Performance Evaluation of the Parallel Polytree Approximation Distribution Algorithm on Three Network Technologies*

Julio Madera
Department of Computing, University of Camagüey
Camagüey, Cuba
jmadera@inf.reduc.edu.cu

and

Enrique Alba, Gabriel Luque
Departamento de Lenguajes y Ciencias de la Computación
E.T.S.I. Infomática, University of Málaga, Spain
{eat, gabriel}@lcc.uma.es

Abstract

This paper proposes two parallel variants of an Estimation of Distribution Algorithm (EDA) that represents the probability distribution by means of a single connected graphical model based on a polytree structure. The main goal is to design a new and more efficient EDA. Our algorithm is based on the master/slave model that allows to perform the estimation of the probability distribution (the most time-consuming phase in EDAs) in a parallel way. The aim of our experimental studies is manifold. Firstly, we show that our parallel versions achieve a notable reduction of the total execution time with respect to existing algorithms. Secondly, we study the behavior of the algorithm from the numerical point of view, analyzing the different versions. Finally, our methods are evaluated over three interconnection networks (Fast Ethernet, Gigabit Ethernet, and Myrinet) and a study on the influence of the parallel platform in the communication is performed.

Keywords: Parallel Estimation of Distribution Algorithms, Bayesian Networks, Polytree Approximation Distribution Algorithm

1 INTRODUCTION

Evolutionary Algorithms (EAs) are non deterministic search techniques designed as an attempt to solve adaptive and hard optimization tasks on computers [2]. In fact, it is possible to find this kind of algorithms applied for solving complex problems in economy, telecommunications, bioinformatics, etc. The landscape of these problems often shows multiple optima, noisy regions, and a large dimensionality. These algorithms work over a set (*population*) of potential solutions (*individuals*) by applying some stochastic operators on them, called *variation operators* (e.g., natural selection, recombination, or mutation), in order to search for the best solutions.

In the last years a new family of EAs known as Estimation of Distribution Algorithms (EDAs) [3, 8] has deserved a large attention in the scientific community related to optimization, evolutionary computation, and probabilistic models. These algorithms have arisen as an alternative to other

*The two last authors acknowledge founds from the Spanish Ministry of Education and European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project, http://oplink.lcc.uma.es).

methods where it is necessary to fit a high number of parameters. EDAs have been motivated by the need to identify the interrelations between the variables, a key issue to solve complex problems. In EDAs, a different kind of variation operators is used. The successive generations of individuals are created by using estimations of the probability distributions observed in the current population, instead of evolving the population with the typical variation operators (like crossover and mutation) used in other EAs. Hence, the main feature distinguishing EDAs from other more classical EAs is that EDAs learn the interactions among variables (building blocks) in the problem. At the same time, this is the principal flaw of EDAs due to the computational complexity of this learning and simulation task. EDAs are classified according to the used learning model, and range from those that assume total independency of the variables to those that assume pairwise interactions; some EDA families also consider unrestricted general models of interaction between variables. In Algorithm 1, we show the pseudocode for an EDA.

Algorithm 1 EDA

```
Set $t \leftarrow 1$;
Generate $N >> 0$ points randomly;
while termination criteria are not met do
    Select $M \leq N$ points according to a selection method;
    Estimate the distribution $p^s(x,t)$ of the selected set;
    Generate $N$ new points according to the distribution $p^s(x,t)$;
    Set $t \leftarrow t+1$;
end while
```

Apart from the computation time of the objective function, the main computational cost of these algorithms is the time consumed in learning the probabilistic graphical model at each step. One obvious approach to deal with this problem is the parallelization and/or distribution of the algorithms.

The current focus of the research in the development of parallel EDAs has been concentrated on two main approaches: evolution of parallel populations and distribution of procedures acting on sequentially evolving populations. Although this field is quite new, parallel EDAs have deserved interest in the recent past, e.g., a good review can be found in [5]. We only concentrate in works that make use of the second approach.

Lozano et. al. [4] proposed two parallel versions for an algorithm that uses Probabilistic Graphical Models in combinatorial optimization ($EBNA_{BIC}$). Extending the preceding work, Mendiburu et. al. [6] proposed several parallel implementations of EDA algorithms, not only for discrete domains but also for continuous ones. To learn the Bayesian network a manager/worker model is employed. In [7] is proposed an extension of $pEBNA_{BIC}$ that was presented in the previous work.

Ocenasek and Schwarz [9, 10] proposed two methods of parallelization for the Bayesian Optimization Algorithm (BOA) [11]. The main idea of the $pBOA$ [9] is to parallelize the learning of the Bayesian network step. To do that, each processor introduces arcs in the tentative network irrespective of other processors. In this algorithm, the authors used explicit topological ordering of the variables to keep the model acyclic.

In this work, our effort focuses on the parallelization of this learning process. We propose two parallel solutions for a special EDA that represents the joint probability distribution by means of a single connected Bayesian network. The algorithm is named Polytree Approximation Distribution Algorithm ($PADA$) [12]. Our approach takes advantage of the detection of independencies to construct the Bayesian network (see Eq. 2 and 3). Previos models in the literature [4,9,10] do not include this feature. We also extend the works of Ocenasek and Schwarz [9, 10], performing the learning phase without taking into account restrictions in the order of the variables. Parallel $EBNA_{PC}$ algorithm [6,7] have an approach similar to our proposed method but it starts from a completely undirected graph, and it also uses the χ^2 to perform the independence tests while our algorithm creates the network from scratch and it uses a threshold (ε) in the independence tests. Another important difference is that our proposal considers the balance of the computations carried out by the processors. Finally, we also test our parallel methods with different communication networks to give relative advantages of modern networks.

The outline of the paper is as follows. In the next section, we present the sequential version of the $PADA$ algorithm. In Section 3 we discuss the details of two new parallel algorithms. Section 4 shows the computational experiments and the evaluation of the results using the two parallel versions on the selected problem benchmark. Finally, conclusions and some future directions are outlined.

2 POLYTREE APPROXIMATION DISTRIBUTION ALGORITHM

In this section we present the sequential $PADA$ algorithm and its principal features. As an EDA, $PADA$ executes the main loop of a canonical EDA. The main difference with respect to other EDAs lies in how it estimates the probability distribution of the selected set of tentative solutions. In this technique, the probability model is a single connected Bayesian network known as polytree.

Formally, we can define a Bayesian network over the set of random variables $X = \{X_1, X_2, \ldots, X_n\}$. The factorization of the joint probability distribution can be expressed as:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi_{X_i}) \tag{1}$$

where, π_{X_i} is the set of parents of X_i (i.e., exists an arc from each π_{X_i} to X_i).

In this paper we concentrate in the simple connected Bayesian networks, specially in polytree structures. Basically, a polytree is a type of graph in which there is at most one undirected path between any two vertices. In other words, a polytree is a DAG for which there are no undirected cycles. Networks with tree structure capture the main features of causality among the variables (dependency among their values) and provide a good computational environment for optimization purposes. In the other hand, polytrees allow to describe high order interactions and exhibit many of the computational advantages of simple trees.

The dependence relations are extracted from databases and this process is known as Bayesian network structure learning. There are two different ways to classify structure learning algorithms. One kind of algorithms for structure learning uses a score+search procedure. These methods define a metric (cost function) that measures the suitability of every candidate Bayesian network to a database of cases. Other approaches perform (in)dependence tests to obtain a list of (in)dependence weighted assertions. These algorithms construct the graph structure satisfying as much as possible the assertions on the list. The algorithms presented in this paper belong to this last class.

As we stated at the beginning of this section, our algorithm uses a Bayesian network to represent the probability distribution. We propose the parallelization of the learning phase in the $PADA$ algorithm. We use a structure learning that applies (in)dependence tests. For our work, the following standard definitions are needed:

The mutual information $I(X_i, X_j) = Dep(X_i, X_j)$ of the random variables X_i and X_j is:

$$I(X_i, X_j) = \sum_{x_i, x_j} p(x_i, x_j) \cdot \log \frac{p(x_i, x_j)}{p(x_i) \cdot (x_j)} \tag{2}$$

The conditional mutual information $I(X_i, X_j|X_k) = Dep(X_i, X_j|X_k)$ of the random variables X_i and X_j given X_k is defined as:

$$I(X_i, X_j|X_k) = \sum_{x_i,x_j,x_k} p(x_i, x_j, x_k) \cdot \log \frac{p(x_i, x_j, x_k)}{p(x_i, x_k) \cdot (x_j, x_k)} \tag{3}$$

The measure of global dependency $Dep_g(X_i, X_j)$ of the random variables X_i and X_j given X_k is defined as:

$$Dep_g(X_i, X_j) = \min_{\{X_k\}}(Dep(X_i, X_j), I(X_i, X_j|X_k)) \tag{4}$$

2.1 Learning Polytree Approximation Algorithm

This algorithm takes as a starting point the Polytree Approximation Algorithm (PA) developed by Acid and de Campos [1]. The basic idea of the PA consists of preserving the edges with larger weights. The weight of an edge is given by the global dependency $Dep_g(X_i, X_j)$ (see Eq. 4), that represents the lowest value between the marginal dependency value $Dep(X_i, X_j) = I(X_i, X_j)$ and the value of mutual conditional information for each of the others variables $(I(X_i, X_j|X_k))$. The calculation of the marginal dependency is the most complex step $(O(n^3))$ of this algorithm.

The method employed to learn the polytree is a modified version of the PA algorithm. It is shown in the Algorithm 2 and it is called LPA. This algorithm uses two different thresholds for independency: ε_o and ε_1. Also, it performs changes with respect to the orientation of the edges.

This algorithm produces the fittest polytree to the data during the learning phase. Our goal is to parallelize the LPA method. Now let us detail the main steps of this technique.

The LPA algorithm starts with an empty graph (G) and an empty list (L) to store the dependence relations among the variables detected in the data (line 1). For each pair of variables $\langle X_i, X_j \rangle$ the algorithm computes the marginal dependency (Eq. 2) and arcs with $Dep(X_i, X_j) > \varepsilon_0$ are stored in the list L (lines 2-7). With the resulting arcs in L, LPA computes the mutual conditional information (Eq. 3), removing the arcs with $Dep(X_i, X_j|X_k) < \varepsilon_1$ (lines 8-16). For all the edges $\langle X_i, X_j \rangle$ in L, the algorithm calculates the value of $Dep_g(X_i, X_j)$ (Eq. 4), using this value to sort the list L (lines 17-20). Now we construct the skeleton of the polytree (lines 21-26). As it can be seen, the edges are inserted in the graph traversing the list L, taking into account the restriction that the insertion of the arc $< X_i, X_j >$ does not create an undirected cycle in G. With the resulting skeleton, the algorithm proceeds to set up the direction of the edges in G (lines 28-31). To do this, for each connection $X_i - X_k - X_j$ if $I(X_i, X_j|X_k) > I(X_i, X_j)$ then it creates a $head - to - head$ pattern. The remaining edges are oriented applying a cost function based on the BIC metric or at random, without introducing new $head - to - head$ patterns.

3 PARALLEL PROPOSALS

A detailed study of the Algorithm 2 concludes that computing the marginal dependency and the mutual conditional information are the most costly steps: $O(n^2)$ and $O(n^3)$, respectively. This computation is easily separable and can be executed in parallel. Our approaches exploit these facts: complexity and separability. To do these parallel computations, the algorithms take advantage of the well-known master/slave model.

Algorithm 2 LPA algorithm

```
1: Start with an empty graph $G$ and an empty list $L$
2: for all $X_i, X_j \in X$ do
3:     Compute $Dep(X_i, X_j)$
4:     if $Dep(X_i, X_j) \geq \varepsilon_0$ then
5:         Insert the edge $< X_i, X_j >$ in $L$
6:     end if
7: end for
8: for all $< X_i, X_j >$ in $L$ do
9:     for all $X_k \in X, X_k \neq X_i, X_j$ do
10:        Compute $I(X_i, X_j|X_k)$
11:        if $I(X_i, X_j|X_k) < \varepsilon_1$ then
12:            Delete the edge $< X_i, X_j >$ from $L$
13:            Select the next edge $< X_i, X_j >$ in $L$
14:        end if
15:    end for
16: end for
17: for all $< X_i, X_j >$ in $L$ do
18:    Compute $Dep_g(X_i, X_j)$
19: end for
20: Sort $L$ in decreasing order by $Dep_g(X_i, X_j)$
21: repeat
22:    Select the next edge $< X_i, X_j >$ in $L$
23:    if $< X_i, X_j >$ does not create a cycle in G then
24:        Add $< X_i, X_j >$ to $G$
25:    end if
26: until $n - 1$ edges have been added
27: for all $X_i - X_k - X_j \in G$ do
28:    if $I(X_i, X_j|X_k) > Dep(X_i, X_j)$ then
29:        Make head-to-head pattern $X_i \rightarrow X_k \leftarrow X_j$
30:    end if
31: end for
32: Direct the remaining edges applying some cost function
33: Compute $p(x) = \prod_{i=1}^{n} p(x_{j1(i)}, \ldots, x_{jr(i)})$
```

3.1 The First Proposal: $pPADA_{BAL}$

This algorithm is based on the separability of the marginal dependency and the mutual conditional information computation. When we have n variables, the total number of marginal dependence tests performed is $\frac{n\cdot(n-1)}{2}$. Therefore, this computation can be divided among the master and the slaves processors. To do that, each processor maintains a copy of the data set (selected individuals). After this, the next step in the sequential version is the computation of the mutual conditional information. This step performs $\frac{n\cdot(n-1)\cdot(n-2)}{6}$ tests. As we did before, we can divide the computation among all the processors. When each processor computes the marginal dependencies the number of edges obtained in each processor can be different. This can lead to assigning to each processor a different workload, and therefore, its execution time can vary among the processors. To avoid this uneven workload, our first proposed parallel version implements a mechanism to redistribute the work. Firstly, the calculation of the marginal dependencies among the variables is performed in parallel, then the results of this step are sent to the master, and finally this process redistributes the computation of the mutual conditional information among all the processors.

The scheme of this technique is shown in the left diagram of the Figure 1. It must be clear that master also plays the slave role. We can distinguish four phases in this algorithm: an initialization step, the calculation of dependency information, the construction of the graph, and finally, the orientation of the edges in the resulting graph. During the initialization, the master process sends some information to the slaves. In the second phase, the master and the slaves compute the marginal and the conditional mutual information in a parallel and asynchronous way. Also, in this phase, the master redistributes the workload among all the processes in an intermediate step between these two calcu-

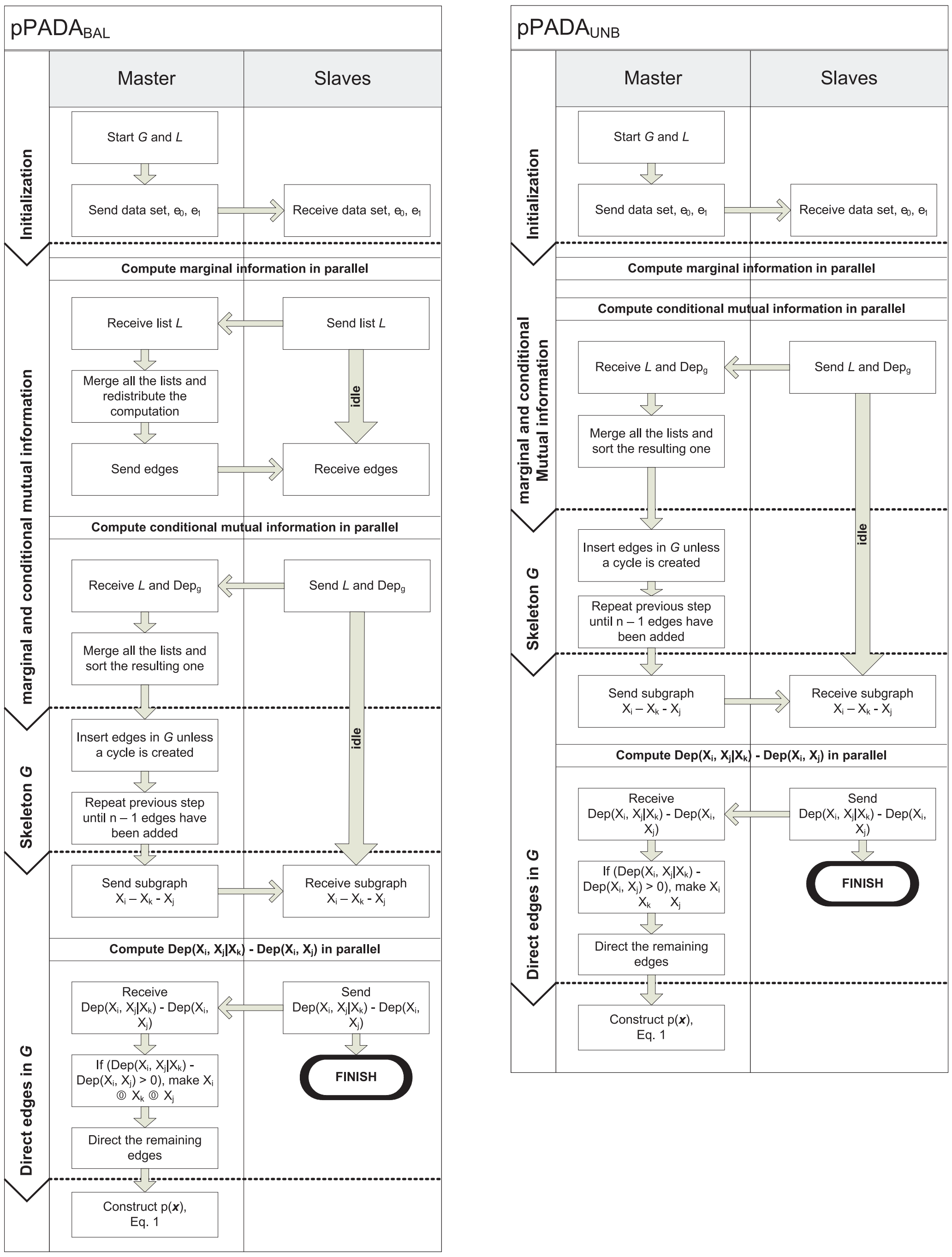
pPADA_BAL
Master
Slaves
Initialization
Start G and L
Send data set, e0, e1
Receive data set, e0, e1
marginal and conditional mutual information
Compute marginal information in parallel
Receive list L
Send list L
Merge all the lists and redistribute the computation
idle
Send edges
Receive edges
Compute conditional mutual information in parallel
Receive L and Dep_g
Send L and Dep_g
Merge all the lists and sort the resulting one
Skeleton G
Insert edges in G unless a cycle is created
Repeat previous step until n – 1 edges have been added
Send subgraph Xi – Xk - Xj
Receive subgraph Xi – Xk - Xj
Direct edges in G
Compute Dep(Xi, Xj|Xk) - Dep(Xi, Xj) in parallel
Receive Dep(Xi, Xj|Xk) - Dep(Xi, Xj)
Send Dep(Xi, Xj|Xk) - Dep(Xi, Xj)
FINISH
If (Dep(Xi, Xj|Xk) - Dep(Xi, Xj) > 0), make Xi → Xk ← Xj
Direct the remaining edges
Construct p(x), Eq. 1
pPADA_UNB
Master
Slaves
Initialization
Start G and L
Send data set, e0, e1
Receive data set, e0, e1
marginal and conditional Mutual information
Compute marginal information in parallel
Compute conditional mutual information in parallel
Receive L and Dep_g
Send L and Dep_g
Merge all the lists and sort the resulting one
idle
Skeleton G
Insert edges in G unless a cycle is created
Repeat previous step until n – 1 edges have been added
Send subgraph Xi – Xk - Xj
Receive subgraph Xi – Xk - Xj
Direct edges in G
Compute Dep(Xi, Xj|Xk) - Dep(Xi, Xj) in parallel
Receive Dep(Xi, Xj|Xk) - Dep(Xi, Xj)
Send Dep(Xi, Xj|Xk) - Dep(Xi, Xj)
FINISH
If (Dep(Xi, Xj|Xk) - Dep(Xi, Xj) > 0), make Xi Xk Xj
Direct the remaining edges
Construct p(x), Eq. 1

Figure 1: Functioning scheme for $pPADA_{BAL}$ and $pPADA_{UNB}$ algorithms.

lations. Once the master has received all the information from the slaves, it arranges the edges in decreasing order by the global dependency and constructs the polytree skeleton, inserting the arcs one by one, according to the established order, and avoiding the creation of cycles (third phase). In the last phase, the master sends the subgraphs to the slaves to perform a parallel computation of the difference between the mutual conditional dependency and the mutual marginal dependency. Finally, using this information, the master process proceeds to orient the edges and it calculates the joint probability distribution according to Eq. 1.

3.2 The Second Proposal: $pPADA_{UNB}$

This second variant is similar to the previous one but it does not implement any balance of the workload among the processors after the calculation of the marginal dependency (see right diagram of Figure 1). This obeys to the fact that there could exist some situations where the number of edges in each sublists does not differ substantially from one process to another and therefore a new balance of the load would unnecessarily increase the Communication time.

4 EXPERIMENTAL RESULTS

This section presents the experiments carried out to test our parallel algorithms. Our objective is to compare the performance and the communication load in three different network technologies: Fast Ethernet, Gigabit Ethernet, and Myrinet. Since the master/slave model does not change the dynamics of the canonical method, we want also to prove that there is no a significant numerical difference among the different versions of the method, although the parallel version is considerably faster.

Our parallel platform is composed of a cluster of eight nodes, where each node has a Intel P-IV processor (CPU 2.8GHz), 512KB of cache memory and 512MB of main memory. The operating system is RedHat Linux, version 7.2. The MPI implementation is LAM (version 7.1.1). All the machines are interconnected using three kinds of networks: Fast Ethernet (10^2 Mbps), Gigabit Ethernet (10^3 Mbps), and Myrinet ($2 \cdot 10^3$ Mbps). Next subsection details the three optimization problems that we use in our experimental benchmark.

4.1 Problems

Since we focus on the learning phase and the influence of the network technology in the algorithm, we have selected three standard problems in the EDA field. Our study includes the well-known $OneMax$, $Plateau$ and $Muehlenbein$ functions. In Table 1 we present, for each of these problems, its name, its fitness function (to be maximized), the size of the studied instance, the chromosome representing the optimal solution to the problem, and the value of this optimum. In all the cases, n represents the dimension of the problem, and a binary genotype is used ($x_i \in \{0, 1\}$).

The Onemax problem simply consists in maximizing the number of ones in a bit-string. In the Plateau function, the chromosome is divided into groups of three genes, and the fitness value is the number of sets containing three ones. Finally, in $Muehlenbein$ function the chromosome is divided into groups of five genes, and the subfunction fitness value oscillate between zero and four, it is a deceptive problem.

Table 1: Benchmark problems.

Problem	Fitness function	Size (n)	Solution chromosome	Optimum
Onemax	$f_{OneMax}(\vec{x}) = \sum_{i=1}^{n} x_i$	300	(1,1,1,1,...,1,1)	300
Plateau	$f_{Plateau}(\vec{x}) = \sum_{i=1}^{m} g(\vec{s_i})$ where $\vec{s_i} = (x_{3i-2}, x_{3i-1}, x_{3i})$, and $m = \frac{n}{3}$ $g(x_1, x_2, x_3) = \begin{cases} 1, & \text{if } x_1 = x_2 = x_3 = 1 \\ 0, & \text{otherwise} \end{cases}$	300	(1,1,1,1,...,1,1)	100
Muehlenbein	$f_{Muehl}(\vec{x}) = \sum_{i=1}^{m} f^5_{Muehl}(\vec{s_i})$ where $\vec{s_i} = (x_{5i-4}, x_{5i-3}, x_{5i-2}, x_{5i-1}, x_{5i})$, and $m = \frac{n}{5}$ $f^5_{Muehl}(x_1, x_2, x_3, x_4, x_5) = \begin{cases} 3.0, & \text{for } x = (0,0,0,0,1) \\ 2.0, & \text{for } x = (0,0,0,1,1) \\ 1.0, & \text{for } x = (0,0,1,1,1) \\ 3.5, & \text{for } x = (1,1,1,1,1) \\ 4.0, & \text{for } x = (0,0,0,0,0) \\ 0, & \text{otherwise} \end{cases}$	300	(0,0,0,0,...,0,0)	240

4.2 Computational Results

In this section we analyze the behavior of our two parallel methods when they are executed over the three communication networks. First, we describe the parameter setting of the experiments, and later we analyze the results.

Several preliminary experiments were performed to analyze how the parameters affect to the performance of the algorithms. From these previous analyses, we conclude that the best configuration for our problem instances is the parameter setting showed in Table 2.

For each combination of the algorithm, network, and problem we perform 100 independent runs to gather statistical meaningful experimental data. To allow a fair comparison among the results of these algorithms, we have configured them to perform a similar computational effort (a predefined maximum number of generations). In concrete, the algorithm execution is stopped when the 20th generation is reached. This termination condition also allows to evaluate the parallel model and their execution time and can also be found in other relevant works like [6].

Table 2: Parameter setting.

Parameter	F_{OneMax}	$F_{Plateau}$	F_{Muehl}
Population size	450	450	1400
Truncation threshold	0.3	0.3	0.3
Elitism	1	1	1
ε_0	0.0029	0.0029	0.0025
ε_1	0.0019	0.0019	0.0015

To analyze the performance of the parallel technique we have used two metrics. The most important metric to study parallel techniques is the speedup. This measure compares two times: the ratio between the sequential time and the parallel one. Efficiency is also usually reported, i.e., the speedup divided into the number of processors (normalization).

Due to space limitation we do not show tables with the total execution time, but we briefly summarize them in this paragraph. For each function, we show the sequential time and the parallel time for eight processors in the following order: balanced Fast Eth., balanced Gigabit Eth., balanced Myrinet, unbalanced Fast Eth., unbalanced Gigabit Eth. and, unbalanced Myrinet. For the F_{OneMax} function the sequential time is approximately 384.27 seconds and for eight processors 57.13, 53.66, 53.46, 56.77, 54.42 and, 54.48 seconds. For the $F_{Plateau}$ function the sequential time is approximately 458.48 seconds and for eight processors 61.35, 57.76, 57.75, 61.14, 58.51 and, 58.35 seconds. Finally, for the F_{Muehl} function the sequential time is approximately 437.16 seconds and for eight processors 84.84, 80.26, 80.51, 85.09, 82.55 and, 82.56 seconds.

Figure 2 shows the speedup (first column) and the efficiency (second column) for all the networks. Several conclusions can be extracted of these figures. First, Gigabit Ethernet and Myrinet networks obtain a very similar value, and the Fast Ethernet gets always a slightly worse speed-up and efficiency. This means that the Fast Ethernet loses more time in the exchange of data packets than the other networks. This is evident especially when the number of processors is increased, since a higher number of processes provokes a more intense utilization of the network, causing more conflicts and producing a moderate loss of efficiency.

Second, the balanced version of the algorithm has better results than the unbalanced approach. This means that the number of edges resulting of the marginal dependency computation is slightly different in every processor and the mechanism to redistribute the workload is beneficial.

Finally, other important conclusion is that for F_{OneMax} and $F_{Plateau}$ problems $pPADA_{BAL}$ is very efficient. In concrete, it achieves an efficiency value of 0.85 with eight processors, and super-linear speedup (the efficiency is higher than 1.0) for any configuration with less than four processors. This superlinear value can be due to the fact that the necessary data (selected population) can be completely

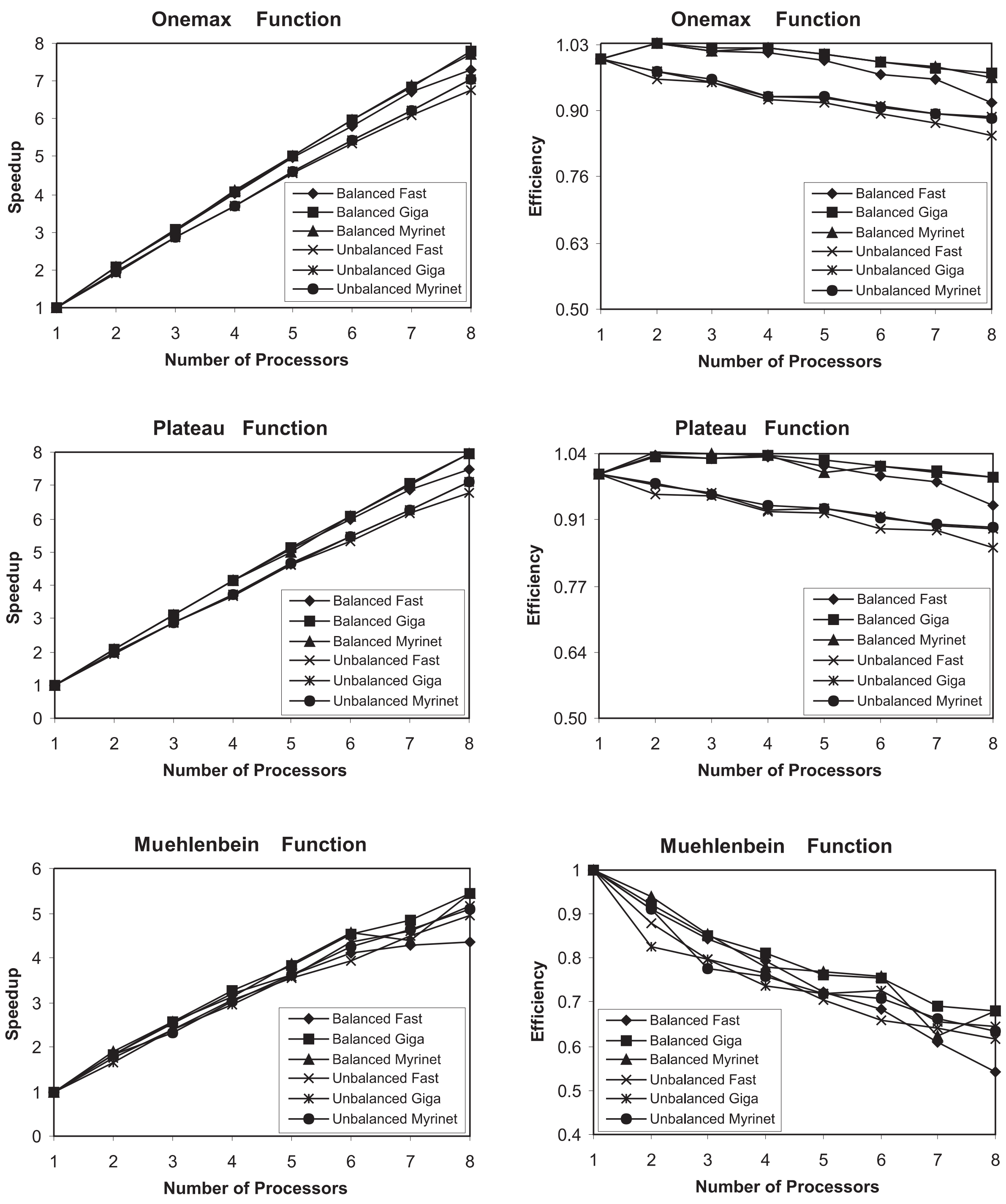

Figure 2: Speedup and efficiency for the three problems (average over 100 runs).

stored in the cache of the processors (512 KB), while the sequential version needs to use the (slower) main memory to store the population and the auxiliary data structures.

The F_{Muehl} problem is quite more complex than the rest. It needs a larger population to obtain accurate results. The increase of the population size provokes a higher communication overhead and consequently the speedup and efficiency are worse than in the other problems. In addition, this problem converges slowly, and in fact, our methods do not find the optimal solution since the search is limited to only 20 generations. The PADA method initially generates polytrees with a high number of edges and iteratively decreases this number. Since our algorithm only executes 20 generations, all the generated polytrees in this problem have a considerably large number of edges, and therefore the exchange of edges among the processes increases the utilization of the network. In spite of these drawbacks, parallel versions allow an important reduction of the execution time with respect to the sequential algorithm (from 39.38% to 81.64%).

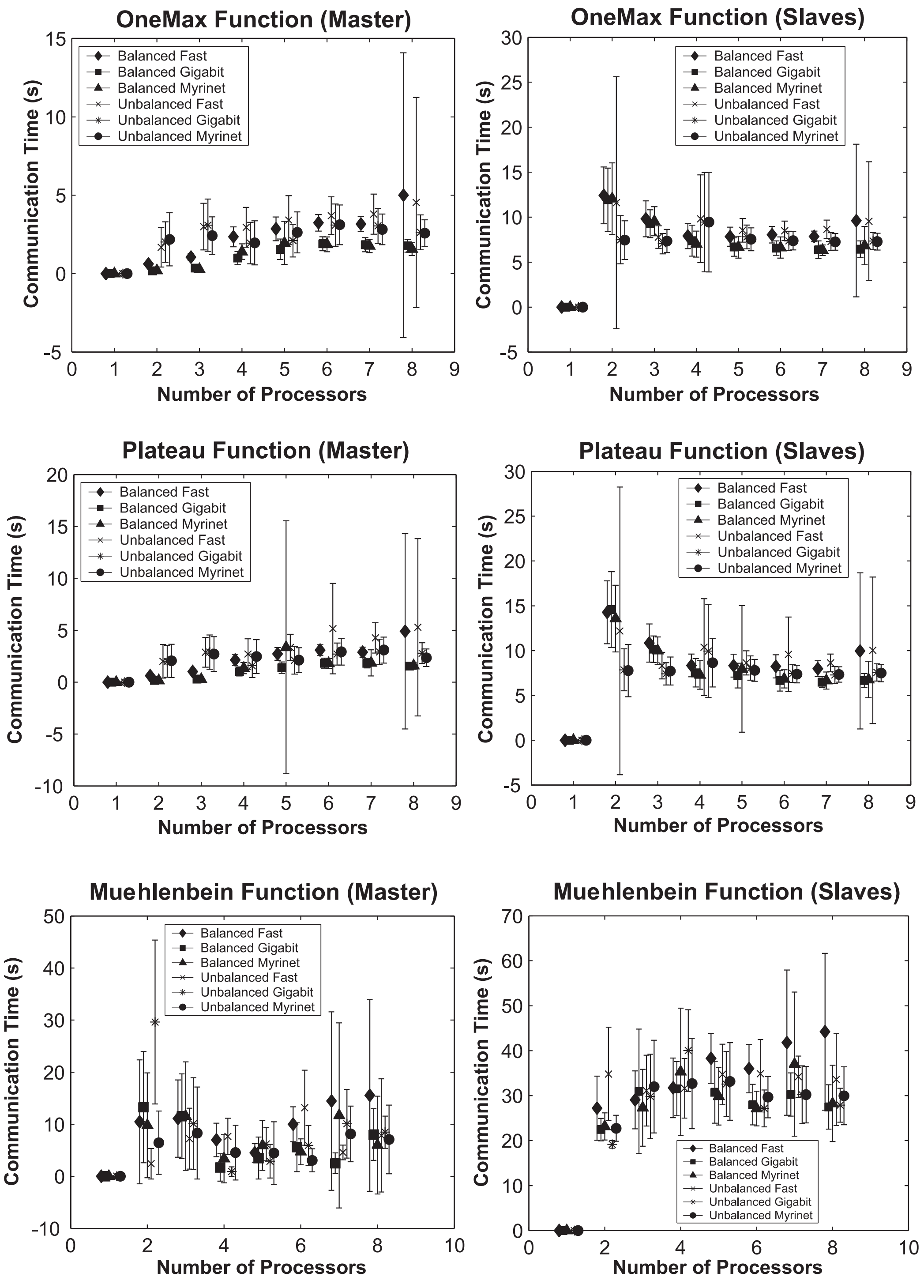

Figure 3: Communication overhead for the three problems.

Now we turn on the communication cost analysis. These values are shown in Figure 3. This figure shows the time spent in communication and synchronization (Comm. & Sync.) for master (left column) and the slaves (right column). For all the tested problems the time spent in communication is small, and hence this implies that our two approaches are suitable for optimization problems. In the case of the F_{OneMax} and $F_{Plateau}$ problems the highest overhead is provoked by the configuration using the Fast Ethernet network with eight processors. F_{Muehl} problem is the one that wastes more time in communication. As we said before, this is due to the fact that the size of the population is larger than in the other problems, and therefore, the quantity of information exchanged is also higher.

We also observe that the unbalanced version produces a higher communication cost than the balanced one. This is quite surprising because the unbalanced version is expected to perform a lower number of exchanges. The reason of this result is that the time that the balanced version loses in the last step of communication (sending the final results to the master) compensates the fact of doing the intermediate communication phase when the information is evenly distributed among the processors.

Another interesting issue is that the communication time is in general small and it does not increase significantly when we use a higher number of processors. This demonstrates that our two proposals have a very good scalability, although it is necessary to check this result in a parallel platform with a larger number of processes (work in progress). Scalability is a very important feature of any algorithm if it is expected to be actually useful for the research community.

Our last interesting observations come from comparing the final fitness value obtained by our approaches. We have configured the algorithms to stop when they reach a predefined number of generations (20 generations). Our aim however is to also prove that there are no significant differences between the algorithms, number of processors and network technologies. Due to space limitation we not show tables with the numerical results. The results show a similar behavior for all the methods. This is an expected result since our algorithms follows a master/slave model that does not change the search dynamics of the sequential algorithm. For the two simplest problems, F_{OneMax} (values between 298.12 and 299.66) and $F_{Plateau}$ (values between 85.04 and 87.92), our algorithms find the optimum quickly, and therefore the results are very good (best solution near to the optimum). On the other hand, the F_{Muehl} problem is a more complex one and the algorithms can not achieve the optimal solution (values between 174.02 and 176.97) due to the small number of generations used, but it is expected the algorithm reached the optimum with a slightly larger number of generations. The statistical tests throw negative results (p-value is larger than 0.05) in all the cases, indicating that the differences are not statistically significative. This statistical analysis includes all the algorithm versus algorithm tests, and all the six combination of algorithms and network technology tests (two algorithms times three networks).

5 CONCLUSIONS

In this paper we analyze the behavior of two parallel versions of the $PADA$ algorithm over different network technologies. Also, we study the effects of these network and the number of processors in the behavior of pPADA. We also study the numerical behavior of the algorithms.

In general, we have observed that the parallel model allows a considerably reduction in the execution time with respect to the sequential version (percentages between 39.38% and 87.40%), obtaining a very good efficiency (even superlinear in some cases). The parallel PADA algorithms also show a good scalability, a promising feature appreciated in actual applications. Also, the balanced version of pPADA shows a better speedup/efficiency than the unbalanced one.

Analyzing the effect of the communication networks, we have observed that the executions times using Myrinet and Gigabit Ethernet are very similar each other and both technologies are faster than the Fast Ethernet one. Also, we noticed that the communication time is in general very low, allowing high speedups of our parallel methods. The Myrinet should have been performed better than Gigabit Ethernet, but its superior performance is not evident here since a small latency is not important in the algorithms studied. We are looking for more intense communication by adding other parallel EDAs and other settings (like island EDAs with tight migration between islands).

From the numerical point of view we have observed that the best solutions found in the last generation are very similar in all the algorithms. In fact, statistical tests demonstrate that significant differences do not exist. This result was expected since the master/slave parallel model does not modify the dynamics of the method.

As a future work we plan to extend this paper to study the scalability of the algorithms in real world problems. We are also working to extend these algorithms to continuous domains and to include new operators and methods for sampling individuals.

REFERENCES

[1] S. Acid and L. M. de Campos. Approximation of causal networks by polytrees: an empirical study. In B. Bouchon-Meunier, R. R. yager, and L. A. Zadeh, editors, *Advances in Intelligent Computing*, LNCS 945, pages 149–158, 1995. Springer-Verlag, Berlin, 1995.

[2] T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, 1997.

[3] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2002.

[4] J. A. Lozano, R. Sagarna, and P. Larrañaga. Parallel Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academis Publishers, 2002.

[5] J. Madera, E. Alba, and A. Ochoa. Parallel estimation of distribution algorithms. In E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms.* John Wiley & Sons, Inc., 2005.

[6] A. Mendiburu, J.A. Lozano, and J. Miguel-Alonso. Parallel estimation of distribution algorithms: New approaches. Technical Report EHU-KAT-IK-1-3, Department of Computer Architecture and Technology, The University of the Basque Country, 2003.

[7] A. Mendiburu, J. Miguel-Alonso, and J.A. Lozano. Implementation and performance evaluation of a parallelization of estimation of bayesian networks algorithms. *Parallel Processing Letters*, 16(1):133–148, 2006.

[8] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In *LNCS 1411: PPSN IV*, pages 178–187, 1996.

[9] J. Ocenásek and J. Schwarz. The parallel bayesian optimization algorithm. In *In Proceedings of the European Symposium on Computational Inteligence*, pages 61–67, 2000. Slovak Republic.

[10] J. Ocenásek and J. Schwarz. The distributed bayesian optimization algorithm for combinatorial optimization. In *In EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control*, pages 115–120, 2001. Athens, Greece.

[11] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *GECCO-99*, volume 1, pages 525–532. Morgan Kaufmann Publishers, 1999. Orlando, FL.

[12] M. Soto, A. Ochoa, S. Acid, and L. M. de Campos. Introducing the polytree aproximation of distribution algorithm. In *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, pages 360–367, 1999. La Habana.