A New Parallel Training Algorithm for Optimum-Path Forest-based Learning

Aldo Culquicondor[1], César Castelo-Fernández[1], and João Paulo Papa[2]

[1] Escuela de Ciencia de la Computación, Universidad Católica San Pablo
Arequipa, Perú
{aldo.culquicondor,ccastelo}@ucsp.edu.pe

[2] Computer Science Department, São Paulo State University - UNESP
Bauru, Brazil
papa@fc.unesp.br

Abstract. In this work, we present a new parallel-driven approach to speed up Optimum-Path Forest (OPF) training phase. In addition, we show how to make OPF up to five times faster for training using a simple parallel-friendly data structure, which can achieve the same accuracy results to the ones obtained by traditional OPF. To the best of our knowledge, we have not observed any work that attempted at parallelizing OPF to date, which turns out to be the main contribution of this paper. The experiments are carried out in four public datasets, showing the proposed approach maintains the trade-off between efficiency and effectiveness.

Keywords: Optimum-Path Forest, Parallel algorithms, Graph algorithms

1 Introduction

Pattern recognition is becoming even more important mainly due to the increasing needs from different applications to extract meaningful information from their data. Additionally, the problem gets worse since data is growing fast in both size and complexity. Humans have an innate ability to recognize patterns, but this is rather difficult to replicate on computers. Several techniques have been developed to address this issue, being the most popular ones Artificial Neural Networks (ANNs) [6] and Support Vector Machines (SVMs) [3]. Recently, a new framework to the design of graph-based classifiers named Optimum Path Forest (OPF) has been introduced in the scientific community. Such framework comprises supervised [10,12,11], semi-supervised [1,2] and unsupervised learning algorithms [14]. As the main advantages, we shall observe some OPF variants are parameterless, and they do not make assumptions about separability of samples. [7]

We refer to OPF as a single classifier in this paper, but it is in fact a framework to the design of graph-based classifiers. This means the user can design his/her own optimum-path forest-driven classifier by configuring three main modules: (i) *adjacency relation*, (ii) *methodology* to estimate prototypes, and (iii) *path-cost function.* Since OPF models the problem of pattern recognition as a graph partition task, it requires an adjacency relation to connect nodes (i.e. feature vectors extracted from dataset samples). Further, OPF rules a competition process among prototype samples, which are the most representative samples from each class. Therefore, a careful procedure to estimate them would be wise. Finally, in order to conquer samples, prototypes must offer them rewards, which are encoded by the path-cost function. In this paper, we are considering the OPF classifier proposed by Papa et al. [12,11], which employs a full connectedness graph, and a path-cost function that computes the maximum arc-weight along a path. For the sake of clarity, we shall refer to this version as OPF only.

Although OPF has obtained recognition results comparable or even more accurate than SVMs and ANNs in a number of different applications, as well as it has been usually much faster for training, it can be time-consuming for very large datasets. Although OPF is parameterless, its training phase takes $\theta(n^2)$, where n stands for the number of training samples. Truly speaking, this is not that bad, since SVMs usually require a considerably higher computational load. However, there is still room for improvements, and that is the main contribution of this paper: to introduce a different data structure that allows the OPF parallelization. As a matter of fact, the proposed approach is able to produce equivalent results to the ones obtained by original OPF classifier concerning accuracy, though up to five times faster using a simple personal computer hardware.

The remainder of this paper is organized as follows. Section 2 reviews OPF theoretical background, and Section 3 presents the modifications that led to the new parallel training algorithm. Section 4 discusses the experiments, and Section 5 states conclusions and future works.

2 Supervised Classification based on Optimum-Path Forest

Let $\mathcal{Z}$ be a dataset whose correct labels are given by a function $\lambda(x)$, for each sample $x \in \mathcal{Z}$. Thus, $\mathcal{Z}$ can be partitioned into a training ($\mathcal{Z}_1$), validation ($\mathcal{Z}_2$) and testing ($\mathcal{Z}_3$) set. Also, we can derive a graph $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{A}_1)$ from the training set, where $\mathcal{A}_1$ stands for an adjacency relation known as *complete graph*, i.e. one has a full connectedness graph where each pair of samples in $\mathcal{Z}_1$ is connected by an edge. Additionally, each node $\mathbf{v}_i^1 \in \mathcal{V}_1$ concerns the feature vector extracted from sample $x_i^1 \in \mathcal{Z}_1$. All arcs are weighted by the distance among their corresponding graph nodes. A similar definition can also be applied to the validation and test sets.

The OPF proposed by Papa et al. [12] comprises two distinct phases: (i) *training* and (ii) *testing.* The former step is based upon $\mathcal{Z}_1$, meanwhile the test phase aims at assessing the effectiveness of the classifier learned during the previous phase over the testing set $\mathcal{Z}_3$. Additionally, a *learning* algorithm was proposed to improve the quality of samples in $\mathcal{Z}_1$ by means of an additional set $\mathcal{Z}_2$. Roughly speaking, the idea is to train an OPF classifier over $\mathcal{Z}_1$ and then classify $\mathcal{Z}_2$.

Further, we replace non-prototype samples in $\mathcal{Z}_1$ by misclassified samples in $\mathcal{Z}_2$, and the very same process is executed once again (i.e. training over $\mathcal{Z}_1$ and classification over $\mathcal{Z}_2$). The above procedure is executed until the accuracy between consecutive iteration does not change.

2.1 Training

The training step aims at building the optimum-path forest upon the graph $\mathcal{G}_1$ derived from $\mathcal{Z}_1$. Essentially, the forest is the result of a competition process among prototype samples that ended up partitioning $\mathcal{G}_1$. Let $\mathcal{S} \subseteq \mathcal{Z}_1$ be a set of prototypes, which can be chosen at random or using some other specific heuristic. Papa et al. [12] proposed to find the set of prototypes that minimizes the classification error over $\mathcal{Z}_1$, say that $\mathcal{S}^* \subseteq \mathcal{Z}_1$. Such set can be found by computing a Minimum Spanning Tree $\mathcal{M}$ from $\mathcal{G}_1$, and then marking as prototypes each pair of samples (x_1, x_2), adjacent in $\mathcal{M}$, such that $\lambda(x_1) \neq \lambda(x_2)$.

Further, the competition process takes place in $\mathcal{Z}_1$, where nodes in $\mathcal{S}^*$ try to conquer the remaining samples in $\mathcal{Z}_1 \setminus \mathcal{S}^*$. Basically, such process is based on a reward-compensation procedure, where the prototype that offers the *minimum cost* is the one that will conquer the sample. The reward is computed based on a path-cost function, which should be *smooth* according to Falcão et al. [5]. Therefore, Papa et al. [12] proposed to use f_{max} as the path-cost function, defined as follows:

$$\begin{aligned} f_{max}(\langle s \rangle) &= \begin{cases} 0 & \text{if } \mathbf{s} \in \mathcal{S}^* \\ +\infty & \text{otherwise,} \end{cases} \\ f_{max}(\pi_s \cdot (\mathbf{s}, \mathbf{t})) &= \max\{f_{max}(\pi_s), d(\mathbf{s}, \mathbf{t})\}, \end{aligned} \tag{1}$$

where $\pi_s \cdot (\mathbf{s}, \mathbf{t})$ stands for the concatenation between path π_s and arc $(\mathbf{s}, \mathbf{t}) \in \mathcal{A}_1$. Also, a path π_s is a sequence of adjacent and distinct nodes in $\mathcal{G}_1$ with terminus at node $\mathbf{s} \in \mathcal{Z}_1$.

In short, by computing Equation 1 for every sample $\mathbf{s} \in \mathcal{Z}_1$, we obtain a collection of optimum-path trees (OPTs) rooted at $\mathcal{S}^*$, which then originate an optimum-path forest. A sample that belongs to a given OPT means it is more strongly connected to it than to any other in $\mathcal{G}_1$.

2.2 Testing

In the testing step, each sample $\mathbf{t} \in \mathcal{Z}_3$ is classified individually as follows: $\mathbf{t}$ is connected to all training nodes from the optimum-path forest learned in the training phase, and it is evaluated the node $\mathbf{s}^* \in \mathcal{Z}_1$ that conquers $\mathbf{t}$, i.e. the one that satisfies the following equation:

$$C(\mathbf{t}) = \arg\min_{\mathbf{s} \in \mathcal{Z}_1} \max\{C(\mathbf{s}), d(\mathbf{s}, \mathbf{t})\}. \tag{2}$$

The classification step simply assigns the label of $\mathbf{s}^*$ as the label of $\mathbf{t}$. Notice that a similar procedure to classify $\mathcal{Z}_2$ can be employed, too.

3 Parallel-driven Optimum-Path Forest Training

In this section, we present the proposed approach based on parallel programming to speed up the naïve OPF training algorithm, hereinafter called POPF. Since standard OPF makes use of a *priority queue* implemented as a binary heap, it does not support multiple access at the same time. Therefore, POPF uses a simpler data structure along with a slightly different (parallel) training process, which is based on three main assumptions, as discussed below.

The first observation concerns the optimum-path computation process for each $\mathbf{t} \in \mathcal{Z}_1$, which is independent to other samples. On the other hand, costs need to be updated on a data structure so that a new sample can be selected for the next iteration, in order to expand the optimum path computed already. For this purpose, LibOPF [13] uses a binary heap as suggested in [5]. However, such data structure is not prepared for concurrent updates, i.e. if one attempts to perform the computation of $f_{max}(\mathbf{t})$ for each $\mathbf{t} \in \mathcal{Z}_1$, a *mutex* would be required for each update process in the heap. However, this approach would not scale well if one increases the number of threads. Furthermore, this data structure introduces a $\mathcal{O}(\log(n))$ overhead in each update, where $n = |\mathcal{Z}_1|$.

The second observation concerns the graph, which is fully connected, implying that, at each iteration, all nodes need to be explored. Therefore, the computation of f_{max} for all $s \in \mathcal{Z}_1$ takes $\theta(n)^2$ operations in total. In order to overcome such quadratic complexity, we can implement the priority queue as a standard array, but exploring the set of nodes in parallel and performing a parallel linear-search. At each iteration, each thread $\delta_i, \forall i = 1, \ldots, m$, explores a subset $\mathcal{W}_{(s,i)}$, such that $\mathcal{W}_s = \mathcal{W}_{(s,1)} \cup \cdots \cup \mathcal{W}_{(s,m)}$ is the set of neighbors of s, thus performing two tasks[3]: (1) to update the costs of each $t \in \mathcal{W}_{(s,i)}$ according to f_{max} using arc $(\mathbf{s}, \mathbf{t})$, and (2) to compute the node $s^{(*,i)} \in \mathcal{W}_{(s,i)}$ with minimum cost for $\mathcal{W}_{(s,i)}$. Afterwards, the main thread finds the node s^* with minimum cost among all $s^{(*,i)}, \forall i = 1, \cdots, m$. Such node s^* will be the first one to come out of the priority queue in the next iteration. Therefore, by using m threads, the overall time complexity of the training algorithm is reduced to $\theta(n^2/m)$.

Finally, the third observation is related to the Prim's algorithm, which is used to calculate the Minimum Spanning Tree over $\mathcal{Z}_1$. As a matter of fact, we can use the very same OPF algorithm with a different path-cost function to compute the MST. Therefore, the aforementioned ideas can be applied to compute the MST too, taking advantage of parallelism in all the steps of the training process.

Algorithm 1 summarizes the ideas presented in this section. Note that even though parallelization takes place during the searching process for the best predecessor only, it is be better to start all threads only once at the beginning of the algorithm. The proposed approach was efficiently implemented using OpenMP [4], a well-known API for shared-memory parallel programming. OpenMP pragmas used in the implementation are included as comments.

[3] Notice $\mathcal{W}_{(s,i)}$ stands for the set of neighbours of node s in charge of thread δ_i.

Algorithm 1: POPF Training Algorithm

```
Input: A $\lambda$-labeled training set $\mathcal{Z}_1$, and number $m$ of threads.
Output: Optimum path forest $P_1$, cost map $C_1$, label map $L_1$ and cost-ordered
        set $Z'_1$
1  $Z'_1 \leftarrow \emptyset$
2  $\mathcal{S}^* \rightarrow SelectPrototypes(\mathcal{Z}_1)$
3  foreach $\mathbf{s} \in \mathcal{Z}_1 \setminus S$ do $C_1(\mathbf{s}) \leftarrow +\infty$
4  foreach $\mathbf{s} \in \mathcal{S}^*$ do $C_1(\mathbf{s}) \leftarrow 0$, $P_1(\mathbf{s}) \leftarrow nil$, $L_1(\mathbf{s}) \leftarrow \lambda(\mathbf{s})$
5  $\mathbf{s} \leftarrow$ any element from $\mathcal{S}^*$
6  in parallel for threads $i = 1, 2, \ldots, m$              # omp pragma: parallel
7  while $s \neq nil$ do
8      in main thread Insert $s$ in $Z'_1$                     # omp pragma: master
9      $s_i \leftarrow nil$
10     synchronization barrier                                # omp pragma: barrier
11     split among threads,                                       # omp pragma: for
12     foreach $t \in Z_1$ where $t \neq s$ do
13         if $C_1(t) > C_1(s)$ then
14             $cst \leftarrow \max\{C_1(s), d(s,t)\}$
15             if $cst < C_1(t)$ then $P_1(t) \leftarrow s$, $L_1(t) \leftarrow L_1(s)$, $C_1(t) \leftarrow cst$
16         if $s_i = nil$ or $C_1(t) < C_1(s_i)$ then $s_i \leftarrow t$
17     collect $\arg\min_{s_i} C(s_i)$ in $s$                 # omp pragma: critical
18     synchronization barrier
19 return classifier $[P_1, C_1, L_1, Z'_1]$
```

4 Experiments and Results

In this section, we present the methodology used to assess the robustness of the proposed approach, as well as the experimental results. Table 1 presents the description of the datasets used in this work, which were taken from UCI Machine Learning Repository [8]. We intentionally chose datasets with numeric features, to avoid extra pre-processing, and with different orders of magnitude, to better describe the scalability of our approach.

Table 1. Description of the datasets and percentages used for $\mathcal{Z}_1$, $\mathcal{Z}_2$ and $\mathcal{Z}_3$.

Dataset	Short Description	instances	features	classes	$\lvert\mathcal{Z}_1\rvert$	$\lvert\mathcal{Z}_2\rvert$	$\lvert\mathcal{Z}_3\rvert$
Letter Recog.	Character image features	20,000	16	26	20%	40%	40%
Shuttle	Distinguish shuttles	43,500	9	7	20%	40%	40%
MiniBooNE	Distinguish electron neutrinos	130,064	50	2	20%	40%	40%
SUSY	Distinguish supersymmetric particles	5,000,000	18	2	2%	2%	98%

We compared POPF against naïve OPF using a microcomputer equipped with a 3.1Ghz Intel Core i7 processor, 8 GB of RAM, and running Linux 3.16. The programs were compiled using GCC 5.0, which implements OpenMP 4 specification. Also, we varied the number of threads concerning POPF according to the maximum concurrency allowed by the processor. For each experiment, we executed a hold-out-based partition of the dataset over 10 executions for mean accuracy and computational load computation purposes.

Table 2 presents the results regarding execution time, number of learning iterations and classification accuracy for $\mathcal{Z}_3$ – as defined by Papa et al. in [12] –, where POPF-m stands for POPF executed with m threads. Clearly, POPF maintained OPF accuracy for all number of threads, meaning the classifier obtained through the proposed approach preserves the same properties of the original one. Only a slight variation concerning MiniBooNE dataset can be observed. This is explained by the fact that same-cost samples could be stored in a different order in $\mathcal{Z}'_1$, which will change the evaluation order during the classification process over $\mathcal{Z}_3$ and will assign a different class when ties occur.

Table 2. Comparison against OPF and POPF with different number of threads.

Dataset	**Technique**	**learning time (sec)**	**# iter.**	**Accuracy**	***S***	***E***
Let. Rec.	OPF	15.52 ± 0.14	7	95.64%	1	1
	POPF-2	9.21 ± 0.08	8	95.64	1.69	0.84%
	POPF-4	5.07 ± 0.01	8	95.64	3.06	0.77%
	POPF-8	3.42 ± 0.02	8	95.64	4.54	0.57%
Statlog	OPF	31.45 ± 0.32	4	95.69%	1	1
	POPF-2	16.57 ± 0.15	4	95.69%	1.90	0.95
	POPF-4	9.03 ± 0.09	4	95.69%	3.48	0.87
	POPF-8	5.98 ± 0.08	4	95.69%	5.26	0.66
Miniboone	OPF	$1,442.96 \pm 6.66$	8	79.37%	1	1
	POPF-2	755.20 ± 19.04	8	79.50%	1.91	0.96
	POPF-4	438.81 ± 9.16	8	79.50%	3.29	0.82
	POPF-8	273.95 ± 7.68	8	79.50%	5.27	0.66
SUSY	OPF	$13,149.8 \pm 50.0$	9	66.55%	1	1
	POPF-2	$6,398.9 \pm 32.5$	9	66.55%	2.06	1.02
	POPF-4	$3,784.3 \pm 21.2$	9	66.55%	3.47	0.87
	POPF-8	$2,592.8 \pm 115.7$	9	66.55%	5.07	0.63

In Table 2 we also include parallel performance measures: speedup (S) – measuring gain in running time – and efficiency (E) – measuring thread utilization. They are defined as follows [9]:

$$S = \frac{T_s}{T_p} \qquad \text{and} \qquad E = \frac{S}{m} = \frac{T_s}{m \cdot T_p}, \tag{3}$$

where T_s and T_p stand for the execution time of traditional and parallel OPF, respectively, and m denotes the number of threads.

We can observe that maximum speedup is obtained using 8 threads, being about five times faster than traditional OPF. Furthermore, speedup improves when the size of the dataset increases. Another worth noticing observation is that for the largest dataset, when using 2 threads, the efficiency obtained was greater than 100%. This confirms that POPF is considerably more efficient than traditional OPF, not only because of the parallel implementation, but also due to its asymptotic improvement. Figure 1 presents charts for S and E.

Regarding the overall parallel efficiency, it is important to stress the efficiency results obtained for both 4 and 8 threads. On one hand, we obtained an efficiency between 77% and 87% considering 4 threads, which is an outstanding result for any parallel implementation. On the other hand, the efficiency considering 8 threads was between 57% and 66%, which is a good thread utilization considering the fact that the processor used has only 4 physical cores (implementing 8 threads with HyperThreading® technology).

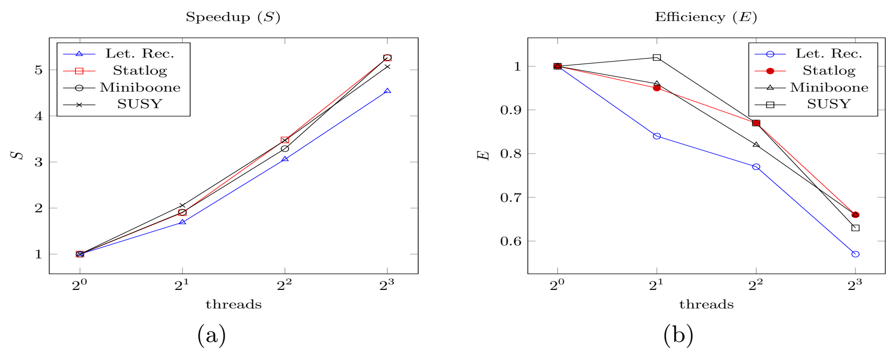

Fig. 1. Parallel performance measures for POPF learning process: (a) Speedup (S); and (b) Efficiency (E).

5 Conclusions and Future Work

In this work, we were able to parallelize the OPF training algorithm, and we demonstrated its efficiency concerning classification tasks. This new approach is based on three important observations: (i) the optimum-path computation process for each training sample is independent to each other; (ii) the full connectedness training graph allows us to replace the binary heap by a simple list (suitable for parallelization); and (iii) the computation of the MST during the training phase can also be performed in parallel. These changes allow to reduce the asymptotic complexity of the implementation and also turns the parallelization feasible.

We have observed that POPF preserves the accuracy of the original algorithm, but it is able to perform the learning phase at least five times faster using commodity hardware. Thus, an OPF with hundreds of thousands of nodes can be calculated in less than an hour. As such, POPF allows to perform classification of very large datasets when timing restrictions are present, and it brings closer the possibility of performing nearly real-time classification for reasonable sized-datasets even on a single computer or mobile device.

However, such real-time implementation still needs improvements in the classification algorithm. Thus, we are considering the use of spatial data structures to index the optimum path-forest obtained during training, so that a fewer amount of nodes are considered in classification, thus improving its running time.

Acknowledgments

The authors would like to thank Capes PROCAD grant 2966/2014, CNPq grants #306166/2014-3 and #470571/2013-6, FAPESP grant #2014/16250-9 and Universidad Católica San Pablo (UCSP) for their support.

References

1. Amorim, W.P., Falcão, A.X., Carvalho, M.H.: Semi-supervised pattern classification using optimum-path forest. In: 27th SIBGRAPI Conference on Graphics, Patterns and Images. pp. 111–118 (2014)
2. Amorim, W.P., Falcão, A.X., Papa, J.P., Carvalho, M.H.: Improving semi-supervised learning through optimum connectivity. Pattern Recognition (2016), `http://www.sciencedirect.com/science/article/pii/S0031320316300668`, (to appear)
3. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning 20(3), 273–297 (1995)
4. Dagum, L., Enon, R.: OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE 5(1), 46–55 (1998)
5. Falcão, A.X., Stolfi, J., de Alencar Lotufo, R.: The image foresting transform: Theory, algorithms, and applications. Pattern Analysis and Machine Intelligence, IEEE Transactions on 26(1), 19–29 (2004)
6. Haykin, S., Network, N.: A comprehensive foundation. Neural Networks 2(2004) (2004)
7. Haynes, S.D., Stone, J., Cheung, P.Y.K., Luk, W.: Video image processing with the sonic architecture. Computer 33(4), 50–57 (Apr 2000)
8. Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
9. Pacheco, P.: An introduction to parallel programming. Elsevier (2011)
10. Papa, J.P., Falcão, A.X.: A new variant of the optimum-path forest classifier. In: Advances in Visual Computing, Lecture Notes in Computer Science, vol. 5358, pp. 935–944. Springer Berlin Heidelberg (2008)
11. Papa, J.P., Falcão, A.X., De Albuquerque, V.H.C., Tavares, J.M.R.: Efficient supervised optimum-path forest classification for large datasets. Pattern Recognition 45(1), 512–520 (2012)
12. Papa, J.P., Falcao, A.X., Suzuki, C.T.: Supervised pattern classification based on optimum-path forest. International Journal of Imaging Systems and Technology 19(2), 120–131 (2009)
13. Papa, J., Falcão, A., Suzuki, C.: LibOPF: A library for the design of optimum-path forest classifiers (2014), software version 2.1 available at `http://www.ic.unicamp.br/~afalcao/LibOPF`
14. Rocha, L.M., Cappabianco, F.A.M., Falcão, A.X.: Data clustering as an optimum-path forest problem with applications in image analysis. International Journal of Imaging Systems and Technology 19(2), 50–68 (2009)