Original Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–15

DOI: 10.1177/1094342016678302
hpc.sagepub.com
SAGE

Deploying massive runs of evolutionary algorithms with ECJ and Hadoop: Reducing interest points required for face recognition

Francisco Chávez[1], Francisco Fernández[1], Daniel Lanza[2], César Benavides[3], Juan Villegas[4], Leonardo Trujillo[5], Gustavo Olague[6] and Graciela Román[3]

Abstract

In this paper we present a new strategy for deploying massive runs of evolutionary algorithms with the well-known Evolutionary Computation Library (ECJ) tool, which we combine with the MapReduce model so as to allow the deployment of computing intensive runs of evolutionary algorithms on big data infrastructures. Moreover, by addressing a hard real life problem, we show how the new strategy allows us to address problems that cannot be solved with more traditional approaches. Thus, this paper shows that by using the Hadoop framework ECJ users can, by means of a new parameter, choose where the run will be launched, whether in a Hadoop based infrastructure or in a desktop computer. Moreover, together with the performed tests we address the well-known *face recognition* problem with a new purpose: to allow a genetic algorithm to decide which are the more relevant interest points within the human face. Massive runs have allowed us to reduce the set from about 60 to just 20 points. In this way, recognition tasks based on the solution provided by the genetic algorithm will work significantly quicker in the future, given that just 20 points will be required. Therefore, two goals have been achieved: (a) to allow ECJ users to launch massive runs of evolutionary algorithms on big data infrastructures and also (b) to demonstrate the capabilities of the tool to successfully improve results regarding the problem of face recognition.

Keywords

ECJ, face recognition, Hadoop, parallel evolutionary algorithm

1 Introduction

Real life complex optimization problems, such as the one we are interested here, face recognition, are perfect candidates for evolutionary algorithms (EAs), and new parallel hardware provides the means for addressing those problems. Moreover, when the main goal for the EA is to evolve new approaches to improve previous solutions, the need for parallel and distributed infrastructures is a must, given the long time required to reach a solution.

If we focus on the real life problem we are interested in, we may consider standard approaches to face recognition. As we will later see, available techniques typically rely on a number of *interest points*—around 60 of them—that are located on human faces and are analyzed and compared with those contained in available databases to correctly identify those faces. The larger the set of analyzed points, the longer the computing time required to reach a match.

We are especially interested in this problem here not just to improve the computing time of a specific approach. We would like to see if a general

[1]Department of Computer Science, University of Extremadura, Spain
[2]CERN (European Organization for Nuclear Research)
[3]Universidad Autónoma Metropolitana, Departamento de Ingeniería Eléctrica, México
[4]Universidad Autónoma Metropolitana, Departamento de Electrónica, México
[5]Instituto Tecnológico de Tijuana Calzada Del Tecnológico S/N, México
[6]CICESE, México

Corresponding author:
Francisco Chávez, University of Extremadura, C/. Santa Teresa de Jornet, 38, CP 06800, Mérida, Spain.
Email: fchavez@unex.es

improvement can be reached by reducing the number of interest points required for face recognition. Yet, such an approach requires a meta-learning process: the best IPs must first be identified, and then a second training step will allow a given algorithm to employ those IPs to develop a face recognition algorithm.

Today, open source efficient solutions supporting big data approaches are an option, but several factors keep them out of the selections made by EA researchers: they frequently rely on sequential versions of the algorithms, mainly because of the difficulty for migrating software tools to parallel and distributed environments and also due to the typical resistance by users to change to new software (Stelzer and Mellis, 1998).

This paper is built on a previous work where a MapReduce version of a well-known EA tool is described (Chávez et al., 2016). In this paper we describe on the one hand, the real life problem we are interested in, improving face recognition by identifying a reduced set of facial interest points. On the other hand, we develop a new version of the ECJ based tool that makes use of MapReduce; we analyze performance of the tool and then apply it to the problem we are interested in, being thus able to notably reduce the number of interest points required for face recognition.

Therefore the main contribution of this paper is the application of a massively parallel EA, based on the MapReduce approach to the selection of best IPs for the human face recognition problem. Results show how the initial set of 60 IPs is reduced to just 20, while accuracy is maintained.

The rest of the paper is organized as follows: Section 2 presents the problem we address and the need for parallel EAs; in Section 3 we provide a description of the ECJ integration with Hadoop and describe the methodology applied. Afterwards, in Section 4, we show the results obtained. Finally, our conclusions are drawn in Section 5.

2 The problem of face recognition

2.1 ECJ and Hadoop

ECJ is one of the best known EA tools, developed in java. ECJ offers parallel models, such as the island model that can be run on top of desktop multicore systems and clusters. Nevertheless, if there is a need for using a different infrastructure, several changes must be applied to the tool. This was the case in our recently published Desktop Grid version of the tool (Fernández et al., 2014). Yet we are interested here in more reliable infrastructures, such as clusters, that may be used in connection with big data approaches.

In this work, we thus follow an alternative path, while trying to improve the ECJ project by adapting it to the MapReduce model (Dean and Ghemawat, 2008, 2010). Also, another important goal is that internal changes should not affect the way researchers use the tool, so as to prevent any drop in utilization by the people already using the tool: they must be able to launch massive runs on big data infrastructures in the same easy way they run the sequential version of the algorithm.

MapReduce is a model deriving from the functional programming paradigm designed to compute massive commands in its inherent parallel way of describing actions. The idea is to apply *map* operations massively in a parallel way to a large amount of data, and then employ the *reduce* operation to unify and extract conclusions.

Interest in this model has allowed for the development of a number of software tools and frameworks embodying its principles. One of the best known open source initiatives has been developed by the Apache foundation and is called Hadoop (White, 2012; Shvachko et al., 2010).

Apache Hadoop includes an ecosystem to massively process large amounts of data by means of a simple programming model. Hadoop easily scales and can be run within a single computer or on up to several thousand processors. The framework is fault tolerant, thus providing high reliability features. Hadoop provides: (a) a high performance distributed file system (HDFS) so huge amounts of data can be easily shared and accessed by every computer node; (b) tools for managing and balancing work units; and finally (c) the MapReduce programming model.

In this work, we present an enhancement to the ECJ tool based on MapReduce so that the EA community can easily use this kind of technology. Although several parallel and distributed models of EAs could be considered, we focus here on the simplest one, especially designed for the simultaneous evaluation of a number of individuals from the population. Therefore, the idea is that the fitness function can be computed as a Hadoop task; hence, at every generation all fitness evaluations can be managed by the proposed tool and run through the MapReduce model. As we will see later on, the model can be successfully applied to computational intensive problems, such as the one we are more interested in here, related to the selection of IPs in machine learning based face recognition problems.

Face recognition is a basic task used daily by humans in all kind of activities that is still daunting to researchers because of the ease with which it is performed naturally despite and the difficulty of automating it within a computer. Indeed, developing a computational system exhibiting the abilities of humans for face recognition could have multiple purposes such as: biometric authentication for public safety, such as the case of border protection or restricted access; multimedia entertainment and communication, such as the case of data associated with

individuals; and so on. Historically, the development of algorithms for pattern recognition has been connected with the task of face recognition (O'Toole, 2013), since these algorithms offer mathematical and computational models useful in the understanding of such problems. Normally, the endeavor of finding novel features associated with the main facial landmarks is a crucial step in these algorithms. The discovery of prominent features depends upon the relationship between the way in which humans perform the task of face recognition and how the same task is achieved by computers. In particular, the measurement and discrimination that a machine needs to perform the calculation at the moment of evaluating the human face is considered of paramount importance by researchers (Srinivasan and Balamurugan, 2014). Nowadays, there are a number of proposals that have been developed with the aim of solving the automation of this difficult problem.

One idea is to state the face recognition problem in terms of special landmarks called interest points (IPs), which are considered powerful local invariants like the well-known SIFT descriptor (Lowe, 2004), and there are actually some machine designs related to IPs (Trujillo and Olague, 2008; Olague and Trujillo, 2012), as well as to the SIFT descriptor (Perez and Olague, 2013). This idea of computing IPs was actually applied to the problem of facial expression recognition with great results (Trujillo et al., 2005). That proposal is related to the analysis of textures around IPs based on the idea that local information about IPs is employed in defining facial traits (see Figure 4, below). This principle has been extensively used within in content-based image retrieval (CBIR), which has been used in the case of image recognition as applied to biomedical images (Uwimana and Ruiz, 2008) and for natural image landscapes (Serrano et al., 2013). Also the CBIR approach has been used in the face recognition problem with the requirement that at least a set of 60 IPs were needed (Benavides et al., 2015). This set is used to describe the neighborhood around each facial landmark, which is then used to build the information through three statistical traits (average, variance and homogeneity) for each image band. This approach uses as many points as specified by the template that is then used in each database. Thus, if the number of IPs is increased the total computational effort will also increase during learning. The advantage is the fact that a 100% recognition rate is achieved with this methodology. Nevertheless, the automation could be greatly improved if the number of IPs was reduced by some procedure without degrading performance.

Our approach relies on EAs, and we will apply them not just to train a given algorithm on an available database, but to evolve an appropriate set of points, so that the learning process can then be shortened. But this is time consuming: each of the EA individuals within the population will be *by itself* a face recognition algorithm working with a different set of points. Thus every generation will need to evaluate a number of algorithms' instances, each of them working with a different set of points, and the evolutionary approach must lead us not only to the classifier, but also to the appropriate set of points. The mechanism is thus so time consuming that big data approaches in combination with EAs are required. We are thus interested in applying a well-known EA tool in connection with big data infrastructures, and this is the reason why ECJ has been chosen, although a number of modifications and improvements must be introduced to be able to run as desired.

3 Methodology

The main idea is to merge ECJ and Hadoop so that all fitness evaluations are deployed as working units in the MapReduce model. This will allow the system to face a real-life problem where the fitness function takes a long time and/or a large number of individuals are included within the population. Although the specific problem we are addressing does not need a large population, every fitness evaluation—a machine learning process by itself—requires so much computing time that the approach developed perfectly matches the problem.

However, we consider the possibility of including not just a single individual per working unit but a number of them to reduce latencies related to communication processes. According to the MapReduce model the fitness function is the task to be distributed so the main evolutionary loop should be changed accordingly.

After reviewing the ECJ source code, we noticed that several functions were already available providing check pointing facilities. Hence, the population can be stored when required, freezing the process and continuing it later by simply loading the previously saved *serialized file*. This is the main operation on which we build the new functionality, given that the saved file can be accessed on the HDFS by many *mapper* processes, and all could share the state of the EA process and access the set of individuals to be evaluated. We must nevertheless take into account that the serialized information does not include the population of individuals: that must be saved in a different file. Figure 1 graphically depicts how Hadoop works with the *master* process, through the main loop, and illustrates the *slave* layer in charge of launching the map works, called *TaskTrackers*.

Once the check pointing process saves the required information, a file containing all individuals to be evaluated is defined and used as the input to the first task that Hadoop will launch. This first task will then produce as many new *mapper* tasks as required to evaluate all the population. Every *mapper* will then read information from the previously saved problem, shared

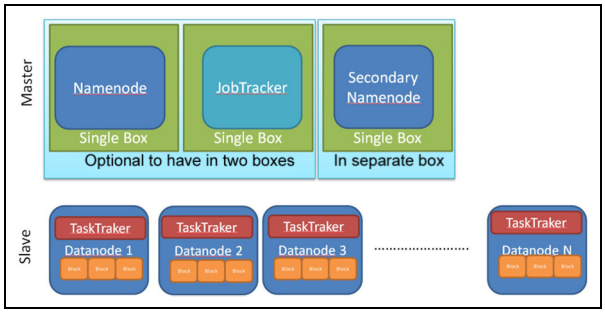

Figure 1. Layers of Hadoop, distributed in a master processor and a number of slave ones.

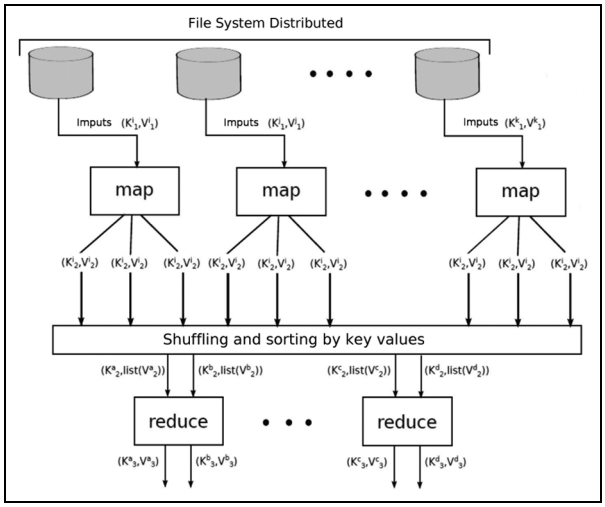

Figure 2. Map and reduce processes in charge of data processing.

through the file system to load the set of individuals to be evaluated. We delegate on Hadoop the load balancing strategy to manage the way work units are distributed along the computing nodes. Figure 2 shows the relationship between the map and reduce processes for ECJ.

A map process requires a series of input/output parameters, namely a key–value pair. In this case, the key–value pair will be the index for the individual to be evaluated and its assigned value. As output, the value will be provided by the fitness function. Once the work is done, all fitness values are gathered, finishing the map step, so that the algorithm can continue within the master process.

All described changes have been applied in a new version named *ECJ + Hadoop*, sometimes providing new functions, such as *ec.hadoop.HadoopEvaluator*. This new process will be in charge of computing the fitness values. Algorithm 1, given as an example, includes the details for the map process that will be run simultaneously in every processor.

Any ECJ user knows that for any new problem to be solved by the tool, it is typically enough to

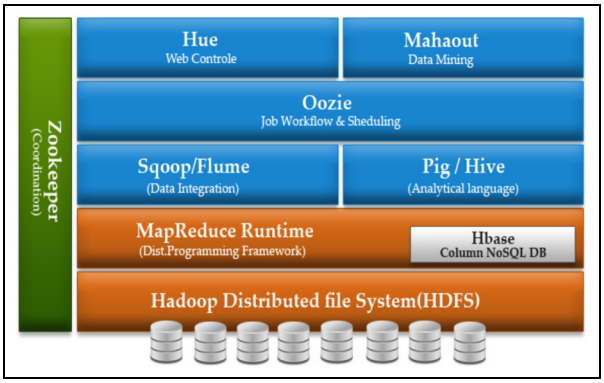

Figure 3. Tools integrated in Cloudera.

implement the fitness function and select the parameter values. Thus, according to the new model described, and given that we want all of the process to be transparent to users, a user will only have to provide the fitness function, and setup the *eval* parameter to the new available value, *ec.hadoop.HadoopEvaluator*. Other parameters are:

- main folder for the serialized file (parameter name: eval.hdfs-prefix, default value: ecj_work_folder_hc);
- address for the JobTracker (parameter name: eval.-jobtracker-address, default value: localhost);
- JobTracker port (parameter name: eval.jobtracker-port, default value: 8021);
- HDFS address (parameter name: eval.hdfs-address, default value: localhost);
- HDFS port (parameter name eval.hdfs-port, default value: 8020).

As a result, the tool can easily implement and deploy any optimization problem that requires a large amount of data. Of course, it is desirable to run this new version of the tool on a cluster of computers. In the experimental stage described next, we employed Cloudera (2015).

3.1 System deployment

Among different possibilities for setting up a Hadoop infrastructure, we decided to use Cloudera (2015). Cloudera is a single platform for big data projects integrating Hadoop among other tools. It allows for centrally management and monitoring of the hardware infrastructure on which it is installed. It allows for both running parallel processes and also sharing a distributed storage system. Cloudera provides a quick and easy installation of all required elements to use Hadoop, and includes an entire ecosystem of tools that facilitate managing tasks, scheduling and the monitoring of jobs. Figure 3 shows the toolkit that accompanies Cloudera Hadoop when installed.

Among the tools included in Cloudera we find:

Time (minutes) for evaluating a population of individuals.

Algorithm 1: ec.hadoop.HadoopEvaluator algorithm

```
import java . io . IOException;
import org . apache . hadoop . mapreduce . Mapper;
import ec . EvolutionState;
import ec . Individual;
import ec . hadoop . writables.FitnessWritable;
import ec . hadoop . writables.IndividualWritable;
import ec . hadoop . writables.IndividualIndexWritable;
import ec . simple . SimpleProblemForm;
public class EvaluationMapper extends
   Mapper<IndividualIndexWritable, IndividualWritable, IndividualIndexWritable, FitnessWritable> {
      public static EvolutionState state;
      @Override
      protected void map(IndividualIndexWritable key, IndividualWritable value, Context context)
         throws IOException, InterruptedException {
            state = value . getEvaluationState();
            Individual ind = value.getIndividual();
            //Evaluate individual
            SimpleProblemForm problem = ((SimpleProblemForm) state . evaluator . p\_problem);
            Problem . evaluate ( state, ind, key. getSubpopulation (), 0);
            //Emit key and value context. write( key, new FitnessWritable(state, ind. fitness));
      }
}
```

Table 1. Infrastructure used to run Cloudera.

Node	RAM (Gb)	HD (Gb)	Cores	Roles Cloudera
1	16	75.5	8	13
2	16	74.7	4	9
3	16	74.7	4	10
4	16	141.4	8	7
5	16	75.5	8	5
6	16	75.5	8	6

- Pig: high level data-flow language to facilitate MapReduce programming;
- Hive: provides a SQL interface with data stored in the HDFS;
- Oozie: workflow planner to manage Hadoop jobs;
- Hue: web interface to simplify the use of Hadoop;
- HBase: distributed non-relational database (NoSQL) running on the HDFS (inspired by Google BigTable);
- Sqoop: allows for efficient transfer of data between Hadoop and relational databases;
- ZooKeeper: centralized configuration, named, distributed and synchronization services groups for large distributed file systems;
- Flume: collection, aggregation and movement of large log files to the HDFS;
- Mahout: scalable machine learning algorithms and data mining on Hadoop.

We configured Cloudera on a hardware infrastructure consisting of six blade servers divided into two blades with four cores each and four blades with eight cores each. The main features of each server are described in Table 1.

Once Cloudera is deployed on the cluster, a cache memory and distributed storage through the HDFS are shared among all computing nodes. This allows us to view the system as a single unit, where data is distributed and can be accessed by all processes deployed in the cluster.

3.2 A real life problem: Face recognition as a pattern recognition application

The problem we want to approach consists of applying an EA capable of deciding which are the best IPs for the learning process. This meta-learning algorithm will employ a genetic algorithm (GA) with a binary chromosome: each bit describes whether a certain IP will be employed during the training and testing processes of the algorithm. A number of databases are available

Table 2. Dataset features.

db	AR	Ex.Yale B	Yale	Bio ID	MUCT
N. images	4,000	16,128	165	1,521	3,755
People	126	27	15	23	276
C/GS	C	GS	GS	GS	C
Type	JPG	PGM	GIF	PGM	JPG
Resolution	768×576	640×480	320×243	648×286	640×480

with hundreds images for the learning step. In this work, we have employed the datasets with the features shown in Table 2 (Fei-Fei et al., 2007). Each image from the database includes meta information known as *IP coordinates*, which are located at different locations of the face.

The fitness function will check whether the algorithm is capable of learning and hence correctly classifying the faces, based only on the IPs selected by the GA, while the learning algorithm performs CBIR as described in Benavides et al. (2015). The training–testing process applied to the individuals must perform the computation of the fitness value, this problem is computationally intensive and the reason for selecting it to test on ECJ + Hadoop.

Each individual provides a set of IPs. The learning algorithm must then employ the faces of the training set from the database to extract the features by CBIR around the points for training. Then, the images of the test set are used to check the accuracy of the algorithm. The first model of the fitness function considered only the retrieval percentage of the classifier for the associated individual, as described in Benavides et al. (2015). A priori, we know that the full set of IPs achieves a 100% recognition rate, but in that model the number of IPs was not included to penalize the fitness results. Therefore, we plot a new fitness function which is defined by equation (1)

$$f = 1 + c - \frac{\%IP}{\%PC} \tag{1}$$

where $\%IP$ is the percentage of IPs of the selected individual and $\%PC$ is the percentage of classification achieved by the selected individual. Moreover, $\%IP = (NP \times 100)/TP$, where NP is the number of selected IPs in the chromosome and TP is the total number of selected IPs of the template using the database and according to Figure 4, while $c = 0.187$ is a penalization factor applied when a minimum of 10 points is reached, experimentally chosen from the images data sets, whose value ranges between 0.15 and 0.187 as a percentage according to the total IP landmarks per data set. The fitness function must apply a number of image processing operations to convert images to the HSI color space, compute statistical values and generate a descriptor vector, which will be the

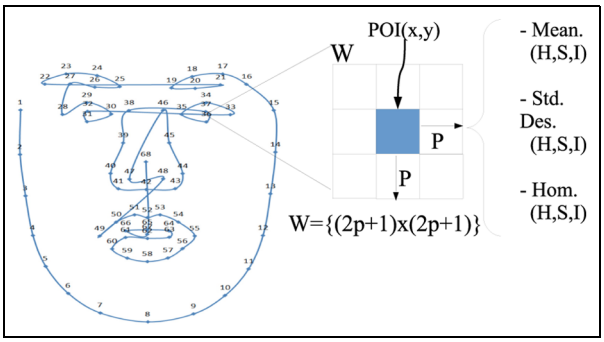

Figure 4. Interest points used for the face recognition problem, and the considered neighbors around each IP for CBIR implementation.

output for each image. An overview of the described evolutionary process when the evaluation is done using MapReduce jobs is shown in Figure 5. For a deeper understanding, a detailed view of the evaluation process is shown in Figure 6, where the process is performed by running two MapReduce jobs. Firstly, a training job (a) will run. In this job, images will be processed by a set of map tasks, and then a final reduce task will take outputs from all map tasks and generate a normalized matrix that is finally used by a *k-Means* algorithm to compute the final matrix. This matrix will be the output of the reduce phase, and so, the result of the training job. After the training job has finished, a query job will run (b). In this job, following similar operations as in the training job, the algorithm tries to recognize the faces based on the knowledge acquired in the previous job (the final matrix). Results obtained are checked with the classes stored into the image database (c), and the true positive percentage is used as the final fitness value, which is then stored into the HDFS. Once all jobs have finished computation, the evaluation phase of the EA concludes and the algorithm can continue to the next step.

As can be seen, the algorithm is computationally intensive, given that each of the individuals of the population is a particular instance of an algorithm that must be completely trained by means of the databases available, and then analyzed as to whether the IPs employed have been sufficient for obtaining good face recognition rates. Given that the EA must be run for a number of

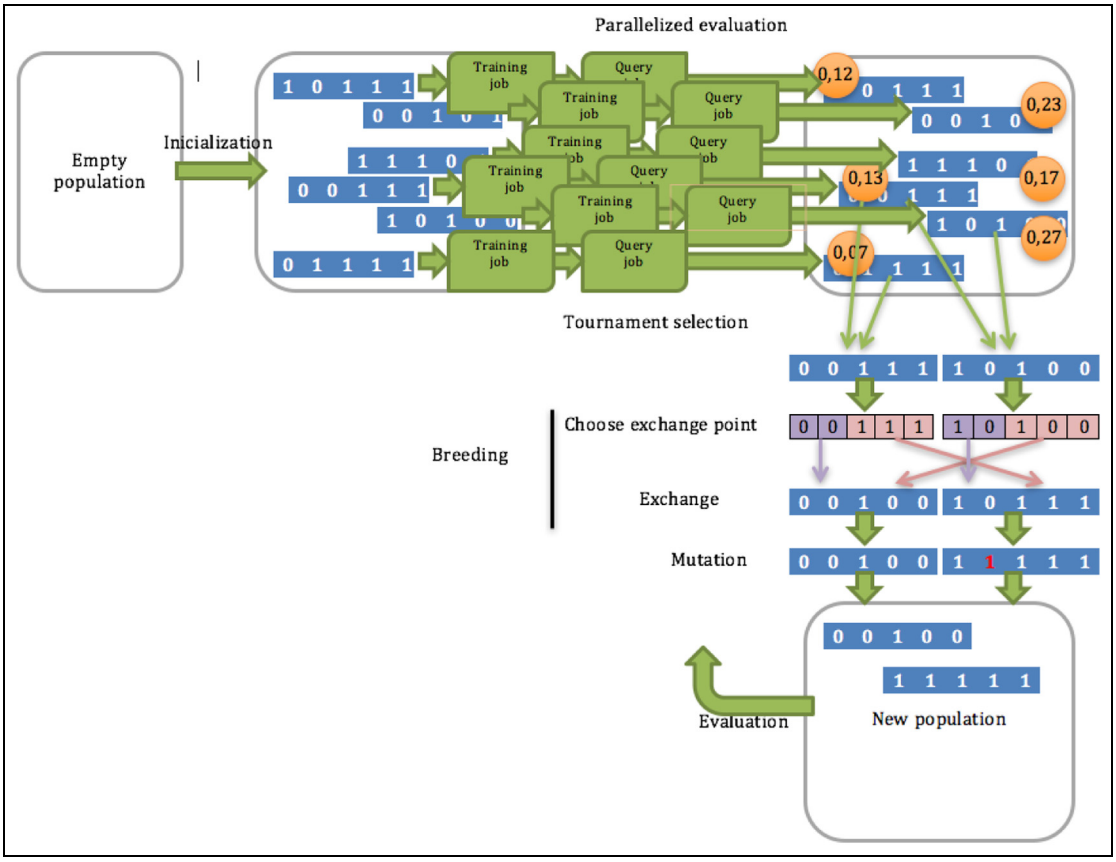

Figure 5. Overview of the evolutionary process when the evaluation is done using MapReduce jobs.

generations and each of the generations includes a set of individuals, the computing time will be large, so the parallel version of the algorithm described above is a perfect candidate to solve the problem at hand.

4 Experiments and results

We performed a series of experiments that allowed us to apply a tuning process to the ECJ + Hadoop tool and also check the advantages of the approach when it is applied to the problem at hand. We describe first the performance analysis for the new ECJ + Hadoop tool, using a standard GP benchmark problem for the testing step, and then the results obtained for the face recognition problem.

4.1 Analysing performance

In order to analyze the performance of this MapReduce version of ECJ, the first series of experiments consisted of running one of the benchmark problems presented by Koza (1992) named *Even Parity*. This problem is considered one of the benchmarks for genetic programming. We have run evenp-12, instead of the more standard five bits version, with the idea of making it difficult enough to be able to justify with large populations during a number of generations.

Note that we are not interested in the problem itself. We simply employ it to study how large populations might be used while considering the latencies that arise due to population handling during the process of creating packages of individuals to be distributed, fitness values collection, application of centralized genetic operators, and so on. The goal is to analyze the performance of the approach to identify ideal situations where ECJ + Hadoop is of interest.

Although many possibilities exist to analyze the results, we decided here to compare our results with the standard ECJ tool which allows us to run the problem using multithread capabilities. The base case comparison is thus a single computer with up to eight threads.

Therefore, we started running the problem with different population sizes: from 30,000 to 1,500,000 individuals. Larger sizes of the population were only employed with a larger number of cores. Table 3 displays a summary of the experiments run on a single computer. As can be seen, the larger the number of cores, the shorter the time required to evaluate a single generation.

Next we ran similar experiments using EJC + Hadoop and compared the results with those

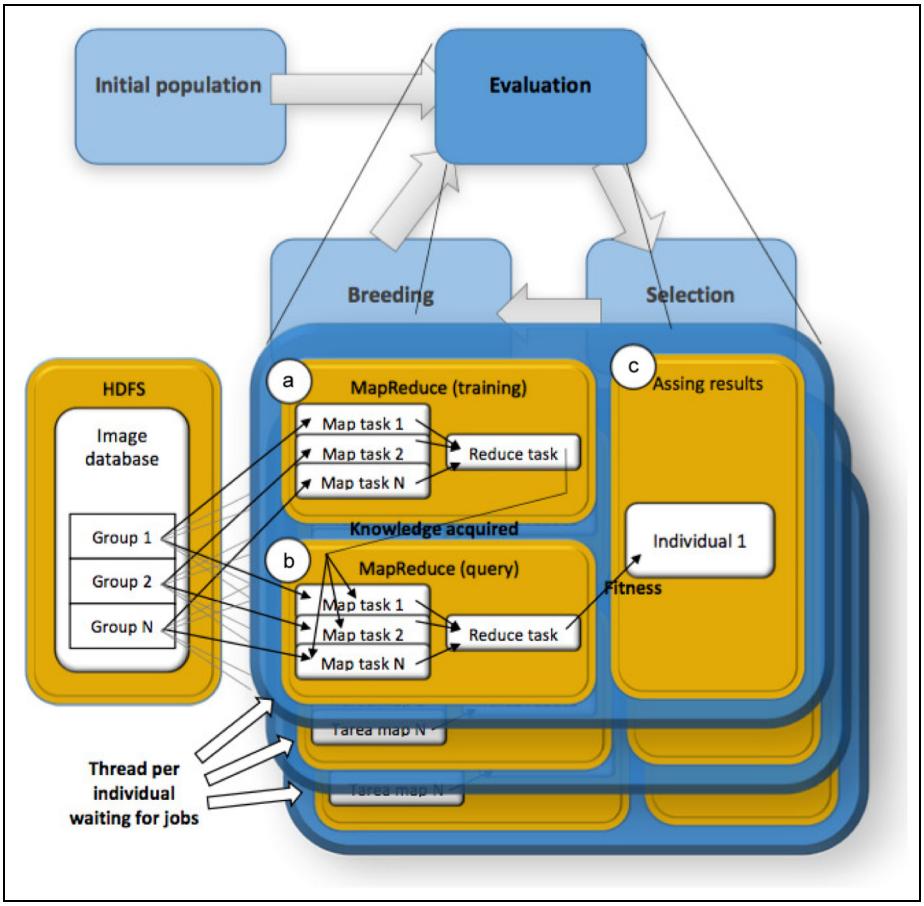

Figure 6. Detailed view of the evaluation phase and the data flow between jobs and tasks. (a) MapReduce job for training; (b) MapReduce job for recognizing faces; and (c) storing computed fitness on the HDFS.

Table 3. Experiments on a standard PC with eight threads. Fitness evaluation time in seconds for a generation (individuals $\times 10^3$).

Threads /Individuals	30	50	100	300	500	1000	1500
1	19	31	65				
2	10	16	32				
4	5	8	16	47	82		
8	3	4	8	25	41	87	130

obtained with the ECJ tool. When running Hadoop based processes we had to take into account some parameters that highly influence performance, and some values whose analysis allow us to understand the behavior of the whole system:

- HDFS block size (sz-HDFS) is a configurable parameter that allows us to manage the number of data blocks in the HDFS, allowing greater parallelization of *mappers* created by the *JobTracker*.
- Time to store the population in the HDFS (t-wrt) is a measure that tells us how long will it take to write the population in the distributed file system.
- Average task time (t-task) shows the average execution time for each task.
- Tasks are the number of tasks launched for each job. This number depends on the workload given by the population size and of the used HDFS block size.
- Evaluation time (t-eval) is the total time to evaluate a population.

Table 4 shows an experiment summary with different configurations, while providing parameters and measures.

In the case of considering a small number of individuals, and the short time required to evaluate each of

Table 4. ECJ + Hadoop results with different high performance distributed file system (HDFS) blocks and populations sizes, using 40 cores (time in seconds).

sz-HDFS	sz-Pob	t-wrt	t-task	Tasks	t-eval
14	3×10^6	22	45	55	141
	2.5×10^6	18	44	46	132
	2×10^6	14	46	36	123
	1.5×10^6	10	45	27	101
	1×10^6	18	40	18	84
10	2.5×10^6	18	40	64	174
	2×10^6	14	38	50	111
	1.5×10^6	11	38	38	94
	1×10^6	7	38	25	64
	0.75×10^6	6	30	19	58
	0.5×10^6	4	30	12	52
5	2.5×10^6	18	25	127	132
	2×10^6	14	22	101	101
	1.5×10^6	11	22	75	92
	1×10^6	8	24	49	73
	0.75×10^6	6	22	37	60
	0.5×10^6	4	21	25	40
3	2.5×10^6	19	20	211	140
	2×10^6	14	18	167	109
	1.5×10^6	11	20	124	103
	1×10^6	8	17	82	61
	0.5×10^6	4	17	41	44
1.25	1×10^6	8	11	196	70
	0.5×10^6	4	11	97	43
	0.3×10^6	2	11	58	31
	0.1×10^6	1.5	8	19	24
	1×10^4	1	7	10	19
	3×10^4	0.6	7	6	19
1	3×10^4	2	10	72	34
	1×10^4	1.5	8	24	25
	5×10^3	1	6	12	19
	3×10^3	0.6	6	7	19

them, the experimental results using ECJ + Hadoop were discouraging. The test provides poor performance when compared with the standard multicore approach, so we decided to further explore some of the elements that could influence the computing time.

While continuing to use the checkpointing feature in order to store the current state of the evolutionary process, we removed the population from it and saved into a separated file which is then used as the input for the MapReduce job. Individuals are serialized and directly saved within the HDFS. Previously, for storing individuals, text format was used; however, a serialized model saves time and resources when saving individuals due to the shorter length of the serialized model. The impact of this improvement is large due to the way in which the HDFS must save the information on a number of devices. Moreover, we employed compression processes to save read/write operation time. Also, given the large size of the populations that we are managing, we decided to enlarge the buffers employed for transferring information, which also influences the transfer rates.

We must also consider that different processes running on nodes are held in Java virtual machines (JVMs) and with each new generation these virtual machines are created and destroyed, which is time consuming. However, Hadoop has a parameter to avoid this behavior and allow the JVMs to be reused, saving time. This feature was enabled, and an improvement was observed in processing times.

Figure 7 shows the time required for different population sizes when using the standard runs of ECJ considering different numbers of cores. The red, green, purple and yellow lines show the result of the experiments on a single computer using one, two, four and eight threads respectively, using the standard ECJ tool. We can observe in these results that the time scales linearly with the size of the population making it easy to predict the behavior of any number of individuals. Also, the relationship between the number of cores can be easily appreciated. We did not use a higher number of threads since the used hardware did not have the proper architecture to produce reasonable results with more threads: the processor supported a maximum of eight threads running in a parallel fashion. In further comparisons, the best results from local executions that were obtained with eight threads were used in order to determine if the developed Hadoop based solution makes sense depending on the population size.

Table 5. Time (minutes) for evaluating a population of individuals.

ECJ Tool/Individuals	10	20	30	40	50	60
Std sequential	44.5	89	133.5	178	222.5	267
Std multi-threading	10.7	21.4	32.1	42.8	53.5	64.2
Hadoop	6.4	12.8	19.2	25.6	32	38.4

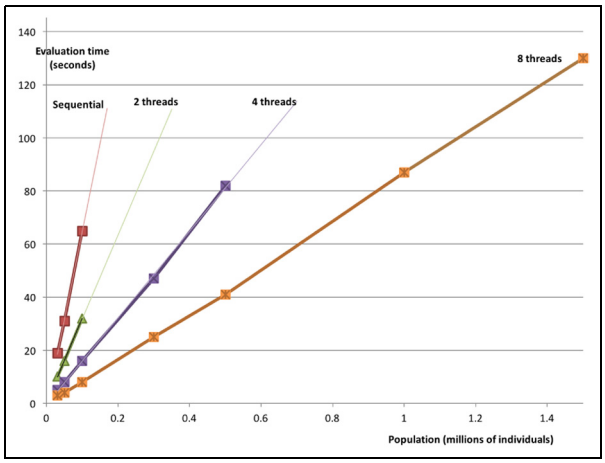

Figure 7. Evaluation times when using standard ECJ with multithreads.

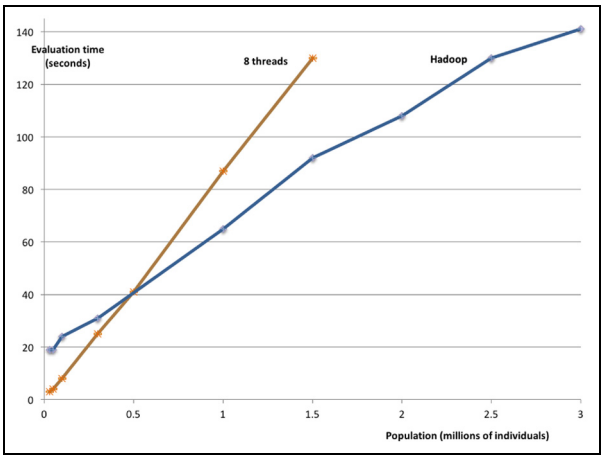

Figure 8. Comparison of best execution times between the local multi threading version vs. the Hadoop implementation.

Once results from parallel local processing have been obtained, times for the Hadoop based solution were calculated with different population sizes, as was the case for multithreading. Figure 8 shows the comparison between the best result from local executions, times with eight threads and the times obtained when using Hadoop. The blue line corresponds with the Hadoop based run using up to 40 available cores.

Machines had in total 40 cores; however, we considered six of them to be busy running the services that Hadoop needs to run in order to provide the required orchestration of the HDFS and the distributed resource manager (YARN). Nevertheless, every single task that performs the evaluation of individuals in Hadoop was restricted to one core and the configured total amount of cores was set to 34. This means that even if more cores were available, tasks would not be able to be run on more than 34 cores.

All previously described improvements were applied in the next round of experiments. Times for the Hadoop based run have being calculated as the average of different runs which were each configured with different block sizes. Block size refers to the size of the blocks in which files, where the individuals of the population are stored, are divided in the distributed file system. As a result, block size affects the total number of individuals that are stored per block. Since one task is launched for every block in the file system, the block size determines the number of individuals that each task will evaluate and, therefore, the time that the task will be running. Depending on the number of individuals and the time each task needs to be launched and cleaned, the block size should be configured accordingly. We did so during the experiments using different population sizes, here providing the average values. All runs employ the total number of cores available following the restrictions described before. Although many operations are performed by Hadoop when launching and running map/reduce tasks, we can clearly observe the difference obtained through standard ECJ multicore performances.

The most interesting part of Figure 8 is where the blue line overpasses the yellow one which corresponds with the ECJ with eight threads; this is the point at which ECJ + Hadoop starts to provide better performance. Therefore, we could conclude that for more standard population sizes, which are always below 500,000 individuals, it is not useful to employ ECJ + Hadoop, particularly in problems where the fitness evaluation time is short, as was the case for the Even Parity problem. We must take into account that for the Even Parity problem, the computing time is so short that only when huge population sizes are required will ECJ + Hadoop be of interest. On the other hand, if we are addressing a population with a longer fitness evaluation time, the critical point where ECJ + Hadoop is of interest will be reached at much smaller population sizes.

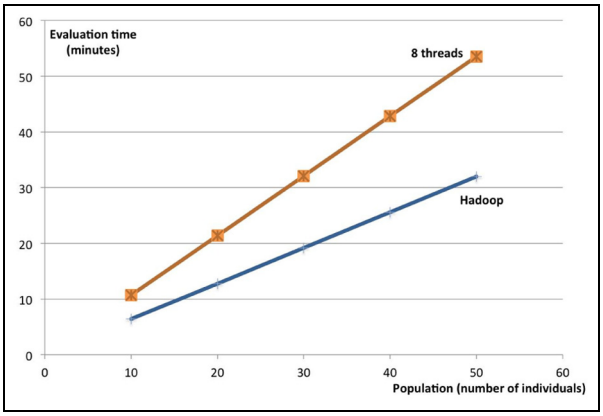

Figure 9. Comparing ECJ with eight threads vs. ECJ + Hadoop

Therefore, in our next experiment we tested the performance of the tool using the real-life problem at hand. We focus first on performance analysis of the parallel implementation; the quality of solutions obtained for the problem will be analyzed later on.

4.1.1 ECJ + Hadoop on a real life problem. In order to test the advantages of the new ECJ + Hadoop infrastructure, we launched a series of experiments trying to analyze the situation when the new implementation outperforms the results obtained by the sequential approach, as well as the results obtained with a multithread system. Table 5 shows results obtained with different population sizes. We must take into account that Hadoop incurs a computational overhead due to the extra processes and operations that are necessary to deploy and manage a distributed platform.

Table 5 shows running times obtained experimentally with up to 60 individuals using the standard sequential version of ECJ, the multithreading version running on a single PC with eight cores, and then the Hadoop based model with up to 40 cores. The single PC has the restriction of the number of cores, but the parallel infrastructure has no such restriction. We observe that for this specific problem the new version of the tool provides the best results with any of the size of the population tested. Given that ECJ + Hadoop can use as many computer nodes as required, we note that the running time could even improve by simply adding more nodes when required. On the other hand, the standard ECJ tool that can run on a single computer, cannot use more than the eight threads available in the multicore desktop PC in which it is running. Figure 9 allows us to graphically see how the differences increase as we manage larger populations.

As we have seen, ECJ + Hadoop is of interest when long computing times are required for assessing the quality of individuals in the population or when large populations are required to solve the problem.

4.2 Reducing the IP set for face recognition

The challenge of improving face recognition through the application of the CBIR paradigm is tackled here by a reduction in the number of IPs. As was shown in the reformulation of the fitness function (see equation (1)), a penalization was incorporated in all individuals that increase the number of IPs, forcing the GA to select solutions with a lower number of points.

In the first experimental stage of our test using the simple fitness where only the recovery percentage is considered of an individual, we decide to use populations of 100 individuals during 500 generations without achieving 100% in retrieval. In a second stage, the fitness was reformulated in such a way that the algorithm would stop if it found a solution that scores 100% using only 20 IPs. In this case, we use populations of 21 individuals during 100 generations with 70% crossover and 30% mutation for the process of creating the new population, and elitism to preserve the fittest individual. In general, we obtain results within 30 generations on average, as is shown in five examples of results plotted in Figure 10. This image shows on the left side the genome, with the percentage of classification, the number of IPs (NIP) and the fitness value; and on the right side the points used by the genome. We observe that through this technique a reduction from 60 to 20 IPs is achieved.

Note that besides the reduction in the number of points, the results show that the selected points achieving 100% retrieval are located within the central area of the face; in other words, the points outside this area are insignificant for the characterization of the face towards the construction of the associated pattern. This was also noted in previous work of the proposed CBIR methodology based on texture analysis (Benavides et al., 2015).

5 Conclusions

This work presents a new version of the popular ECJ tool, which provides an easy to use way of running EA over BigData infrastructure by making use of the MapReduce model. Moreover, that new version of the tool has been successfully applied to provide new approaches to a hard and time consuming real life problem related to face recognition; by means of an evolutionary approach, we have reduced the number of IPs required to solve the problem.

Firstly, we have developed a parallel version of ECJ that allows us to run experiments on a Hadoop based cluster of computers. The implementation makes use of the checkpointing facility already provided by ECJ. The system has been tuned and some hints for important parameters affecting running times for the experiments were provided. We have then performed a series of tests considering both traditional benchmark

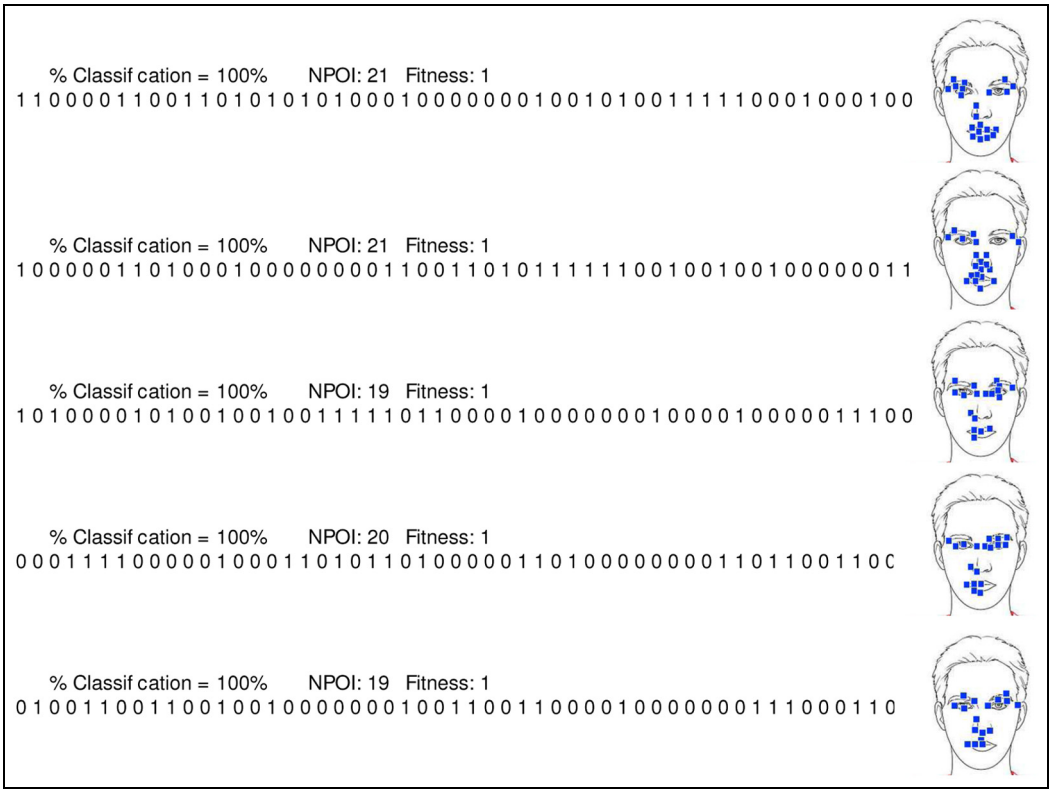

Figure 10. Example of genome construction based on IPs, each component is an IP index of the classifier. All results are achieved with an average of 20 IPs and 100% recognition.

problems already available within the ECJ tool and also the real-life problem of facial recognition. Results show that the new version of ECJ can save computing time when the fitness function is computationally intensive or when large population sizes are required to solve a given problem. Moreover, the implementation does not change the way ECJ is traditionally used: a new parameter has been added to allow researchers to select whether the run will be launched in a single computer or within the Hadoop based cluster.

Secondly, the tool has been employed to develop an evolutionary approach to the problem of selecting the best facial IPs required for a learning algorithm to be able to correctly perform face recognition. The approach has been able to reduce the number of required IPs from 60 to just 20, while a 100% recognition rate is still maintained over a number of standard data sets.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by FP7-PEOPLE-2013-IRSES, Grant 612689 ACoBSEC, Spanish Ministry of Economy, Project UEX:EPHEMEC (TIN2014-56494-C4-2-P); Junta de Extremadura, and FEDER, project GR15068. It was also supported by CONACyT México through (project number 155045, "Evolución de Cerebros Artificiales en Visión por Computadora") and TESE (project number DIMI-MCIM-004/08).

References

Benavides C, Villegas J, Román G, et al. (2015) *Face recognition using CBIR techniques* (Spanish). In: *Proceedings MAEB 2015*, Mérida, Spain, 4–6 February 2015, pp.733–740.

Cantu-Paz E (2000) *Efficient and accurate parallel genetic algorithms* (Vol. 1). Springer, USA. ISBN: 0-7923-7221-2.

Chávez F, Fernandez de, Vega F, Benavides C, et al. (2016) ECJ + HADOOP: An easy way to deploy massive runs of evolutionary algorithms. In: *Proceedings EvoApplications 2016*, LNCS 9598, Porto, Portugal, pp. 91–106. Berlin, Germany: Springer.

Cloudera (2015) Available at: http://www.cloudera.com.

Dean J and Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. *Communications of the ACM* 51(1): 107–113.

Dean J and Ghemawat S (2010) MapReduce: A flexible data processing tool. *Communications of the ACM* 53(1): 72–77.

Du X, Ni Y, Yao Z, et al. (2013) High Performance parallel evolutionary algorithm model based on MapReduce

framework. *International Journal of Computer Applications in Technology* 46(1): 290–295.

ECJ (2015) ECJ: A Java-based evolutionary computation research system. Available at: http://cs.gmu.edu/~eclab/projects/ecj/.

Fei-Fei L, Fergus R and Perona P (2007) Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding* 106(1): 59–70.

Fernández F, Sanchez JM, Tomassini M, et al. (1999) A parallel genetic programming tool based on PVM. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Berlin, Heidelberg, Germany: Springer, pp.241–248.

Fernández F, Tomassini M and Vanneschi L (2003) An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines* 4(1): 21–51.

Fernández F, Chávez F, Trujillo L, et al. (2014) A hybrid ECJ + Boinc tool for distributed evolutionary algorithms. *Research in Computing Science*. Vol. 1, pp. 120–130.

Gonzalez A and Prieto F (2010) Extracción de puntos característicos del rostro para medidas antropométricas. *Revista Ingenierías Universidad de Medellín* 9(17): 139–150.

García-Valdez M, Trujillo L, de Vega FF, et al. (2013) EvoSpace: A distributed evolutionary platform based on the tuple space model. In *European conference on the applications of evolutionary computation*, pp. 499–508. Berlin, Heidelberg: Springer.

González DL, de Vega FF, Trujillo L, et al. (2009) Increasing GP computing power for free via desktop grid computing and virtualization. In: *Proceedings of the 2009 17th Euromicro international conference on parallel, distributed and network-based processing*, 18–20 February 2009, pp.419–423. Piscataway, NJ: IEEE.

Hsu RL, Abdel-Mottaleb M and Jain AK (2002) Face detection in color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(5): 696–706.

Jain AK (2010) Automatic Face Recognition: State of the Art. *Distinguished Lecture Series*, 0-44, September 2010. LI, Stan Z. *Handbook of face recognition*. New York: Springer.

Koza JR (1992) *Genetic programming: On the programming of computers by means of natural selection*. Vol. 1. Cambridge, MA: MIT press.

Laredo JLJ, Eiben AE, van Steen M, et al. (2010) Evag: A scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines* 11(2): 227–246.

Lowe DG (2004) Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60(2): 91–110.

Melab N, Cahon S and Talbi EG (2006) Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing* 66(8): 1052–1061.

Olague G and Trujillo L (2012) Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image and Vision Computing* 29(7): 484–498.

O'Toole A (2013) Face recognition in humans and computers. In: Pashler H (ed.) *Encyclopedia of the Mind*. Vol. 2. Thousand Oaks, CA: SAGE Publications, pp.348–349.

Perez CB and Olague G (2013) Genetic programming as strategy for learning image descriptor operators. *Intelligent Data Analysis* 17(4): 561–583.

Serrano JF, Avilés C, Villegas J, et al. (2013) Self organizing natural scene image retrieval. *Expert Systems with Applications* 40(7): 2398–2409.

Sherry D, Veeramachaneni K, McDermott J, et al. (2012) Genetic programming on the cloud. In: *Applications of Evolutionary Computation*. Berlin, Heidelberg, Germany: Springer, pp.477–486.

Shvachko K, Hairong K, Radia S, et al. (2010) The Hadoop distributed file system, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, 1 -10, Hyatt Regency at Lake Tahoe Incline Villiage, NV, USA, 3–7 May, (2010). In: *Proceedings of the 2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, 3–7 May 2010, pp.1–10. Hyatt Regency at Lake Tahoe Incline Villiage, NV, USA.

Srinivasan A and Balamurugan V (2014) *A Novel Approach for Facial Feature Extraction in Face Recognition*. In *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol II*, pp. 155–162. Springer International Publishing.

Stelzer D and Mellis W (1998) Success factors of organizational change in software process improvement. *Software Process: Improvement and Practice* 4(4): 227–250.

Sun K, Kang H and Park H-H (2015) Tagging and classifying facial images in cloud environments based on KNN using MapReduce. *Optik—International Journal for Light and Electron Optics* 126(21): 3227–3233.

Tomassini M, Vanneschi L, Bucher L, et al. (2013) An MPI-based tool for distributed genetic programming. In: *Proceedings of the 2013 IEEE international conference on cluster computing (CLUSTER 2000)*, Indianapolis, IN, USA, 23–27 September 2013, p.209. Washington, DC: IEEE Computer Society.

Trujillo L and Olague G (2008) Automated design of image operators that detect interest points. *Evolutionary Computation* 16(4): 483–507.

Trujillo L, Olague G, Hammoud R, et al. (2005) Automatic feature localization in thermal images for facial expression recognition. In: *Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)—Workshops*, San Diego, CA, 20–25 June 2005, p.14.

Uwimana E and Ruiz ME (2008) Automatic classification of medical images for content based image retrieval systems (CBIR). *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 12(52): 788–792.

Wagner A, Wright J, Ganesh A, et al. (2012) Toward a practical face recognition system: Robust alignment and illumination by sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(2): 372–386.

White T (2012) *Hadoop: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, Inc.

Xiangxin Z and Ramanan D (2012) Face detection, pose estimation, and landmark localization in the wild. In: *Proceedings of the 2012 IEEE conference on computer vision and pattern recognition (CVPR)*, Providence, RI, USA, 16–21 June 2012, pp.2879–2886. doi:10.1109/CVPR.2012.6248014

Yang MH, Kriegman DJ and Ahuja N (2002) Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(1): 34–58.

Zhang L, Yang M and Feng X (2011) Sparse representation or collaborative representation: Which helps face recognition? In: *Proceedings of the 2011 IEEE international conference on computer vision (ICCV)*, Barcelona, Spain, 6–13 November 2011, pp.471–478. doi:10.1109/ICCV.2011.6126277

Zhang Z, Li W and Jia H (2014) A fast face recognition algorithm based on MapReduce. In: *Proceedings of the 2014 seventh international symposium on computational intelligence and design (ISCID)*, 13–14 December 2014, Vol. 2, pp.395–399. doi:10.1109/ISCID.2014.195

Zhao W, Chellappa R, Phillips PJ, et al. (2003) Face recognition: A literature survey. *ACM Computing Surveys (CSUR)* 35(4): 399–458.

Author Biographies

Francisco Chávez received his PhD in Computer Science from the University of Extremadura, Spain, 2012. Since 2001, he has been with the Department of Computer Science Systems and Telematics Systems at the University of Extremadura, Spain. He is currently an associate professor there, where he is a member of the Artificial Evolution Research Group. He has over 50 national and international publications. He has worked as principal researcher on several research projects supported by the Spanish Government, the Extremadura Regional Government and projects with Spanish industries. His research interests include fuzzy rules based systems, genetic fuzzy systems, image processing and parallel algorithms.

Francisco Fernádez de Vega received his PhD in Computer Science from the University of Extremadura, Spain, 2001. He was vice-head of research at the Centro Universitario de Mérida, University of Extremadura, Spain, from 2004 to 2005, and CIO of the University of Extremadura from 2005 to 2007. He is currently an associate professor of computer science and the director of the GEA Research Group (Artificial Evolution Group). He has published over 200 referred papers. His research interests include bioinspired algorithms, fuzzy logic and cluster and grid computing. He is part of the steering committee of the Spanish Conference on Evolutionary Algorithms (MAEB), and has presented invited tutorials at several international conferences (including IEEE CEC). He was co-chair of the first and second workshops on parallel bioinspired algorithms, held jointly with IEEE ICPP in 2005, and ACM GECCO in 2007, and the first, second and third workshop on parallel architectures and bioinspired algorithms, held by IEEE PACT (Toronto 2008, Raleigh 2009, Vienna 2010). He has edited a several special issues dealing with parallel bioinspired algorithms (*Journal of Parallel and Distributed Computing, Journal of Soft Computing* and *Journal of Parallel Computing*).

Daniel Lanza received the MSc degree in computer science in 2014 from the University of Extremadura, Spain, and is currently a PhD degree student in computer science. He is currently a big data engineer at CERN, the European Organization for Nuclear Research.

César Benavides received his engineering degree in electronics in 2012 and the MSc in 2015 from the Universidad Autónoma Metropolitana (UAM), México. He is currently a PhD student in the science and information technologies postgraduate program at UAM. His research interests are computer vision, digital image processing, pattern recognition, evolutionary computing and distributed and parallel computing.

Juan Villegas received his BS degree in applied mathematics from the Universidad Autónoma Metropolitana (UAM), México, in 1996. He then obtained his master's degree in computer science from UAM in 2005 and his PhD from computer science program of the Pattern Recognition Laboratory at the Center for Computer Research of the National Polytechnic Institute of Mexico in 2009. Since 2010, he has held a position as full professor at the UAM. He has been distinguished as a level 1 national researcher by the National System of Investigation (SNI) of Mexico, and as faculty advisor for NASA during its Third Annual Lunabotics Mining Competition, in 2012. His research interests are pattern recognition, image analysis, artificial neural networks, associative memories and evolutionary algorithms. He has published more than 10 international papers and has directed two master's degree theses.

Leonardo Trujillo is a research professor at the Technical Institute of Tijuana in Mexico (ITT), involved in interdisciplinary research within the fields of evolutionary computation, computer vision, data analytics, pattern recognition and autonomous robotics. Dr. Trujillo received a master's in computer science from ITT in 2004, and a doctorate in computer science from CICESE (Mexico) in 2008. Dr. Trujillo's work is primarily focused on genetic programming (GP) and the application of GP to pattern recognition problems. He has published over 30 papers in top journals in these fields, and over 50 papers in conference proceedings, receiving several best paper awards from conferences such as GECCO and Evo Star. He is currently leading several national and international research projects, as well as serving as the head of cybernetics research and the graduate program of engineering sciences at ITT.

Gustavo Olague received his PhD in computer vision, graphics and robotics from INPG (Institut Polytechnique de Grenoble) and INRIA (Institut National de Recherche en Informatique et Automatique) in France. He is a Professor in the department of computer science at CICESE (Centro de Investigación Científica y de Educación Superior de Ensenada), México, and the Director of its EvoVisión Research Team. He is also an Adjoint Professor of Engineering at UACH (Universidad Autonóma de Chihuahua). He has authored over 100 conference proceedings papers and journal articles. He coedited two special issues in *Pattern Recognition Letters* and *Evolutionary Computation*, and he served as co-chair of the real-world applications track at the main international evolutionary computing conference, GECCO (ACM SIGEVO Genetic and Evolutionary Computation Conference). Professor Olague has received numerous distinctions, among them the Talbert Abrams Award presented by the American Society for Photogrammetry and Remote Sensing (ASPRS) for authorship and recording of current and historical engineering and scientific developments in photogrammetry; best paper awards at major conferences such as GECCO, EvoIASP (European Workshop on Evolutionary Computation in Image Analysis, Signal Processing and Pattern Recognition) and EvoHOT (European Workshop on Evolutionary Hardware Optimization); and two bronze medals at the Humies (GECCO award for human-competitive results produced by genetic and evolutionary computation). His main research interests are evolutionary computing and computer vision. He is author of the book *Evolutionary Computer Vision* published by Springer in 2016.

Graciela Román-Alonso is a full professor at the Universidad Autónoma Metropolitana, México. In 1997, she received her PhD from the Université de Technologie de Compiègne, France. She was an invited professor at the PARADISE Research Laboratory at the University of Ottawa, Canada, 2010–2012. Her research interests include dynamic load distribution, parallel and distributed computing systems, scientific computing, 3D streaming, P2P networks, and vehicular ad hoc networks.