

Training Optimum-Path Forest on Graphics Processing Units

Adriana S. Iwashita¹, Marcos V. T. Romero¹, Alexandro Baldassin², Kelton A. P. Costa¹
and João P. Papa¹

¹*Department of Computing, São Paulo State University, Bauru, São Paulo, Brazil*

²*Department of Statistics, Applied Mathematics and Computation, São Paulo State University, Rio Claro, São Paulo, Brazil*

Keywords: Optimum-Path Forest, Graphics Processing Unit.

Abstract: In this paper, we presented a Graphics Processing Unit (GPU)-based training algorithm for Optimum-Path Forest (OPF) classifier. The proposed approach employs the idea of a vector-matrix multiplication to speed up both traditional OPF training algorithm and a recently proposed Central Processing Unit (CPU)-based OPF training algorithm. Experiments in several public datasets have showed the efficiency of the proposed approach, which demonstrated to be up to 14 times faster for some datasets. To the best of our knowledge, this is the first GPU-based implementation for OPF training algorithm.

1 INTRODUCTION

Pattern recognition techniques have as the main goal to learn a discriminative function that separates the dataset samples in different classes. Such learning process is performed over a training set, which can be labeled or not, and the effectiveness of the classifier is assessed on a testing set (Duda et al., 2000). Thus, the learning is the most important phase of a classifier, and also the most computational expensive.

In the recent years, there is a considerable number of works that focuses on developing machine learning algorithms on Graphics Processing Units (GPUs). Catanzaro et al. (Catanzaro et al., 2008), for instance, presented a GPU-based implementation for the well-known Support Vector Machines (SVM) classifier, which was about $9 - 35\times$ faster when compared to the traditional CPU (Central Processing Unit) implementation.

In the same year, Do et al. (Do et al., 2008) also proposed an SVM implementation on GPUs, and Jang et al. (Jang et al., 2008) presented an Artificial Neural Network (ANN) version using Compute Unified Device Architecture (CUDA) and OpenMP (Open Multi-Processing), which is a library for concurrent programming. In this work, the authors highlighted the importance of considering the real needs of massive programming, i.e., the algorithms that will be implemented in such parallel environments need to contain modules that can be executed concurrently. Otherwise, the throughput of data interchange between

processors may degrade the performance of the whole system.

Papa et al. (Papa et al., 2009a; Papa et al., 2012) proposed a pattern recognition technique called Optimum Path Forest (OPF) aiming to overcome the time spent with the learning phase. They showed that OPF can obtain similar effectiveness to SVM, but it can be faster in the training phase. The OPF models the problem of pattern recognition as a graph partition into optimum-path trees (OPTs), which are rooted by key samples (prototypes) that compete among themselves in order to conquer the remaining samples. Further, Iwashita et al. (Iwashita et al., 2012) proposed an optimization of the OPF training algorithm in CPU, being such approach up to two times faster than traditional OPF with similar recognition rates.

In this paper, we presented an GPU-based implementation of the OPF training phase, which can be faster than traditional OPF and also more efficient than the approach proposed in (Iwashita et al., 2012). We have validated the proposed technique using several public datasets. As far as we know, this is the first GPU-based implementation of the OPF classifier. The remainder of this paper is organized as follows. Sections 2 and 3 present the OPF background and the proposed CUDA-based training algorithm, respectively. Section 4 discusses the experimental results, and Section 5 states conclusions.

2 PATTERN CLASSIFICATION USING OPTIMUM-PATH FOREST

Let $Z = Z_1 \cup Z_2$ be a dataset where Z_1 and Z_2 are the training and test sets, respectively, with $|Z_1|$ and $|Z_2|$ samples. Let $\lambda(s)$ be the function which associates the correct label $i, i = 1, 2, \dots, c$ to any sample $s \in Z_1 \cup Z_2$ from class i . Let $S \in Z_1$ be the set of prototype samples, and let \vec{v} be the algorithm that extracts n features from any sample $s \in Z_1 \cup Z_2$, and return a feature vector $\vec{v}(s) \in \mathbb{R}^n$. The distance $d(s, t)$ between two samples, s and t , is given by the distance between his feature vectors $\vec{v}(s)$ e $\vec{v}(t)$. The problem consists in use $S, (\vec{v}, d), Z_1$ and Z_2 to project an optimal classifier, which can predict the correct label $\lambda(s)$ of any sample $s \in Z_2$.

Let (Z_1, A) be a complete graph in which nodes are samples in Z_1 , where any pair of samples defines an edge on A (i.e., $A = Z_1 \times Z_1$), creating a completed weighted graph of Z_1 set. A path is a sequence of samples $\pi = \langle s_1, s_2, \dots, s_k \rangle$, where $(s_i, s_{i+1}) \in A$ for $1 \leq i \leq k - 1$. A path is said to be trivial if $\pi = \langle s_1 \rangle$. It is associated for each path π the cost given by a smooth cost function f (Falcão et al., 2004), denoted by $f(\pi)$. A path is optimum if $f(\pi) \leq f(\tau)$ for any path τ , where π and τ ends in the same sample s , regardless of their origin. We also denote $\pi \cdot \langle s, t \rangle$ the concatenation of the path π ending in s and edge (s, t) .

The OPF algorithm can be used with any soft cost function, which can combine samples with similar properties (Falcão et al., 2004). However, the OPF was designed using the f_{max} cost function, due to its theoretical properties to estimate optimum prototypes (Allène et al., 2007):

$$\begin{aligned} f_{max}(\langle s \rangle) &= \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise} \end{cases} \\ f_{max}(\pi \cdot \langle s, t \rangle) &= \max\{f_{max}(\pi), d(s, t)\}, \end{aligned} \quad (1)$$

the $f_{max}(\pi)$ compute the maximum distance between adjacent samples in π , when π is not a trivial path.

The OPF algorithm associates an optimal path $P^*(s)$ of S to all samples $s \in Z_1$, making a optimum path forest P (a function without cycles which associates to all $s \in Z_1$ his predecessor $P(s)$ in $P^*(s)$, or assigns *nil* when $s \in S$). Let $R(s) \in S$ the root of $P^*(s)$ which can be achieved using $P(s)$. The OPF algorithm computes, for each $s \in Z_1$, the cost $V(s)$ of $P^*(s)$, the label $L(s) = \lambda(R(s))$ and his predecessor $P(s)$.

We say S^* is an optimum set of prototypes when the algorithm propagates the labels $L(s) = \lambda(s)$ for all $s \in Z_1$. Thus, S^* can be found by exploring the theoretical relationship between the Minimum Spanning

Tree (MST) (Allène et al., 2007) and the minimum path tree to f_{max} . The training consists essentially of two steps: finding S^* and a OPF classifier rooted at S^* .

When we compute a MST on the completed graph (Z_1, A) , we obtain a connected acyclic graph whose nodes are all the samples in Z_1 , and the edges are not directed and weighted. Their weights are given by the distance d between the features vectors of adjacent samples. This spanning tree is optimum in the sense that the sum of the weights of its edges is minimal compared to other spanning trees on the completed graph. On MST, each pair of samples is connected by a path, which is optimum under f_{max} .

The optimum prototypes are the connected elements on MST with different labels Z_1 , i.e., the closest elements from different classes. By removing the edges between different classes, such adjacent samples become prototypes in S^* and can compute an optimum path forest without misclassification at Z_1 (competition process). Therefore, each prototype defines its own Optimum-Path Tree (OPT), and the collection of all OPT's define the optimum-path forest, which gives the name to the classifier.

The classification step is straightforward, i.e., given a test sample $s \in Z_2$, we connect it to all training nodes of the optimum-path forest generated in the training phase, and we evaluate which node $p \in Z_1$ minimizes the equation below:

$$V(s) = \min_{\forall Z_1} \max\{V(p), d(p, s)\}. \quad (2)$$

Thus, the node $p \in Z_1$ that minimizes $V(s)$ will be the one that conquer s .

3 TRAINING OPTIMUM-PATH FOREST ON CUDA

The implementation of the OPF training phase on CUDA environment was conducted using particular code instructions to execute similar actions in parallel. As graphics cards have their processing structures organized into parallel architectures, it is important to shed light over the implemented algorithms must have parallelizable modules to obtain performance gain.

As described in Section 2, the OPF training step is composed by two modules: (i) the first one computes the MST over the complete graph and marks the connected elements with different labels as prototypes; and (ii) with prototypes in hand, we can proceed with the competition process between them to build the OPTs.

The proposed GPU-based implementation of OPF

classifier also employs the above idea, i.e., we first paralleled the MST computation and then the competition process between prototypes. For both approaches, there are three main steps: to find the element p with lowest cost that will try to conquer the other nodes, to calculate the distance between node p and other nodes, and to verify whether the node p can conquer other nodes or not.

In this paper, we adopted a similar procedure to compute the MST and the optimum-path forest. The methodology is basically the same, i.e., one needs to find the node with lowest cost to begin the competition process. Considering the MST computation, for instance, the root node r will be associated with cost 0, and the remaining nodes with cost $+\infty$. Thus, we can force r to control the MST computation. The next step is to compute the distance between r and the remaining nodes to find out the shortest one, which will be the new leader, i.e., the sample t with lowest cost. This procedure can be conducted using f_{sum} , which is similar to f_{max} path-cost function (Equation 1):

$$f_{sum}(\langle s \rangle) = \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise} \end{cases}$$

$$f_{sum}(\pi \cdot \langle s, t \rangle) = f_{sum}(\pi) + d(s, t). \quad (3)$$

In this case, S stands for the set of root nodes. In the context of MST computation, we have only one root node. As one can see, the cost offered to t is just the summation of the cost of s with the distance (arc-weight) between s and t . Notice that we can slightly change some points to make the above procedure capable to compute the optimum-path forest: instead of using just one node root to begin the competition algorithm to compute MST, we can now use a set of prototype nodes, i.e., the connected nodes in the minimum spanning tree with different labels. Then, we can place f_{max} instead of f_{sum} and execute the competition process again.

In regard to the first step, i.e., to find out the element with the lowest cost, we employed the primitive *min_element* from *Thrust* (Hoberock and Bell, 2010) library, which returns the element with the lowest cost from a set of nodes using parallel algorithms. *Thrust* is a library to the development of parallel algorithms based on the C++ STL (Standard Template Library), and works like a high-level interface to increase developer productivity.

The next step is to compute the distance between the current node and the remaining ones. In this case, we employed a collection of threads to take care of a set of nodes. This means the distance between node p and the remaining nodes of the graph will be computed in parallel using an approach similarly to a multiplication of a matrix by a vector (Fujimoto, 2008).

The rows of this matrix, called here as “feature matrix”, are the training set nodes represented by their feature vectors. This matrix is allocated in the texture memory, and can be accessed by all threads; the feature vector of a node p is allocated on the global memory.

A thread is allocated for each element of the feature matrix, which will calculate the distance between an element of the matrix with node p , storing this result in a “partial results matrix”. Each block of threads has an array of partial results, which are allocated in the shared memory of them. The final result (distances between p and other nodes) is a distance vector allocated on the global memory, in which position i means the distance between nodes p and i . The level of parallelism of these operations can be classified as fine-granularity, since there are a large number of small and simple processes running on the instruction level.

Figures 2 to 10 exemplifies the proposed approach. Figure 1 illustrates the training set: the graph nodes are initialized with cost ∞ , and the node 1 will be the root of the MST; thus the cost 0 is assigned to it. As aforementioned, the first step of OPF training algorithm is to compute the MST and to find the prototypes. In Figure 2, we can face the situation when the proposed algorithm is in the first iteration (MST computation). Using the *min_element* primitive, node 1 leaves the queue to conquer the remaining nodes, since it has the lowest cost. The set of GPU threads is responsible for calculating the distance between node 1 and the other nodes; it also verifies whether the node cost is lower than the distance offered by node 1.

Figure 3 illustrates the situation after the first iteration, and the selection of node 2 (using *min_element* primitive) as the next dominant node. The dotted nodes in Figure 4 represent the prototypes found out after the 4th iteration. The arrow highlights the node 4, which will begin the conquering process. As showed in Figure 2, node 4 now leaves the queue to conquer the other nodes, and the set of GPU threads is

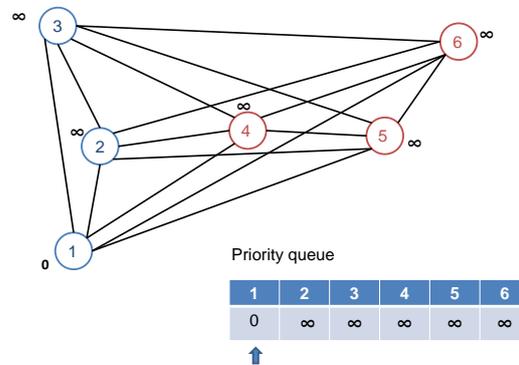


Figure 1: Training set.

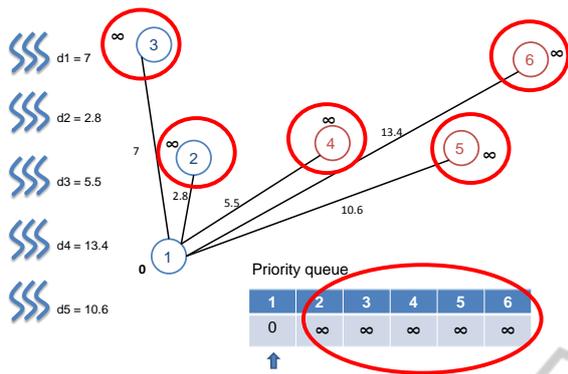


Figure 2: First iteration of the MST algorithm.

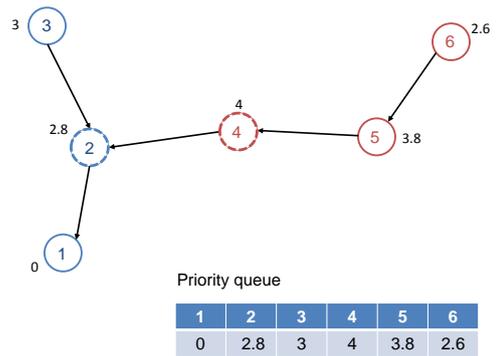


Figure 5: MST generated for the training set.

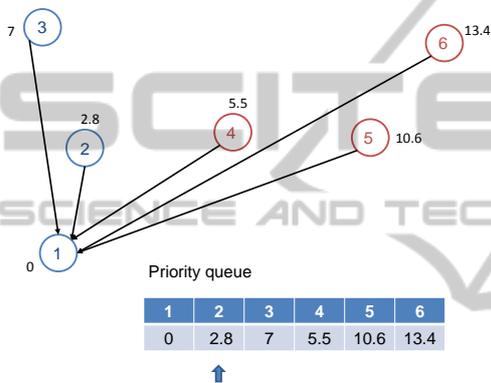


Figure 3: Step 2 of the MST algorithm.

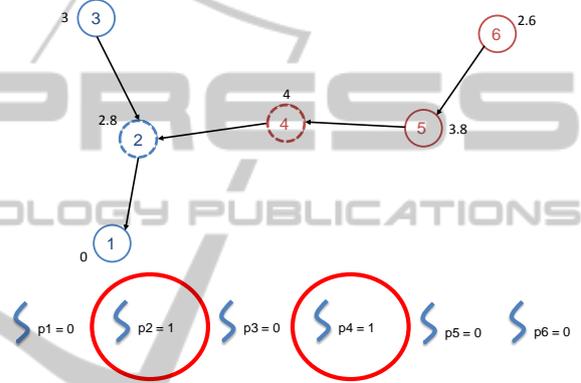


Figure 6: Each thread responsible for checking whether a node is a prototype or not.

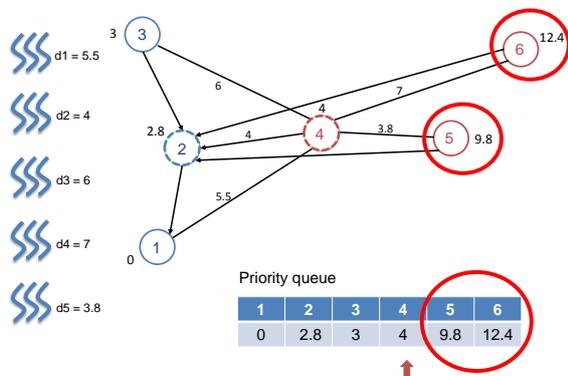


Figure 4: After the 4th iteration, two prototypes are found out and node 4 now conquers two nodes.

responsible for calculating the distance between node 4 and the other nodes.

Figure 5 depicts the final MST, and the dotted nodes stand for the prototypes. Figure 6 illustrates the set of nodes after the MST construction. In this phase, each thread is responsible for updating a node, i.e., each thread verifies whether the node has been assigned as a prototype or not. If the node is prototype, the thread updates the node cost to 0 and updates this

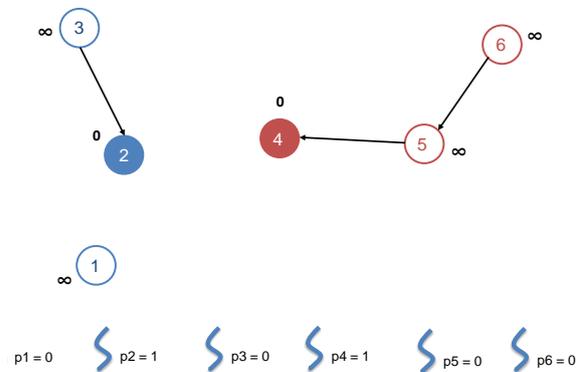


Figure 7: Each thread updates the cost of the node; and if it is also a prototype, the thread updates its predecessor.

predecessor node to *NIL*. If the node is not a prototype, this cost value will be ∞ . Figure 7 illustrates the nodes after the aforementioned updating process.

Figure 8 illustrates the 2nd phase of the training step: the *min_element* primitive chooses the prototype node 2 to leave the queue and then to conquer the remaining nodes.

Figure 9 shows the node 2 on action: each set

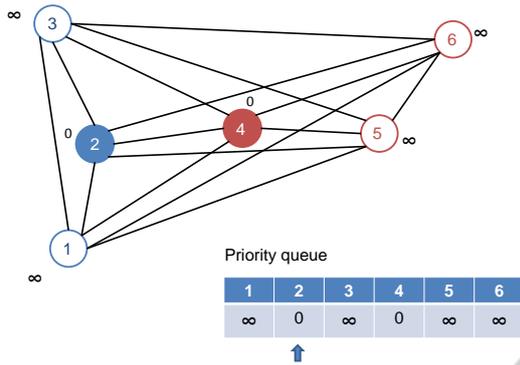


Figure 8: Graph with the prototypes marked: 2nd phase of the training step.

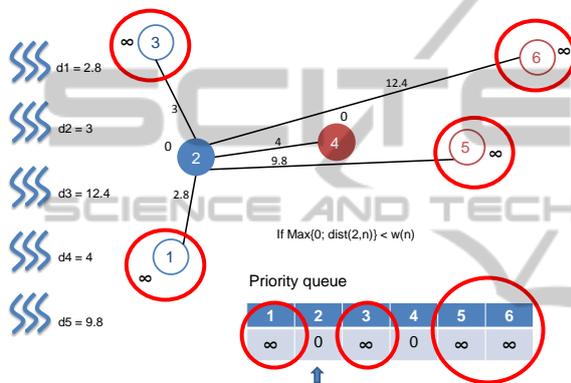


Figure 9: Each set of threads calculates the distance between 2 and the other nodes.

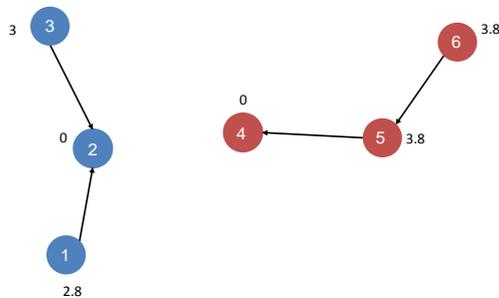


Figure 10: Resulting optimum-path forest.

of GPU threads is responsible for calculating the distance between node 2 and the other nodes. Using the path-cost function f_{max} (Equation 1), node 2 tries to conquer the other nodes.

Finally, Figure 10 presents the resulting optimum-path forest using the proposed algorithm.

4 EXPERIMENTS

In this section, we discuss the experiments conducted

in order to show the efficiency of the proposed optimization for OPF training step. We have used several public datasets with different sizes to assess the training efficiency, and also the classification accuracy in distinct scenarios. Table 1 presents the datasets used in this work.

Table 1: Datasets used in the experiments. With respect to Covtype, we have used 25% of the original dataset.

Dataset	# Samples	# Features
Connect (Frank and Asuncion, 2013)	67557	126
Indian Pine (Landgrebe, 2005)	21025	220
Letter (King et al., 1995)	15000	16
Mnist (LeCun et al., 1988)	60000	780
Poker (Frank and Asuncion, 2013)	25010	10
Salinas (Kaewpajit et al., 2003)	111104	204

In regard to training and test set sizes, we employed a cross-validation procedure with different percentages for that sets, which range from 10% to 90% of the training set size, with steps of 10. In order to validate the proposed GPU-based OPF training algorithm, we have compared it against traditional OPF and the optimization presented by Iwashita et al. (Iwashita et al., 2012).

Figures 11 and 12 display, respectively, the mean accuracy and training times with different percentages of training and test sets for Connect dataset. Notice the proposed approach has a gain about 2.523 with 30% of the training set. If we consider 90% of training dataset, the GPU-based OPF is about 5.522 faster than traditional one. Notice the accuracies are also similar among the compared techniques, being the proposed approach also faster than the algorithm proposed by (Iwashita et al., 2012).

Figures 13 and 14 display, respectively, the mean accuracy and training times for Indian Pine dataset. Considering 40% of the training set size, the proposed approach has been 2.004 times faster for training than traditional OPF. If we take into account 90% of training set size, the gain of GPU-based OPF is 4.021 times when compared with traditional OPF.

Figures 15 and 16 display, respectively, the mean accuracy and training times for Letter dataset. Notice the proposed approach was 1.825 times slower than traditional OPF with 90% of the training set size. This same behaviour can be observed with Poker dataset (Figures 19 and 20). The problem concerns with the few number of features in these datasets, which did not compensate the data interchange between CPU and GPU, since the distances are also computed using different threads.

Figures 17 and 18 display, respectively, the mean accuracy and training times for Mnist dataset. In this

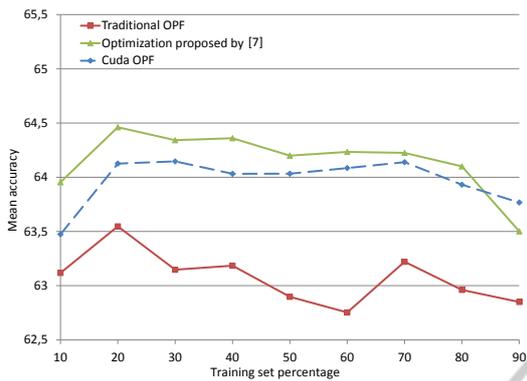


Figure 11: Mean classification accuracy for Connect dataset.

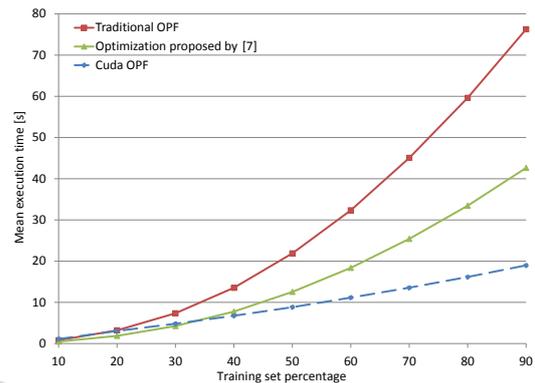


Figure 14: Mean execution times in seconds with respect to the training step for Indian Pine dataset.

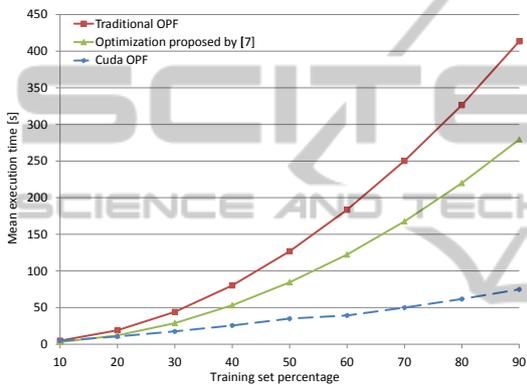


Figure 12: Mean execution times in seconds with respect to training step for Connect dataset.

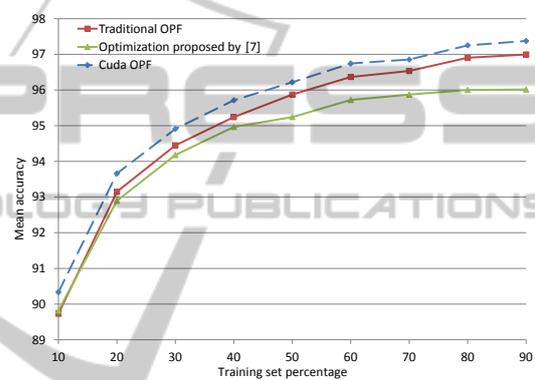


Figure 15: Mean classification accuracy for Letter dataset.

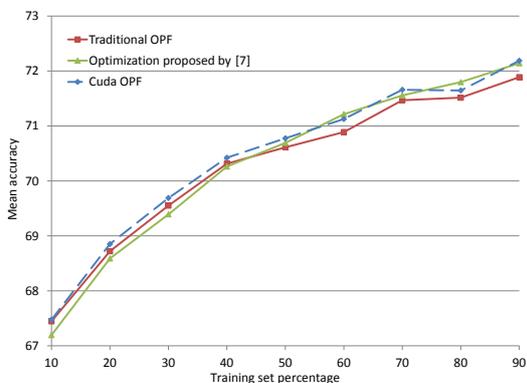


Figure 13: Mean classification accuracy for Indian Pine dataset.

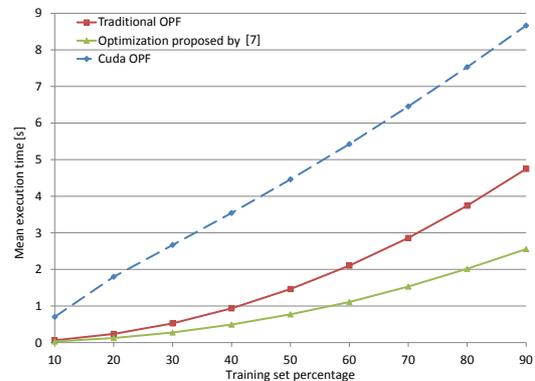


Figure 16: Mean execution times in seconds with respect to the training step for Letter dataset.

case, we can see a gain of 14.405 times with 90% of the training size of the proposed approach when compared against with traditional OPF.

Figures 21 and 22 display, respectively, the mean accuracy and training times for Salinas dataset. If we consider 10% of the training set size, the proposed approach is 3.038 times faster than traditional OPF; if we take into account 90% of the training set size, the

proposed approach is 13.580 faster than traditional OPF. In this case, the GPU-based OPF has been more accurate than the compared variants.

5 CONCLUSIONS

In this paper, we presented a GPU-based training algorithm of the OPF classifier. Our approach employs

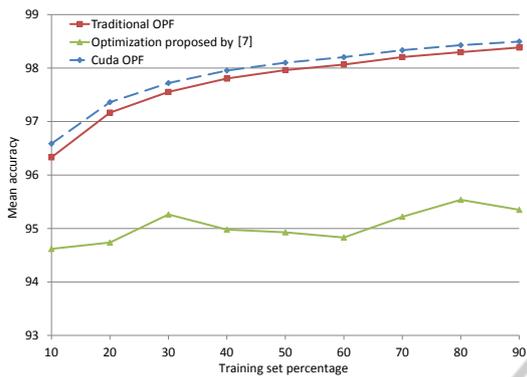


Figure 17: Mean classification accuracy for Mnist dataset.

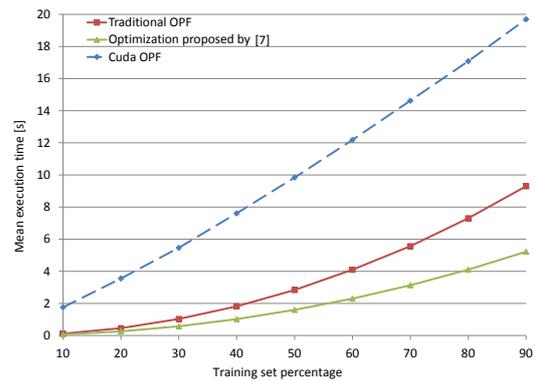


Figure 20: Mean execution times in seconds with respect to the training step for Poker dataset.

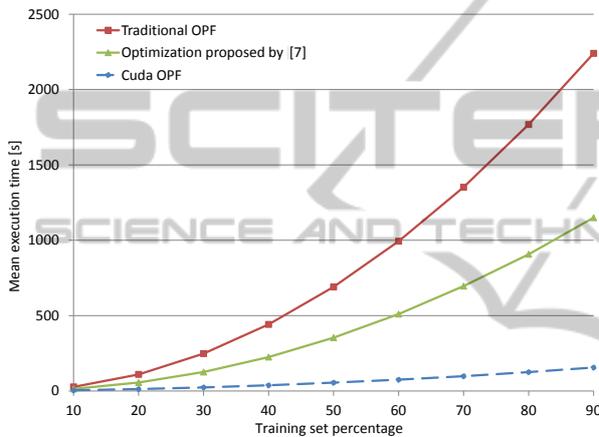


Figure 18: Mean execution times in seconds with respect to training step for Mnist dataset.

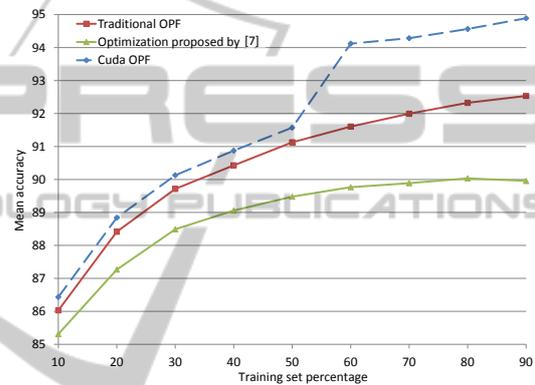


Figure 21: Mean classification accuracy for Salinas dataset.

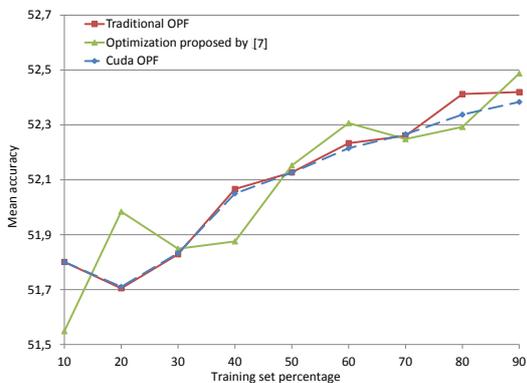


Figure 19: Mean classification accuracy for Poker dataset.

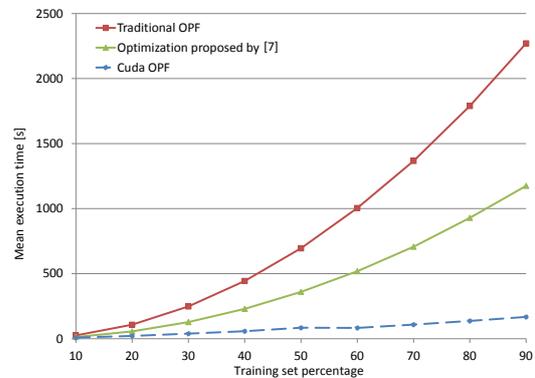


Figure 22: Mean execution times in seconds with respect to the training step for Salinas dataset.

the idea of a vector-matrix parallel multiplication, in which the same framework is employed for both MST and optimum-path forest computation.

Experimental results have shown that the proposed technique have obtained similar recognition rates to those obtained by the traditional approach and the CPU-based optimization proposed by Iwashita et al. (Iwashita et al., 2012). Additionally, the proposed

approach has been faster for training, except when we have datasets with a few features. In this case, the throughput of the data interchange between CPUs and GPUs did not compensate the parallelization effort. In datasets with a considerable number of features, we have observed a gain up to 14 times.

ACKNOWLEDGEMENTS

The authors are grateful to FAPESP grants #2009/16206-1, #2010/12697-8 and #2011/08348-0, and CNPq grants #470571/2013-6 and #303182/2011-3.

REFERENCES

- Allène, C., Audibert, J. Y., Couprie, M., Cousty, J., and Keriven, R. (2007). Some links between min-cuts, optimal spanning forests and watersheds. In *Proceedings of the International Symposium on Mathematical Morphology*, pages 253–264. MCT/INPE.
- Catanzaro, B., Sundaram, N., and Keutzer, K. (2008). Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on Machine learning*, pages 104–111, New York, NY, USA. ACM.
- Do, T.-N., Nguyen, V.-H., and Poulet, F. (2008). Speed up svm algorithm for massive classification tasks. In *Proceedings of the 4th international conference on Advanced Data Mining and Applications*, pages 147–157, Berlin, Heidelberg. Springer-Verlag.
- Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience.
- Falcão, A., Stolfi, J., and Lotufo, R. (2004). The image foresting transform theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):19–29.
- Frank, A. and Asuncion, A. (2013). UCI machine learning repository.
- Fujimoto, N. (2008). Faster matrix-vector multiplication on geforce 8800gtx. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8.
- Hoferock, J. and Bell, N. (2010). *Thrust: A Parallel Template Library*. Available at <http://www.meganewtons.com/>, Version 1.3.0.
- Iwashita, A. S., Papa, J. P., Falcão, A. X., Lotufo, R., Oliveira, V. M., Albuquerque, V. H. C., and Tavares, J. M. R. S. (2012). Speeding up optimum-path forest training by path-cost propagation. In *Proceedings of the XXI International Conference on Pattern Recognition*, pages 1233–1236.
- Jang, H., Park, A., and Jung, K. (2008). Neural network implementation using cuda and openmp. In *DICTA '08: Proceedings of the 2008 Digital Image Computing: Techniques and Applications*, pages 155–161, Washington, DC, USA. IEEE Computer Society.
- Kaewpijit, S., Moigne, J., and El-Ghazawi, T. (2003). Automatic reduction of hyperspectral imagery using wavelet spectral analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 41(4):863–871.
- King, R. D., Feng, C., and Sutherland, A. (1995). Statlog: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):289–333.
- Landgrebe, D. (2005). *Signal Theory Methods in Multi-spectral Remote Sensing*. Wiley, Newark, NJ.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1988). Gradient-based learning applied to document recognition. Available at <http://yann.lecun.com/exdb/mnist/>.
- Papa, J. P., Falcão, A. X., Albuquerque, V. H. C., and Tavares, J. M. R. S. (2012). Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, 45(1):512–520.
- Papa, J. P., Falcão, A. X., and Suzuki, C. T. N. (2009a). Supervised pattern classification based on optimum-path forest. *International Journal of Imaging Systems and Technology*, 19(2):120–131.
- Papa, J. P., S., C. T. N., and Falcão, A. X. (2009b). *LibOPF: A library for the design of optimum-path forest classifiers*. Software version 2.0 available at <http://www.ic.unicamp.br/~afalcao/LibOPF>.