following sequences

| | | |
|---|---|---|
| A | G | G |
| G | C | G |
| A | A | G |

⇓

Step 1:

Translate to binary codification

| | | |
|---|---|---|
| 1000 | 0010 | 0010 |
| 0010 | 0100 | 0010 |
| 1000 | 1000 | 0010 |

⇓

Step 2:

Sum all bits from binary codification per position—bold numbers highlight the result

| | | |
|---|---|---|
| **20**10 | **111**0 | 00**3**0 |

⇓

Step 3:

Create the centroid—bold numbers highlight the result (most activated bit in a 4-bit window)

| | | |
|---|---|---|
| **1**000 | **111**0 | 00**1**0 |

**FIG. 2.** Example of a centroid calculation.

The centroid value is represented by a synthetic sequence, where each bit is composed of the mode of the most activated bit in a 4-bit window (a nucleotide). Figure 2 shows a centroid calculation example of a group composed of the sequences AGG, GCG, and AAG. When the clustering process is concluded, we selected the closest sequence to each cluster centroid. Those selected sequences are the most representative non-TIS sequences of each cluster, which will be used in the support vector machines (SVM) classifier (Hearst et al., 1998) to better predict the TIS sequences. To store the sequence, in contrast to Rodrigues et al. (2012), who chose a char array, we used a struct with three integers (64 bits) to store the sequences (totaling 192 bits per sequence).

The use of this struct reduces the computational cost of the distance calculating by removing an inner loop, reducing the number of operations. With only three XOR operations, we can evaluate the distance between the sequences (Table 2) substantially reducing the operations needed for this evaluation. Furthermore, all floating-point operations have been replaced by integer operations, which are more efficient. Given the operations performed to calculate the distance and update the centroids as a unit of computational cost, it is possible to express the computational cost of the *K-means* and *K-modes* algorithms as presented in Equations 1 and 2, respectively,

$$CostKmeans = 3pk\sum_{i=1}^{d} i + k\sum_{i=1}^{d} 2i, \tag{1}$$

$$CostKmode = 3pk\frac{d}{l} + k\sum_{i=1}^{d} 2i, \tag{2}$$

where $k$ is the number of centroids to be obtained, $p$ is the number of points of the data set (number of instances), $d$ is the dimension of the data set (number of attributes), and $l$ is the size of the integer in a given architecture. Therefore, using the *K-modes* instead of *K-means* in this TIS prediction context incurred in a significant reduction in the number of operations to be performed.

### 3.2. Parallel loops

The *K-means* and *K-modes* algorithms can easily be implemented in parallel. We focused on the loop responsible for calculating the distance between the points and the centroids, and the loop responsible for updating the centroids values, as those are the parts that most affect the algorithm's performance. Furthermore, they do not have many data dependencies, which simplify the parallel implementation using atomic operations, without reducing the performance.

### 3.3. *GPU* K-modes

GPUs are most effective when the data are granular because they are capable to do thousands of calculations simultaneously. Thus, we choose to implement the distance calculation and the centroid calculation in GPU. The part of the code that runs on GPUs is called the kernel. Algorithm 2 is the pseudocode of the kernel implemented in CUDA. In this code, *tmpC* is an auxiliary struct (an array of integer) that counts the number of active bits in each position of the centroid (represented by step 2 in Algorithm 2). *Delta* represents the number of sequences that had their cluster changed in a given iteration.

**Algorithm 2:** Pseudocode implemented in CUDA.

```
  Data: Set of Points $S$, Set of Centroids $C$
  Result: Clusters, tmpC, delta
1 foreach point $n$ in $S$ do
2     foreach centroid $c$ in $C$ do
3         Calculate the distance between $c$ and $n$;
4         if the distance is the nearest then
5             Add $n$ to $c$;
6             Increment Delta;
7     update the value of tmpC;
```

The kernel is responsible for calculating the distance between the instances and the cluster centroids. The kernel has two stop conditions: one in line 6, where we calculate the number of points which have their centroid changed, and the other one is in line 7, where we calculate the sum of all active bits presented in the centroid. The two stop conditions are arithmetic, solved by using the *atomicInc* function from the CUDA API. Minimizing the data transfer between host and device is essential to achieve a good performance in CUDA (Ryoo et al., 2008). For this reason, we stored all data used by the kernel in the device memory, so the algorithm does not need to transfer the data again to the device in the next iteration. One exception is the *Delta* variable because the host needs to access it in all iterations.

## 4. METHODS

This section shows the method to evaluate the parallel algorithms in the context of the TIS prediction problem. This way we highlight the following content: databases, the class balancing process, the computational environment, the evaluation metrics of the SVM classifier, and the developed parallel version.

### 4.1. Database description

The databases used in our experiments were extracted from the public repository RefSeq (Pruitt and Maglott, 2001) from the National Center for Biotechnology Information (NCBI).[2] Our study is based on the following organisms: *Arabidopsis thaliana* (AT), *Caenorhabditis elegans* (CE), *Drosophila melanogaster* (DM), *Homo sapiens* (HS), *Gallus gallus* (GG), *Mus musculus* (MM), and *Rattus norvegicus* (RN). Each molecule was identified according to its inspection level and classified as: *Model*, *Inferred*, *Predicted*, *Provisional*, *Reviewed*, *Validated*, and WGS.[3] In this work, we have considered only mRNA molecules

[2] Available at www.ncbi.nlm.nih.gov

[3] A description of each status is available at www.ncbi.nlm.nih.gov/books/NBK21091

Table 3. Extracted Sequences

| *Organism* | *TIS* | *Non-TIS* | *Proportion* |
|---|---|---|---|
| AT | 24329 | 587886 | 1:24 |
| DM | 19782 | 435892 | 1:22 |
| HS | 15845 | 353606 | 1:22 |
| CE | 8763 | 449240 | 1:51 |
| MM | 269 | 6256 | 1:23 |
| RN | 31 | 334 | 1:11 |
| GG | 13 | 534 | 1:41 |

AT, *Arabidopsis thaliana*; CE, *Caenorhabditis elegans*; DM, *Drosophila melanogaster*; GG, *Gallus gallus*; HS, *Homo sapiens*; MM, *Mus musculus*; RN, *Rattus norvegicus*; TIS, translation initiation sites.

with inspection level *Reviewed* since those underwent a thorough reviewing process that better ensures their reliability.

The procedure for extracting TIS and non-TIS sequences followed the methodology proposed by Silva et al. (2011). We collected the sequences from the *upstream* and *downstream* mRNA regions and out of reading frame. In accordance with the best result obtained in Silva et al. (2011), for each AUG of the mRNA molecule, we considered 10 nucleotides in the upstream region and 30 nucleotides in the downstream region (Fig. 1). Table 3 shows the number of TIS and non-TIS sequences extracted for each organism with the proportion between these two classes, which show the class imbalancing issue.

## 4.2. Class balancing

To address the class imbalancing issue (Table 3), we chose to do an undersampling of the majority class. This process was performed using the proposed *K-modes* clustering algorithm to select the most representative instances of the each cluster of instances from majority class (non-TIS). To define the number of clusters, we used the number of instances in the minority class, so the process results in equal number of instances of each class.

The balancing process occurred during the 10-fold cross validation (Kohavi, 1995), in which the data set was divided into 10 subsets. Nine subsets were used for training, whereas only one was used for validation. We applied the balancing in each fold for each training process. This process was repeated 10 times, and after the final repetition, the average performance and standard deviation were calculated.

## 4.3. SVM parameter definition

A SVM classifier was used to predict the TIS and non-TIS sequences. The SVM is a machine learning algorithm that can be used for both classification and regression problems. In this approach, each training sequence is represented as a point in the *n*-dimensional space and the algorithm works with the objective of finding the hyperplane that best distinguishes (separates) the analyzed classes.

The classifier's efficiency depends on an appropriate selection of the parameters of its kernel function and on a margin of separation of the optimal hyperplane, represented by *C*. We used the same parameters defined by Silva et al. (2011) and Rodrigues et al. (2012), that is, a fourth-order polynomial function was adopted and the trade-off between training error and margin was 1.0.

## 4.4. Test environment

The GPU experiments were executed on a computer with an Intel Core i5 2.7 GHz, 16GB of RAM, and an Nvidia GeForceGT 640M with 512 MB. The other experiments were executed in a cluster SGI Altix at the Centro Nacional de Supercomputação (CESUP).[4] Our cluster hardware contains 64 GB of RAM and 2 dodeca core AMD Opteron, totalizing 36 cores. This configuration intent to mimic the same test environment used by Rodrigues et al. (2012).

[4] Available at www.cesup.ufrgs.br

### 4.5. Performance measures

Five metrics were used to evaluate the SVM classifier: accuracy (*Ac*), precision (*Pr*), sensitivity (*Se*), specificity (*Sp*), and adjusted accuracy (*Adj*). To evaluate the performance of the parallel versions of the *K-means* and *K-modes* algorithms, we used the metric speedup (*Sd*).

The accuracy (Eq. 3) shows the rate of positive and negative examples correctly classified, where TP, TN, FP, and FN denote the number of true positives, true negatives, false positives, and false negatives, respectively.

$$Ac = 100 \times \frac{TP + TN}{TP + TN + FP + FN}. \tag{3}$$

The precision metric shows the proportion of positive examples correctly classified among all predicted as positives (Eq. 4).

$$Pr = 100 \times \frac{TP}{TP + FP}. \tag{4}$$

The sensitivity is relative to the hit rate in the positive class. It is also called rate of true positives (Eq. 5).

$$Se = 100 \times \frac{TP}{TP + FN}. \tag{5}$$

The specificity corresponds to the hit rate in the negative class (Eq. 6).

$$Sp = 100 \times \frac{TN}{TN + FP}. \tag{6}$$

The adjusted accuracy is the arithmetic average between the sensitivity and specificity, described by Equation 7.

$$Adj = \frac{Sensitivity + Specificity}{2}. \tag{7}$$

Finally, we use the Speedup (Eq. 8) to calculate the computational performance gain between the algorithm versions. $T_{\mathrm{seq}}$ represents the total time of the sequential version and the $T_{\mathrm{p}}$ is the total time of the parallel version.

$$Sd = \frac{T_{\mathrm{seq}}}{T_{\mathrm{p}}}. \tag{8}$$

## 5. RESULTS AND DISCUSSION

We performed experiments to analyze the behavior of *K-means* and *K-modes* algorithms for the TIS prediction problem as well as their performance. The databases were divided into two groups, large and small. The first group (large databases) is composed of the organisms AT, CE, DM, and HS. The second group (small databases) is composed of the organisms GG, MM, and RN.

Table 4. Execution Time in Seconds

| *Organism* | *Sequential time* | *GPU time* | *Speedup* |
|---|---|---|---|
| AT | 11368.20 | 55.87 | 203.45 |
| DM | 6221.25 | 32.22 | 193.08 |
| HS | 3913.76 | 22.22 | 176.13 |
| CE | 2110.14 | 15.5 | 136.13 |
| MM | 6.8 | 0.92 | 6.97 |
| GG | 0.09 | 0.87 | 0.10 |
| RN | 0.08 | 0.87 | 0.09 |

GPU, graphics processing unit.

Table 5. Classification Results

| *Organism* | *Ac-P* | *Ac-R* | *Pr-P* | *Pr-R* | *Se-P* | *Se-R* | *Sp-P* | *Sp-R* | *Adj-P* | *Adj-R* |
|---|---|---|---|---|---|---|---|---|---|---|
| AT | 91.82 | 91.3 | 32.24 | 27.74 | 91.59 | 83.27 | 91.83 | 91.25 | 91.71 | 87.26 |
| CE | 89.83 | 90.02 | 14.67 | 14.93 | 89.58 | 89.71 | 89.83 | 89.03 | 89.71 | 89.87 |
| DM | 93.55 | 95.22 | 31.06 | 43.01 | 92.83 | 90.83 | 93.58 | 90.47 | 93.21 | 60.64 |
| HS | 93.78 | 91.15 | 42.05 | 39.83 | 92.76 | 89.11 | 93.82 | 88.93 | 93.29 | 89.02 |
| GG | 99.29 | 99.3 | 80.00 | 80.00 | 75.00 | 75.00 | 100.00 | 100.00 | 87.5 | 87.5 |
| MM | 94.35 | 94.61 | 42.13 | 46.29 | 92.12 | 92.17 | 94.44 | 91.39 | 93.28 | 91.78 |
| RN | 95.42 | 93.71 | 71.00 | 45.00 | 56.66 | 92.01 | 96.06 | 91.20 | 77.86 | 91.70 |

Ac, accuracy; Adj, adjusted accuracy; Pr, precision; Se, sensitivity; Sp, specificity; P, proposal; R, Rodrigues et al. (2012).

For each database, the algorithm was executed 10 times, with the exception of the sequential version from Rodrigues et al. (2012), which was executed only three times per organism, due to the high execution time demanded. Table 4 shows the average execution time in seconds for each organism and the speedup between the sequential and GPU versions. The average execution time shows, as expected, that the performance is directly related to the number of sequences and centroids to be processed. For the large

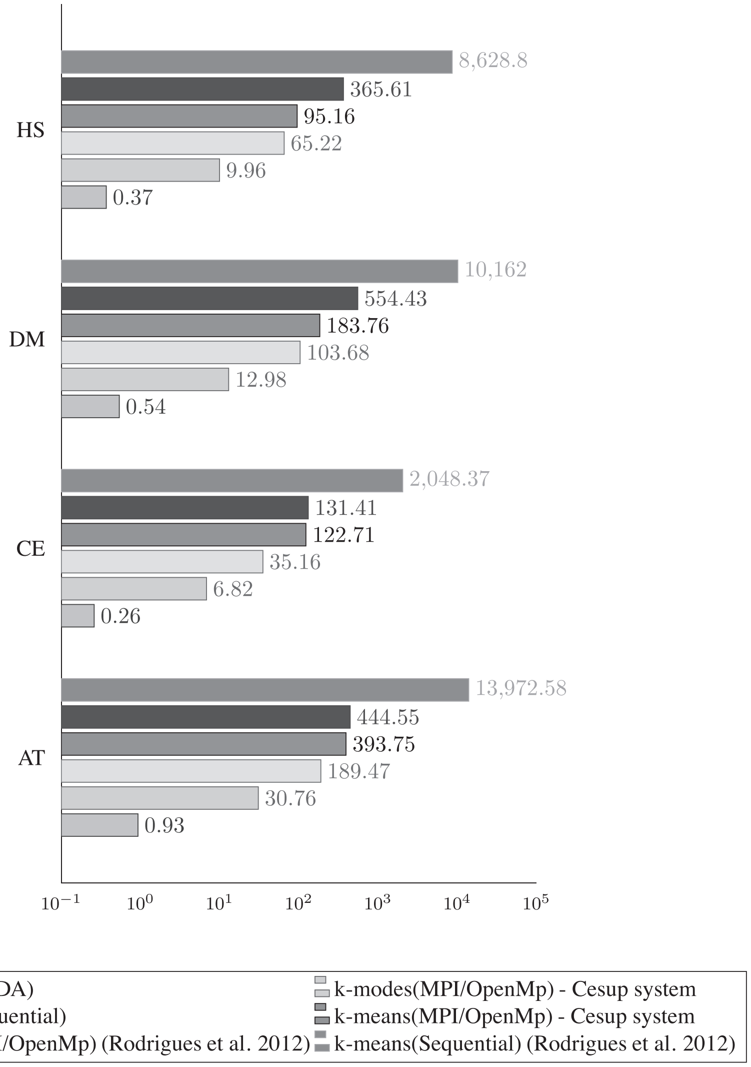


**FIG. 3.** Execution time in minutes for large bases. AT, *Arabidopsis thaliana*; CE, *Caenorhabditis elegans*; CUDA, Compute Unified Device Architecture; MPI, message passing interface; DM, *Drosophila melanogaster*; HS, *Homo sapiens*; OpenMP, Open Multi-Processing.

databases, the speedup was very high (up to 203.45 times). However, for two of the small databases (GG and RN), the execution time was slower in the GPU. This is because the overhead caused by the kernel launch and data transfers between host and device. In this case, the time to perform those operations is higher than the time to calculate the centroids.

Table 5 shows the 10-fold average for accuracy (*Ac*), precision (*Pr*), sensitivity (*Se*), specificity (*Sp*), and the adjusted accuracy (*Adj*) for each organism. Most of the results were very similar to those presented by Silva et al. (2011) and Rodrigues et al. (2012). For the large and small databases, the adopted methodology and the *K-modes* algorithm got better results for most of the metrics in comparison to the previous works. Although for RN (small database) there is a higher variation, the results are better except for sensitivity and adjusted accuracy.

Using our GPU *K-modes* implementation as clustering technique we achieved one of our objectives since the results of this article are similar those found in our previous work (Rodrigues et al., 2012) with an significant speedup and using less resources. Figure 3 presents the comparison between the execution times of the algorithm versions for the large databases: AT, CE, DM, and HS. Figure 4 presents the same comparison for the small databases: GG, MM, and RN.

According to the results shown in Figure 3, it is clear which versions of the algorithm are faster for large database. The sequential version of the *K-means* was the slowest, followed by its parallel versions. Interestingly, all versions of the *K-modes* (including the sequential) outperformed the *K-means* algorithm. This is very positive as it highlights how the algorithm used considerably fewer resources. Although the *K-means* executed by Rodrigues et al. (2012) and the *K-means* version executed in this work are the same, their execution times are slight different. Even theoretical replicating of the same configuration from the Rodrigues et al. (2012) experiments, the results obtained from the CESUP were better. Thus, the CESUP MPI implementation and the hardware are more optimized for internode communication, decreasing the overhead of the parallel execution. Therefore, our results corroborate with the scalability results from Rodrigues et al. (2012). The *K-modes* and *K-means* algorithms had a great performance in parallel

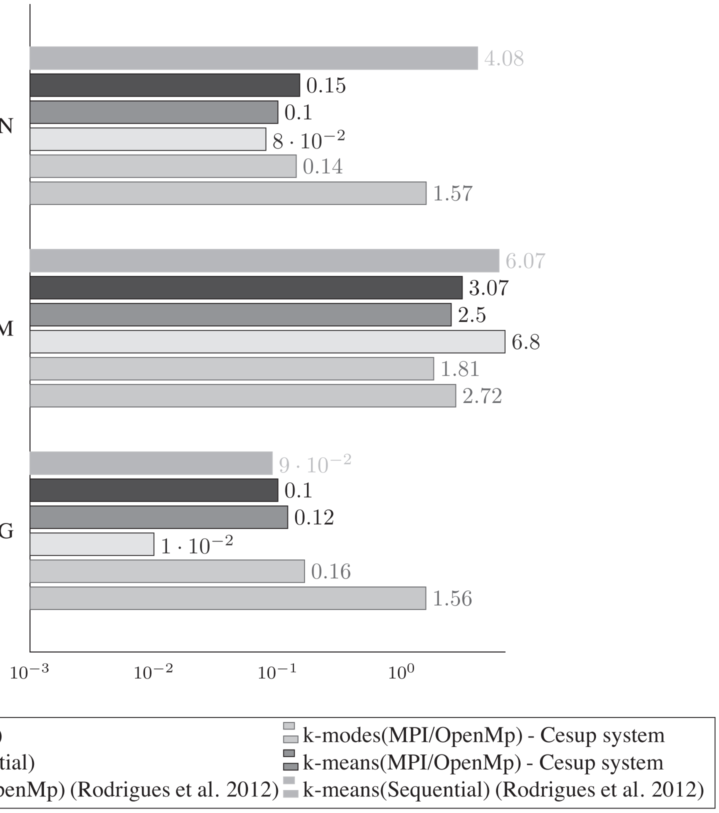


**FIG. 4.** Execution time in seconds for small bases. GG, *Gallus gallus*; MM, *Mus musculus*; RN, *Rattus norvegicus.*

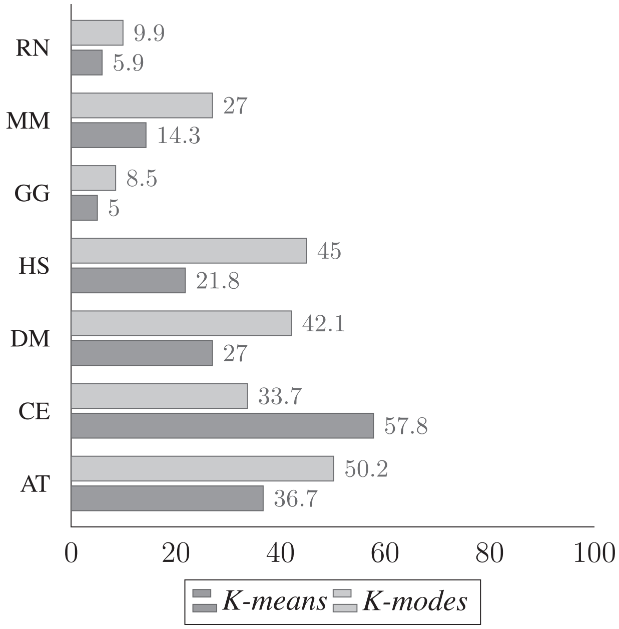


**FIG. 5.** Total number of iterations per organism.

environments, especially when using GPU. The speedup from the sequential *K-means* version to the CUDA *K-modes* has reduced the execution time from days to less than 1 minute.

Figure 4 shows results for small databases. For the smallest base (GG), all parallel versions were slowest. This means that the parallel versions expended more time managing resources than doing computation. Regarding to the RN, which is a slightly bigger database, the parallel versions using MPI/OpenMP achieved better results than the sequential version; however, its GPU version was still slower when compared with the sequential algorithm. Finally, the MM database, which can be considered a medium-sized database, had better results using the parallel versions. In this case, the MPI/OpenMP version showed a modest advantage in comparison to the CUDA version. In our experiments, the CUDA implementation nearly outperformed all other algorithms, but its overhead was shown to be higher than the MPI/OpenMP in our context.

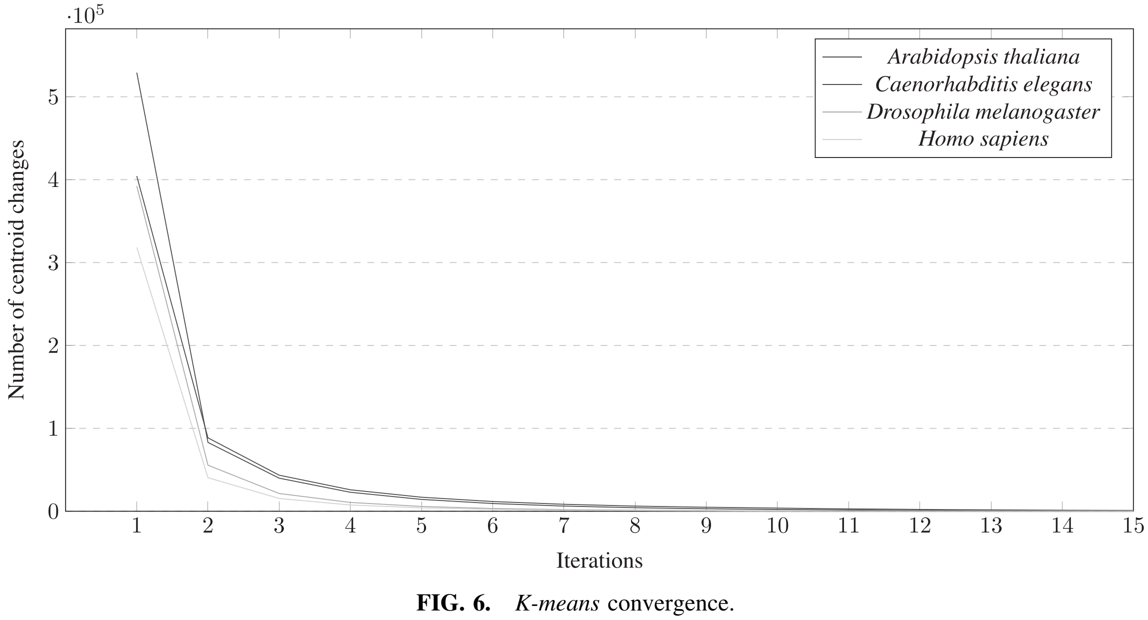


**FIG. 6.** *K-means* convergence.

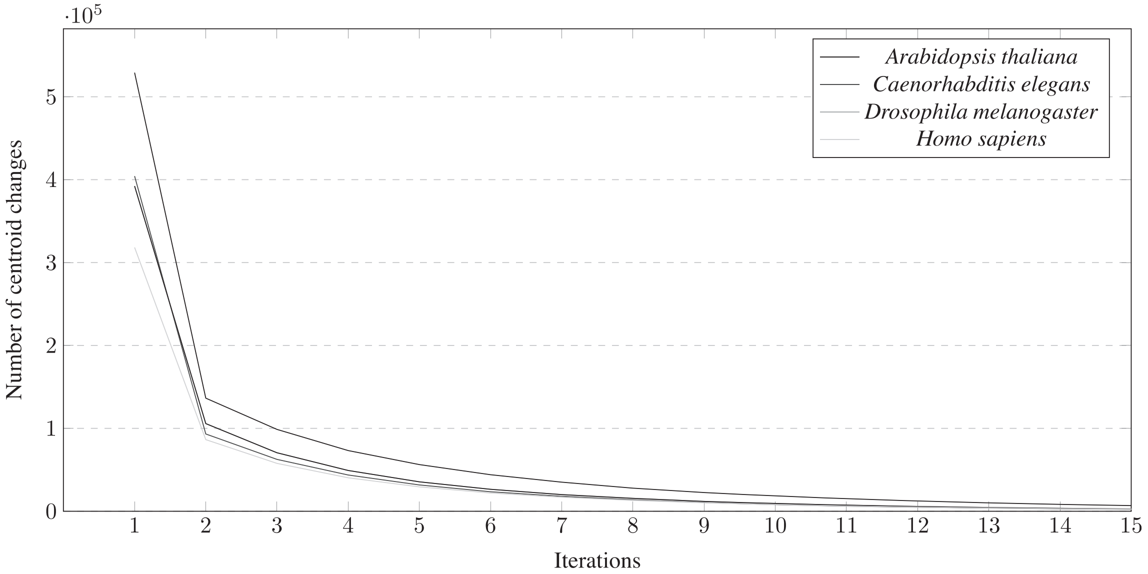


**FIG. 7.** *K-modes* convergence.

### 5.1. *Convergency*

Figure 5 presents the 10-fold average of the number of iterations per organism. It is clear that the *K-means* algorithm tends to converge faster than the *K-modes*, with the exception of the CE database. However, even with the slower convergence curve, the *K-modes* algorithm was still faster than the *K-means* in total execution time.

Figures 6 and 7 present the convergence behavior of the clustering process for the large databases of the *K-means* and *K-modes* algorithms, respectively, for the first 15 iterations. From these results, it is possible to evaluate the bias of the centroids. Note that the convergence (in other words, the cluster formation) was fast for all databases. The major changes are made in early iterations of the algorithm, once the initial centroid is chosen randomly. Also, Figure 5 corroborates with Figures 6 and 7 showing that the *K-means* algorithm tends to converge faster than the *K-modes*.

### 5.2. *Overall evaluation*

This section highlights the *K-modes* performance evaluation with focus on large bases (Fig. 3). First of all, the sequential *K-modes* implementation, disclosed in this article to cluster non-TIS sequences, reduced the processing time for all databases in relation to the *K-means* algorithm versions. We observed a 132 times faster performance (e.g., HS) compared with the sequential version. Moreover, using the parallel CUDA *K-modes* version, we achieved up to 23,321 and 988 times faster execution time (e.g., HS) compared with the sequential and parallel MPI/OpenMP *K-means* versions presented by Rodrigues et al. (2012). In comparison to the MPI/OpenMP *K-means* version executed on CESUP system, the parallel CUDA *K-modes* is 257 times faster. Concerning only *K-modes* versions, the parallel CUDA implementation (e.g., AT) obtained a speedup up to 203 and 33 times in comparison to the sequential and parallel MPI/OpenMP *K-modes* versions, respectively. Therefore, our proposed *K-modes* algorithm implementation to cluster non-TIS sequences proved to be the best alternative using less resources and having a huge performance gain without compromising quality of the classification results as presented in Table 5.

## 6. CONCLUSION

As presented is this article, the computational cost to balance the TIS and non-TIS sequences can be extremely high for large data sets, and research involving optimization is needed to make it viable. Our results showed how parallel CUDA *K-modes* is efficient for solving this issue. GPUs are important accelerators and can provide high performance in this field of Bioinformatics area. It is important to highlight that days of processing can be reduced to seconds and due to the several tasks that need computational power, fast processing is crucial for sharing computer resources.

Besides the validation of our proposal, our article shows the related state of the art the following main contribution: a fast parallel *K-modes* algorithm implemented in more than one approach, but mainly in CUDA, for clustering nucleotide sequences in the field of TIS prediction. This way, we reinforce the need to investigate this field to find and propose new improvements toward heterogeneous computing as approach to explore parallel and high-performance computing (HPC) in Bioinformatics. For this reason, further works in heterogeneous HPC are fundamental to new advances in this context.

## ACKNOWLEDGMENTS

This study was partially financed by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES)—Finance Code 001. We also thank CESUP, FAPEMIG, and CNPq for the partial support.

## AUTHOR DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

## REFERENCES

Batista, G.E.A.P.A., Prati, R.C., and Monard, M.C. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.* 6, 20–29.

Baydoun, M., Ghaziri, H., and Al-Husseini, M. 2018. CPU and GPU parallelized kernel K-means. *J. Supercomput.* 74, 3975–3998.

Cong, J., Fang, Z., Lo, M., et al. 2018. Understanding performance differences of FPGAs and GPUs, 288. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, New York, NY.

Dagum, L., and Menon, R. 1998. OpenMP: An industry standard API for shared-memory programming. *IEEE Comput. Sci. Eng.* 5, 46–55.

Dhanasekaran, B., and Rubin, N. 2011. A new method for GPU based irregular reductions and its application to K-means clustering, 2:1–2:8. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*. ACM, New York, NY.

Díaz, N., Velazco, A.F.R., and Márquez, C.A.O. 2014. Gene predictors ensemble for complex metagenomes, 147–154. *In* L.F. Castillo, M. Cristancho, G. Isaza, A. Pinzón and J.M.C. Rodríguez, eds. *Advances in Computational Biology*. Springer International Publishing, Cham.

Dysvik, B., and Jonassen, I. 2001. J-express: Exploring gene expression data using Java. *Bioinformatics* 17, 369–370.

Gropp, W., Lusk, E., Doss, N., et al. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.* 22, 789–828.

Hatzigeorgiou, A.G. 2002. Translation initiation start prediction in human cDNAs with high accuracy. *Bioinformatics* 18, 343–350.

Hearst, M.A., Dumais, S.T., Osuna, E., et al. 1998. Support vector machines. *IEEE Intell. Syst. Their Appl.* 13, 18–28.

Hong-tao, B., Li-li, H., Dan-tong, O., et al. 2009. K-means on commodity GPUs with CUDA, 651–655. In *2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 3.

Jamroz, M., and Kolinski, A. 2013. Clusco: Clustering and comparison of protein models. *BMC Bioinformatics* 14, 62.

Kang, Y., and Park, Y.B. 2015. The performance evaluation of k-means by two MapReduce frameworks, Hadoop vs. Twister, 405–406. In *2015 International Conference on Information Networking (ICOIN)*. Siem Reap, Cambodia.

Kantabutra, S., and Couch, A.L. 2000. Parallel K-means clustering algorithm on nows. *NECTEC Tech. J.* 1, 243–247.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection, 1137–1143. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence—Volume 2*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Kozak, M. 1984. Compilation and analysis of sequences upstream from the translational start site in eukaryotic mrnas. *Nucleic Acids Res.* 12, 857–872.

Li, Y., Zhao, K., Chu, X., et al. 2010. Speeding up k-means algorithm by GPUs, 115–122. In *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*. Bradford, UK.

Liu, Y., Guo, J., Hu, G., et al. 2013. Gene prediction in metagenomic fragments based on the SVM algorithm. *BMC Bioinformatics* 14(Suppl 5), S12.

Pruitt, K.D., and Maglott, D.R. 2001. RefSeq and LocusLink: NCBI gene-centered resources. *Nucleic Acids Res.* 29, 137–140.

Rao, S.N.T., Prasad, E.V., and Venkateswarlu, N.B. 2009. A scalable k-means clustering algorithm on multi-core architecture, 1–9. In *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*. Delhi, India.

Rodrigues, L.M., Zárate, L.E., Nobre, C.N., et al. 2012. Parallel and distributed kmeans to identify the translation initiation site of proteins, 1639–1645. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Seoul, South Korea.

Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., et al. 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, 73–82. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, New York, NY.

Saffran, J., Garcia, G., Souza, M.A., et al. 2017. A low-cost energy-efficient Raspberry Pi cluster for data mining algorithms, 788–799. In *Euro-Par 2016: Parallel Processing Workshops*. Springer International Publishing, Cham.

Silva, L.M., de Souza Teixeira, F.C., Ortega, J.M., et al. 2011. Improvement in the prediction of the translation initiation site through balancing methods, inclusion of acquired knowledge and addition of features to sequences of mRNA. *BMC Genomics* 12(Suppl 4), S9.

Souza, M.A., Maciel, L.A., Penna, P.H.M.M., et al. 2018. Energy efficient parallel k-means clustering for an Intel® hybrid multi-chip package, 372–379. In *High Performance Machine Learning Workshop, International Symposium on Computer Architecture and High Performance Computing*.

Stormo, G.D., Schneider, T.D., and Gold, L.M. 1982. Characterization of translational initiation sites in *E. coli*. *Nucleic Acid Res.* 10, 2971–2996.

Tao, G., Xiangwu, D., and Yefeng, L. 2015. Parallel k-modes algorithm based on MapReduce, 176–179. In *2015 Third International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*. Moscow, Russia.

Waghmare, V.N., and Kulkarni, D.B. 2010. Convex hull using k-means clustering in hybrid (MPI/OpenMp) environment, 150–153. In *2010 International Conference on Computational Intelligence and Communication Networks*. Bhopal, India.

Wu, R., Zhang, B., and Hsu, M. 2009. Clustering billions of data points using GPUs, 1–6. In *Proceedings of the Combined Workshops on Unconventional High Performance Computing Workshop Plus Memory Access Workshop*. ACM, New York, NY.

Xu, Y.H., Vidal, M., Arejita, B., et al. 2018. Implementation of the k-means algorithm on heterogeneous devices: A use case based on an industrial dataset. *Adv. Parallel Comput.* 32, 642–651.

Yeung, K.Y., Haynor, D.R., and Ruzzo, W.L. 2001. Validating clustering for gene expression data. *Bioinformatics* 17, 309–318.

Address correspondence to:
*Dr. Henrique C. Freitas*
*Department of Computer Science*
*Pontifícia Universidade Católica de Minas Gerais*
*Av. Dom Jose Gaspar 500*
*Belo Horizonte 30535-901*
*Minas Gerais*
*Brazil*

*E-mail:* cota@pucminas.br