Introducing Belief Propagation in Estimation of Distribution Algorithms: A Parallel Framework

A. Mendiburu[1], R. Santana[2] and J. A. Lozano[2]

Intelligent Systems Group
[1]Department of Computer Architecture and Technology
[2]Department of Computer Science and Artificial Intelligence
University of the Basque Country
Paseo Manuel de Lardizábal 1, 20080. San Sebastian - Donostia, Spain
alex@ehu.es, rsantana@ehu.es, ja.lozano@ehu.es

Abstract

This paper incorporates Belief Propagation into an instance of Estimation of Distribution Algorithms called Estimation of Bayesian Networks Algorithm. Estimation of Bayesian Networks Algorithm learns a Bayesian network at each step. The objective of the proposed variation is to increase the search capabilities by extracting information of the, computationally costly to learn, Bayesian network. Belief Propagation applied to graphs with cycles, allows to find (with a low computational cost), in many scenarios, the point with the highest probability of a Bayesian network. We carry out some experiments to show how this modification can increase the potentialities of Estimation of Distribution Algorithms. Due to the computational time implied in the resolution of high dimensional optimization problems, we give a parallel version of the Belief Propagation algorithm for graphs with cycles and introduce it in a parallel framework for Estimation of Distribution Algorithms [13]. In addition we point out many ideas on how to incorporate Belief Propagation algorithms into Estimation Distribution Algorithms.

1 Introduction

Estimation of Distribution Algorithms (EDAs) [18, 11] have turned in the last ten years in a lively research area inside the Evolutionary Computation field. These algorithms are characterized by the learning and posterior sampling of a probability distribution learnt from the selected individuals at each step. The most sophisticated methods use probabilistic graphical models to encode the probability distribution. Particularly, in the field of combinatorial optimization, Bayesian networks are the most commonly used formalism [10, 24, 17]. Although the algorithms using Bayesian networks get the best results in terms of objective function value they come with the cost of the learning of the probabilistic graphical model. Learning a Bayesian network is an NP-hard [2] problem and therefore local (in general heuristics) algorithms need to be used at each step. The computational cost implied by learning dominates the algorithm runtime (we do

not consider the cost of evaluating the objective function) even with the use of these local search algorithms. Given that situation, researchers in the field of EDAs are wondering how to take advantage of the learnt graphical model. A first approach is to use the Bayesian network learnt at each step as a model of the process being optimized [27]. In this paper we follow a different approach, we plan to increase EDAs search capabilities by looking for the point with the highest probability in the graphical model. To do that we use belief propagation algorithms in graphs with cycles.

Belief Propagation (BP) algorithms [22, 32, 33] are commonly used in graphical models to carry out inference tasks. For instance, they can be used to reason in probabilistic graphical models (calculate marginal probabilities or posterior probabilities), or to calculate the point with the highest probability. The general problem of carrying out inference in graphical models is NP-complete, therefore BP can only be applied to small models (with small size cliques). Recently the connection between BP algorithms with techniques coming from the field of statistical physics, particularly with algorithms developed to minimize the free energy [33], has brought to the field new ideas. One of these ideas is the application of BP algorithms in graphs with cycles (we will call from now on Loopy Belief Propagation (LBP)). While the convenient convergence and exactness properties of BP in acyclic graphs are lost, it has been widely proved in many applications that these algorithms often produce the expected results with a low computational cost [3, 4, 32].

We propose in this paper the use of LBP in the sampling phase of an Estimation of Bayesian Networks Algorithm (EBNA). At each step of the algorithm and using the learnt Bayesian network the point that receives the highest probability will be looked for. That individual is incorporated into the next population.

1.1 Related work

There are some previous reports on the application of BP algorithms in EDAs. We consider interesting to briefly review this related work here.

In [21] BP is applied to improve the results achieved by the polytree distribution algorithm (PADA). The objective is to construct higher-order marginal distributions from the bivariate marginals corresponding to a polytree distribution. These results are extended in [8] to the factorized distribution algorithm (FDA) with fixed structure. In contrast to the previous approaches, our proposal allows the application of the BP algorithm on more complex models than those learned by PADA and the structure is not fixed from the beginning as is the case of the FDA.

In [30] an algorithm that calculates the most probable configurations in the context of optimization by EDAs was introduced. The algorithm was applied to obtain the most probable configurations of the univariate marginal model used by the Univariate Marginal Distribution Algorithm [15], and models based on trees and polytrees. These results were extended in [26] to pairwise Markov networks which are covered by the more general factor graphs we used in our proposal. However, the results presented in these papers have shown that EDAs that combine Probabilistic Logic Sampling (PLS) with the computation of the most probable configuration are in general more efficient in terms of function evaluations than those that only use PLS.

Recently, more sophisticated BP algorithms have been used in the general context of optimization based on probabilistic models for obtaining higher order consistent marginal probabilities [16], and the most probable configurations of the model [9]. Also in these cases, the structure of the problem is known a priori.

Our proposal rests in the general schema of EDAs and it is more general than the previously presented as it allows the use of non-fixed unrestricted graphical models. In addition, as previously explained, the use of EBNAs spread EDAs. Taking into account that there are several implementations of this algorithm, the generality of our approach allows it to be inserted in such implementations.

The rest of the paper is organized as follows. Section 2 briefly introduces EDAs and BP algorithms. In addition, the new proposal will be presented. A parallel version of the LBP algorithm for factor graphs will be explained in Section 3 together with the parallel framework where it is incorporated. Section 4 presents the experimental results and the parallel performance analysis of the algorithm. Finally Section 5 concludes the paper.

2 Estimation of Distribution Algorithms and Loopy Belief Propagation

2.1 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) [18, 11] are a set of algorithms inside the Evolutionary Computation field. They are based on populations of solutions as Genetic Algorithms but, substitute the reproduction operators by the learning and sampling of a probability distribution. Algorithm 1 shows a pseudocode for a general EDA.

Algorithm 1: **Main scheme of the EDA approach**

1 $D_0 \leftarrow$ Generate M individuals randomly
2 $t = 1$
3 **do** {
4 $\quad D^s_{t-1} \leftarrow$ Select $N \leq M$ individuals from D_{t-1} according to a selection method
5 $\quad p_t(\boldsymbol{x}) = p(\boldsymbol{x} \mid D^s_{t-1}) \leftarrow$ Estimate the joint probability of selected individuals
6 $\quad D_t \leftarrow$ Sample M individuals (the new population) from $p_t(\boldsymbol{x})$
7 } **until** A stop criterion is met

Different algorithms can be given by restricting the complexity of the probability distribution learnt at each step. For instance, the most simple algorithms consider that the variables are independent. The most sophisticated use probabilistic graphical models to codify the probability distribution of the selected individuals at each step. Particularly, we are interested in Estimation of Bayesian Networks Algorithm (EBNA) [10], an algorithm that learns and samples a Bayesian network at each step. A pseudocode for an EBNA can be seen in Algorithm 2.

The most costly step in EBNA (apart from the computation of the objective function) is the learning of the Bayesian network (this is usually done by means of a local search algorithm that adds at each step the arc that increases the score the most). The sampling step is usually done by means of PLS [7].

Recently, researchers in the field have thought on how to take advantage of the probability model learnt at each step, given the computational cost spent on it. One main line of research has been the use of the Bayesian network learnt at each step for modelling purposes. For instance, in

Algorithm 2: **EBNA**$_{BIC}$

$BN_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ where S_0 is an arc-less DAG, and $\boldsymbol{\theta}^0$ is uniform
$p_0(\mathbf{x}) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$
$D_0 \leftarrow$ Sample M individuals from $p_0(\mathbf{x})$
t $\leftarrow 1$
do {
  $D_{t-1}^{Se} \leftarrow$ Select N individuals from D_{t-1}
  $S_t^* \leftarrow$ Using a local search method find one network structure according to the BIC score
  $\boldsymbol{\theta}^t \leftarrow$ Calculate θ_{ijk}^t using D_{t-1}^{Se} as the data set
  $BN_t \leftarrow (S_t^*, \boldsymbol{\theta}^t)$
  $D_t \leftarrow$ Sample M individuals from BN_t
} **until** Stop criterion is met

[27] the probabilistic model learnt at each step is used to shape the phenomenon that is optimized. Another way to use the models, in the field of black-box optimization, is to discover dependencies or relationships between the variables of the problem. In this paper we focus on using the Bayesian networks learnt at each step to improve the search. Our objective is to modify the sampling process to include methods that look for the points with the highest probability.

2.2 Belief Propagation Algorithms

Belief Propagation (BP) [22] is a widely recognized method to solve graphical models inference problems. It is mainly applied to two different situations: (1) when the goal is to obtain marginal probabilities for some of the variables in the problem, and (2) with the aim of searching for the most probable global state of a problem given its model. These two variants are also known as the *sum-product* and *max-product* algorithms.

BP algorithm has been proved to be efficient on tree-shaped structures, and empirical experiments have often shown good approximate outcomes even when applied to cyclic graphs. This has been widely demonstrated in many applications including low-density parity-check codes [25], turbo codes [12], image processing [6], or optimization [1].

We illustrate BP by means of a probabilistic graphical model called factor graph. Factor graphs are bipartite graphs with two different types of nodes: variable nodes and factor nodes. Each variable node identifies a single variable X_i that can take values from a (usually discrete) domain, while factor nodes f_j represent different functions whose arguments are subsets of variables. This is graphically represented by edges that connect a particular function node with its variable nodes (arguments). Figure 1 shows a simple factor graph with six variable nodes $\{X_1, X_2, \ldots, X_6\}$ and three factor nodes $\{f_a, f_b, f_c\}$.

Factor graphs are appropriate to represent those cases in which the joint probability distribution can be expressed as a factorization of several local functions:

$$g(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{j \epsilon J} f_j(\boldsymbol{x_j}) \tag{1}$$

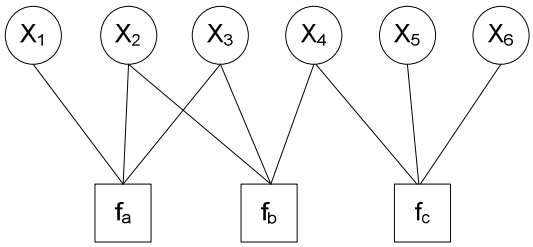

Figure 1: Example of a factor graph.

where $Z = \sum_{\boldsymbol{x}} \prod_{j \epsilon J} f_j(\boldsymbol{x_j})$ is a normalization constant, n is the number of variable nodes, J is a discrete index set, $\boldsymbol{X_j}$ is a subset of $\{X_1, \ldots, X_n\}$, and $f_j(\boldsymbol{x_j})$ is a function containing the variables of $\boldsymbol{X_j}$ as arguments.

Applying this factorization to the factor graph presented in Figure 1, the joint probability distribution would result in:

$$g(x_1, \ldots, x_6) = \frac{1}{Z} f_a(x_1, x_2, x_3) f_b(x_2, x_3, x_4) f_c(x_4, x_5, x_6) \tag{2}$$

The main characteristic of the BP algorithm is that the inference is done using message-passing between nodes. Each node sends and receives messages until a stable situation is reached. Messages, locally calculated by each node, comprise statistical information concerning neighbor nodes.

When using BP with factor graphs, two kinds of messages are identified: messages $n_{i \to a}(x_i)$ sent from a variable node i to a factor node a, and messages $m_{a \to i}(x_i)$ sent from a factor node a to a variable node i. Note that a message is sent for every value of each variable X_i.

These messages are updated according to the following rules:

$$n_{i \to a}(x_i) := \prod_{c \epsilon N(i) \backslash a} m_{c \to i}(x_i) \tag{3}$$

$$m_{a \to i}(x_i) := \sum_{\boldsymbol{x_a} \backslash x_i} f_a(\boldsymbol{x_a}) \prod_{j \epsilon N(a) \backslash i} n_{j \to a}(x_j) \tag{4}$$

$$m_{a \to i}(x_i) := \arg \max_{\boldsymbol{x_a} \backslash x_i} \{ f_a(\boldsymbol{x_a}) \prod_{j \epsilon N(a) \backslash i} n_{j \to a}(x_j) \} \tag{5}$$

where $N(i) \backslash a$ represents all the neighboring factor nodes of node i excluding node a, and $\sum_{\boldsymbol{x_a} \backslash x_i}$ expresses that the sum is completed taking into account all the possible values that all variables but X_i in $\boldsymbol{X}_a$ can take –while variable X_i takes its x_i value.

Equations 3 and 4 are used when marginal probabilities are looked for (sum-product). By contrast, in order to obtain the most probable configurations (max-product), Equations 3 and 5 should be applied.

When the algorithm converges (i.e. messages do not change), marginal functions (sum-product) or max-marginals (max-product) are obtained as the normalized product of all messages received by X_i:

$$g_i(x_i) \propto \prod_{a \epsilon N(i)} m_{a \to i}(x_i) \tag{6}$$

Regarding the max-product approach, when the algorithm converges to the most probable value, each variable in the optimal solution is assigned the value given by the configuration with the highest probability at each max-marginal. Some theoretical results on BP and modifications for maximization can be found in [31].

Given a Bayesian network it is possible to translate it into a factor graph by assigning a factor node to each variable and its parent set (a variable node is created by each variable of the problem).

2.3 Incorporation of LBP into EBNA

We devise a new EDA based on EBNA. The sampling process of EBNA is modified by using LBP. The proposal is to sample M-1 individuals by PLS and using LBP to obtain an additional individual. In order to use LBP, we firstly turn the Bayesian network in a factor graph and then LBP is applied in the factor graph to obtain the new individual. A pseudocode for the proposal can be seen in Algorithm 3.

Algorithm 3: **EBNA$_{BIC}$+LBP**

1. $BN_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ where S_0 is an arc-less DAG, and $\boldsymbol{\theta}^0$ is uniform
2. $p_0(\mathbf{x}) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$
3. $D_0 \leftarrow$ Sample M individuals from $p_0(\mathbf{x})$
4. t $\leftarrow 1$
5. **do** {
6. $\quad D_{t-1}^{Se} \leftarrow$ Select N individuals from D_{t-1}
7. $\quad S_t^* \leftarrow$ Using a search method find one network structure according to the BIC score
8. $\quad \boldsymbol{\theta}^t \leftarrow$ Calculate θ_{ijk}^t using D_{t-1}^{Se} as the data set
9. $\quad BN_t \leftarrow (S_t^*, \boldsymbol{\theta}^t)$
10. $\quad D_t \leftarrow$ Sample $M-1$ individuals from BN_t
11. $\quad$ Convert BN_t into a factor graph
12. $\quad$ Apply LBP to find consistent max-marginal configurations
13. $\quad$ Recover the most probable configuration from the max-marginals
14. } **until** Stop criterion is met

3 Parallel framework for EBNA and LBP

In recent years, the availability of computer clusters or even grids have encouraged the design of parallel applications. Following this trend, we decided to design a parallel framework for EDAs that can be executed efficiently in multiprocessors or clusters of computers.

In [13], different EDAs were analyzed and parallelized trying to make them efficient (in terms of the execution time) when facing complex problems. One of those parallel EDAs is EBNA. As mentioned before, this algorithm uses a Bayesian network to represent the (in)dependencies between the variables. It starts with an empty structure, and performs modifications on it trying

to obtain the best representation. In order to measure the quality of the network learnt, there are several scores that can be used. For example, EBNA$_{BIC}$ algorithm uses the penalized maximum likelihood score denoted by BIC (Bayesian Information Criterion) [29].

The learning process starts from the scratch, and at each step, all possible arc modifications (addition or deletion) will be taken into account, calculating the BIC score for each of these modifications. The parallel proposal for this algorithm focuses precisely on this learning step, designing a manager-worker scheme that allows the manager to control the whole execution of the algorithm and to distribute the work when necessary. In this case, each possible modification will be calculated in a parallel way by each worker, reducing notably the execution time. In addition, depending on the complexity of the fitness function used to evaluate each individual, it can be interesting to sample and evaluate new individuals in a parallel way. A detailed explanation of this parallel approach as well as different approaches for other EDAs can be consulted in [13, 14]. In this paper, we focus on the new contribution, that is, the design of a flexible and parallel version of LBP that will be later included in the sampling phase of EBNA.

3.1 Analysis of the LBP algorithm

As described in the previous sections, LBP is a widely studied and used algorithm and has been rediscovered and adapted repeatedly to particular problems. Thus, different implementations have been developed since the algorithm was first proposed, although most of them are sequential versions and have some limitations regarding the number of nodes, neighbors, or scheduling policies. Concerning parallel versions, to the best of our knowledge, implementations have been designed only for exact inference methods [19].

Taking into account that a parallel framework for EDAs was previously implemented, we decided to follow this trend designing a parallel version of the BP algorithm for factor graphs that can be executed independently or, as is our case, inside an EDA instance. As the size and complexity of the problems that researchers face nowadays is growing notably, we consider more appropriate to develop a parallel version for LBP more than a sequential one (even when the particular nature of the algorithm has high communication needs).

In order to do that, we carefully analyzed the characteristics of the algorithm, with the aim of designing a flexible tool that could be tuned and used with different problems, just selecting, for each parameter, the appropriate value or set of values. In some cases, allowing the user to establish some particular conditions, can make possible to improve (when affordable) the performance of the algorithm. In other cases, it allows to complete several tests in order to observe to which extent initial decisions can condition the final results.

When studying the BP algorithm, we detected three main parameters susceptible to be defined by the user: (1) scheduling policies –i.e. when and how the messages are sent and received– (2) stopping criteria –that fix the conditions that make the program to finish– and (3) initial values for some parameters –including for example, the initial values of the messages.

3.1.1 Scheduling policies

An important aspect when using the LBP algorithm is the way messages are spread through the nodes that conform the factor graph. In many implementations, scheduling is designed following a synchronous model, where a clock triggers the message-sending. In our implementation, we propose a rule-based scheduling. This way, the user can determine the particular conditions that

Table 1: Example of a scheduling based on sets of messages

node	RcvSet	SndSet
X_2	f_a,f_b	f_a,f_b
X_3	f_a,f_b	f_a,f_b
X_4	f_b,f_c	f_b,f_c
f_a	X_1,X_2,X_3	X_1,X_2,X_3
f_b	X_2,X_3,X_4	X_2,X_3,X_4
f_c	X_4,X_5,X_6	X_4,X_5,X_6

govern the behavior of each node, allowing to provide different rules for each node or set of nodes. For each node, two types of rules can be defined:

Number of messages: This is the simplest rule. This rule is triggered when the node receives a fixed number of messages. Then, it calculates the new messages, and sends them to the neighbor nodes from which messages were not received.

Sets of messages: This is a more complete rule, which allows to define different pairs ($RcvSet$,$SndSet$). $RcvSet$ and $SndSet$ represent subsets of nodes –including the empty set for nodes that start sending. When messages have been received from all the nodes contained in $RcvSet$, new messages will be calculated and sent to all the nodes identified in $SndSet$.

In order to illustrate these scheduling options, two examples are provided based on the factor graph shown in Figure 1. One of the options is to use a scheduling based on the number of messages received. Suppose we fix this number to 1 for all the nodes (different values for each node could also be used). Under this condition, each time a node receives a message it calculates its messages and sends them to all the neighbors (except to the one that sent the message). For instance, if f_b receives a message from X_3, it will compute and send messages to X_2 and X_4.

Another option is to use a scheduling based on sets of messages that allow to create a more flexible scheduling. For example, it would be interesting to define a scheduling policy such that initially all variable nodes start sending messages. Next, nodes (either variables or factors) only calculate and send new messages after receiving messages from all their neighbors. Table 1 shows the rules that need to be fixed under configuration in the factor graph of Figure 1.

3.1.2 Stopping criteria

When the execution starts, each node acts following the rules and the parameters set in the initial configuration. According to these parameters, each node will receive, calculate, and send different messages. In an acyclic structure, BP algorithm has been proved to converge –i.e. to reach a stable situation with fixed message values. However, when BP is applied to cyclic structures, the algorithm might obtain good results in some cases but it can not be guaranteed that a stable situation will be reached.

That is why different stopping criteria need to be defined in order to guarantee that a particular execution will always end up. Taking into account the different situations that can happen during

a execution, we have defined three different stopping criteria that are independently checked by each node. The algorithm will stop if either:

- In the last i iterations (message calculations) the same message values are obtained, or
- In the last i iterations the same message sequence is repeated. That is, a cyclic situation is detected, where message values m_1,m_2,...,m_i are repeatedly obtained, or
- A maximum given number of messages is calculated.

3.1.3 Initial settings

In addition to scheduling policies and stopping criteria, there are also different parameters that have to be considered. Examples of those required to be manually defined by the user are the function values for each factor node (problem depending), and the initial message values that nodes will take. It is also possible to prefer that the messages sent by a node to its neighbors should have always the same (fixed) value.

Other additional settings are the following:

Allowed difference: when comparing two messages, they are said to have the same value when the difference between them is lower than the value fixed in this parameter.

Maximum number of messages: this was defined to stop the execution of the program if a stable situation was not reached after calculating that number of messages.

Number of comparisons needed: establishes the number of comparisons (between messages) that should be done before considering a situation stable.

Cache size: determines the number of messages that each node will store.

Algorithm: to decide if either sum-product or max-product will be used.

3.2 Parallel design for the LBP algorithm

Regarding the nature of the LBP algorithm and looking at its behavior, a straightforward parallel approach could be the following: each node (process) is related to a CPU, being only responsible for receiving, calculating, and sending messages according to the scheduling policies. However, this approach depends directly on the size of the problem and the computational resources available. Note that for a network with a thousand of nodes, it will be necessary to use one thousand of CPUs, which are not available for most of researchers. When the number of processors available is lower than the number of nodes in the factor graph, a possible solution could be to assign more than one process (node) to each processor. However, overcharging processors excessively is not advisable since it could negatively affect the performance of the algorithm.

Based on this initial design, we propose a solution where each process is responsible for a group of nodes of the factor graph. This way, the number of processors does not have to equal the number of nodes, making the algorithm affordable for a wider set of scenarios.

As well as parallel EBNA, the application has been designed mixing Message Passing Interface (MPI) and POSIX threads. At the MPI level, the well-known manager-worker scheme was chosen. The manager-process is responsible for loading the problem structure and the user-defined settings.

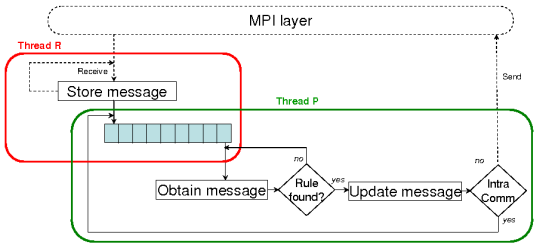

Figure 2: Design of the worker process. There are two different kind of threads: receive MPI messages (thread R), and calculate and send new messages (thread(s) P).

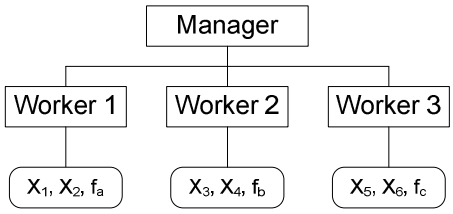

Figure 3: Manager-worker scheme. Node distribution is shown for the factor graph in Figure 1 using three workers.

Once it has sent all the information to the workers, it waits until all workers have finished, gathering all partial results and storing them.

Regarding worker-processes, each of them is responsible for a set of nodes (variables or factors). The distribution of nodes between workers is fixed once the application starts, and is kept unchanged until the execution ends. Inside the worker, a shift-based scheme is used, where each node will be managed sequentially checking if any of its rules is fulfilled; Every time this happens, messages are calculated and sent. The messages calculated by each node will be sent to other workers or queued in the sending worker depending on whether the sender and receiver belong to the same worker or not. Workers were implemented using two different threads: one thread to receive MPI messages (see Thread R in Figure 2), and one or more threads to process received messages, calculate, and send new messages (see Thread P in Figure 2).

Regarding the distribution of nodes of the factor graph, Figure 3 illustrates this manager-worker scheme assuming that three workers are running. The node distribution presented is determined according to the factor graph introduced in Figure 1. In this case, the distribution was done trying to equally distribute variable and factor nodes. However, a deeper study to weigh the effect that different distributions could have on the performance of the algorithm is regarded as a future work to be done.

Figures 4 and 5 describe the pseudo-codes for the manager and workers.

Algorithm 4: **Pseudo-code for the LBP manager process**

```
1  Get the structure of the factor graph and the configuration from file
2  Send the structure and configuration to the workers
3  do {
4      Wait for a worker to finish
5      Receive results from the worker
6  } until All the workers finish
7  Send stop order to the workers
8  Show results
```

Algorithm 5: **Pseudo-code for the LBP worker processes**

```
1  Receive the structure and the configuration from the manager
2  for n = 1 to number of nodes in worker
3      if node starts sending
4          if a scheduling-rule is found
5              Calculate messages
6              Send messages according to the rule
7  do {
8      Wait for a message
9      for n = 1 to number of active nodes
10         if a scheduling-rule is found
11             Calculate messages
12             Check stopping criteria
13             Send messages according to the rule
14 } until a stopping criteria is met
15 Send results to the manager
16 Wait for the stop order
```

4 Experiments

In order to test the quality of the proposed approach, we completed several experiments from two different scopes: (1) behavior of the EBNA-LBP algorithm compared to EBNA, and (2) efficiency (in terms of the execution time) of the parallel framework.

4.1 Comparison of EBNA-LBP and EBNA

In this section, we compare the behavior of EBNA and EBNA-LBP for a number of instances of the generalized Ising model. First, the Ising problem is introduced. We then present convergence results for EBNA and EBNA-LBP. Finally, the behavior of both algorithms is investigated in more detail using different measures of performance and elucidating the role of LBP in the improvements achieved by EBNA-LBP.

4.1.1 Testbed used to evaluate the algorithms

The generalized Ising model is described by the energy functional (Hamiltonian) (see Equation 7) where L is the set of sites called a lattice. Each spin variable σ_i at site $i \in L$ either takes the value 1 or -1. One specific choice of values for the spin variables is called a configuration. The constants J_{ij} are the interaction coefficients. In our experiments we take, $h_i = 0$, $\forall i \in L$. The ground state is the configuration with minimum energy. We pose the problem as the maximization of the negative energy.

$$H = -\sum_{i<j\in L} J_{ij}\sigma_i\sigma_j - \sum_{i\in L} h_i\sigma_i \tag{7}$$

Four random instances of the Ising model were generated with $n = 100$. To generate a random instance where $J_{ij} \in \{-1, 1\}$, each coupling was set to -1 with probability 0.5, otherwise the constant was set to $+1$. The results were verified using the Spin Glass Ground State server, provided by the group of Prof. Michael Juenger[1].

4.1.2 Convergence results

EBNA and EBNA-LBP were run under the same conditions and using the same parameters.

1. Population size $N = 5,000$.
2. Random generation of the initial population.
3. Truncation selection with $T = 0.5$.
4. Replacement is done as follows: $N - 1$ individuals are created and mixed with the individuals of the present population. Then, the worst $N - 1$ individuals are removed from the population. This way, the best individual in each generation is guaranteed to be in the next one.
5. The termination criterion used was to reach a maximum number of generations (250).
6. For the LBP algorithm, allowed difference is 1.0e-3, maximum number of messages are 30, cache-size is 20, 10 is the number of comparisons needed to fix a node, and the algorithm is max-product.

Since the objective of our experiment is to compare the two EDAs, we have not tuned the parameters to obtain the best performance. We have neither applied any local optimization algorithm usually employed [23] to speed up the search of EDAs and attain better results. For each of the four Ising instances ($n = 100$), 30 independent runs were executed. The results of the experiments are summarized in Table 2. In this table, the optimum values for each of the instances, as well as the mean, best and worst fitness values reached by the algorithms are shown.

EBNA-LBP achieves a better average of the best solutions found for all the instances. To determine whether differences between the algorithms are statistically significant we have used the Kruskal-Wallis test to accept or reject the null hypothesis that the samples have been taken from equal populations. The test significance level was 0.01. For all the instances considered, significant statistical differences have been found between both EDAs.

[1] www.informatik.uni-koeln.de/ls_juenger/projects/sgs.html

Table 2: Results of EBNA and EBNA-LBP in four different instances of the Ising model.

	Inst. 1		Inst. 2		Inst. 3		Inst. 4	
	EBNA	EBNA-LBP	EBNA	EBNA-LBP	EBNA	EBNA-LBP	EBNA	EBNA-LBP
Mean	118.67	130.50	122.93	136.00	124.27	137.73	115.87	134.13
Best	132	136	138	138	142	142	134	138
Worst	108	124	106	130	106	134	102	126
Optimum		136		142		142		138

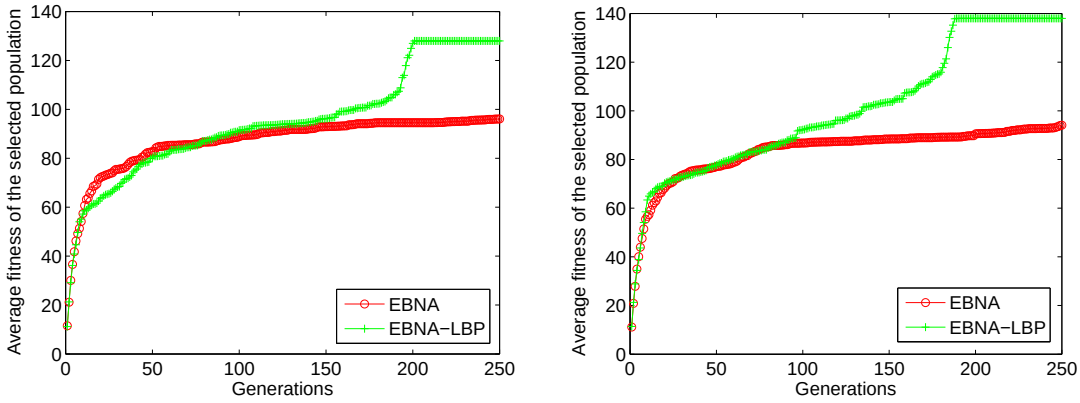

Figure 4: Average fitness of the selected population at each generation for EBNA and EBNA-LBP for the first (left) and second (right) instances.

4.1.3 Behavior of EBNA and EBNA-LBP: comparative analysis

We analyze in detail the different effects of adding LBP to EBNA. We select one representative run of EBNA and EBNA-LBP for the two first instances of the set. For the four different experiments, relevant information about the search is stored. Figures 4 to 7 display two graphs (one for each instance), and each graph shows the information corresponding to the runs executed for both algorithms.

Figure 4 shows the evolution of the average fitness of the selected population for EBNA and EBNA-LBP. The average fitness gives an idea of the general improvements in the population. It can be seen that while at the initial generations EBNA converges faster to better solutions, EBNA-LBP takes more time but then it is able to reach a higher fitness average than that of EBNA.

Another useful descriptor of the search process is the fitness variance and in Figure 5 its evolution for EBNA and EBNA-LBP is shown. It can be appreciated that the variance early decreases for EBNA while for EBNA-LBP it steadily increases. The increase in the population diversity seems to be a side-effect of the LBP application.

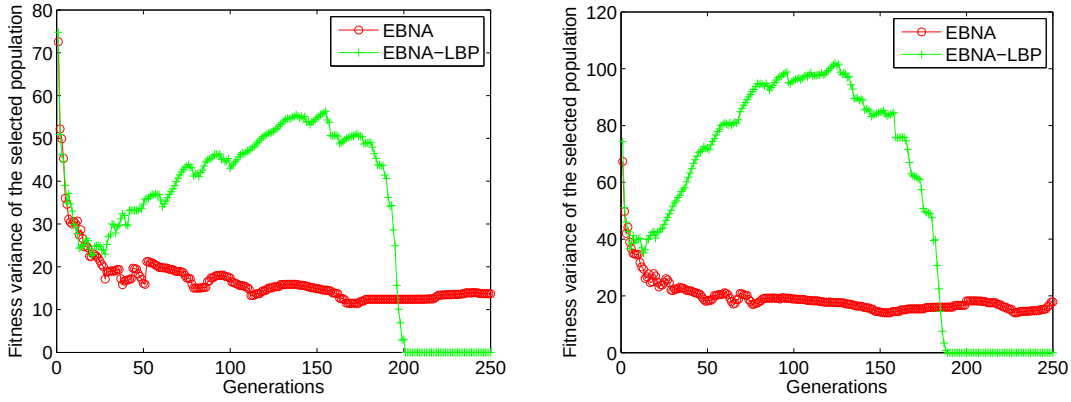

Figure 5: Fitness variance of the selected population at each generation for EBNA and EBNA-LBP for the first (left) and second (right) instances.

The small fitness variance exhibited by EBNA does not necessary imply that the population is genotypically less diverse (depending on the problem, different individuals can have the same fitness value). It turns out to be just the opposite case. Figure 6 shows the number of different individuals in the selected population at each generation of EBNA and EBNA-LBP. From this figure, it can be appreciated that the genotypic diversity of EBNA is always higher that that of EBNA-LBP. At the end of the evolution the diversity of EBNA-LBP has considerably diminished.

Finally, we inspect each of the solutions generated by LBP at each generation and determine when its fitness values were better than those obtained by EBNA-LBP to that generation. Results of EBNA-LBP corresponding to one run for each of the instances are shown in Figure 7.

At the beginning, LBP shows an erratic behavior in terms of the fitness of the solutions found. Not only very good solutions but also very poor solutions are found. LBP is able to guarantee a jump in the best value of the algorithm only in a few of the generations. However, these jumps play an essential role in the behavior of the algorithm allowing it to reach very good solutions. At the end of the runs, LBP is very stable finding solutions with the same fitness or values close to the best solution found so far. In order to give a clearer visualization of the behavior of the algorithm, Figure 7 only shows information up to generation 100. After this generation LBP exhibits a similar behavior.

4.2 Performance evaluation

We ran several experiments increasing the dimension of the problem in order to study the efficiency and scalability of the parallel framework. To this purpose, we ran EBNA-LBP for the Ising problem using three different instances with sizes 100, 256, and 324.

We kept the same parameters used in the experiments presented in the previous section, excepting the population size for $n = 256$ and $n = 324$ that was fixed to $N = 10,000$ and $N = 15,000$ respectively. In addition, the maximum number of generations was set to 50. This allowed us to complete several runs in a reasonable time maintaining the validity of the results.

Experiments were carried out in a cluster of computers with 4 nodes. Each node has two Intel

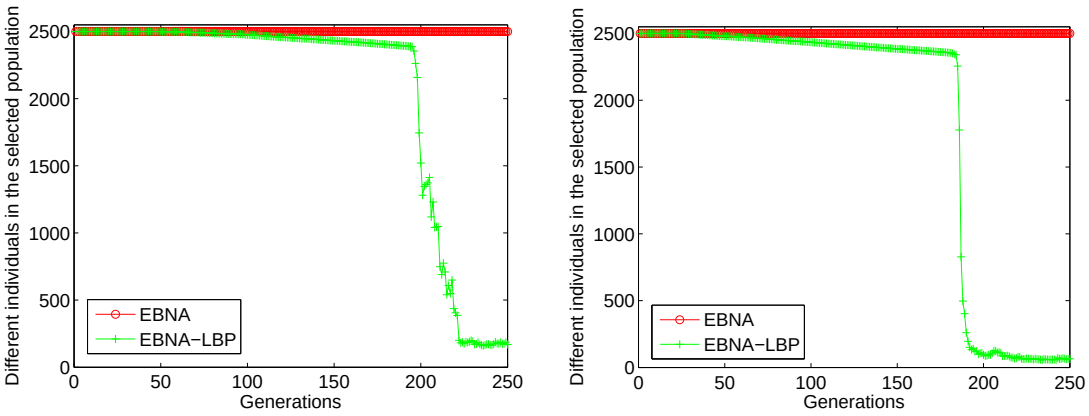

Figure 6: Number of different individuals of the selected population at each generation for EBNA and EBNA-LBP for the first (left) and second (right) instances.

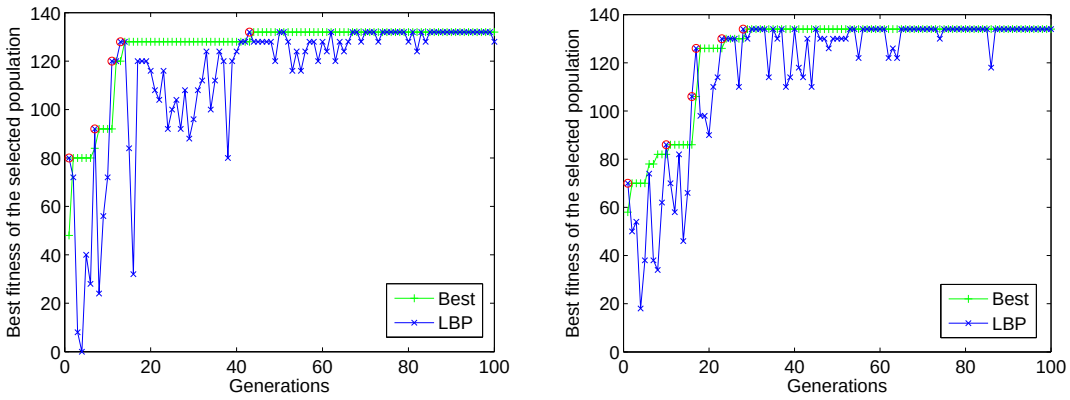

Figure 7: Best solution of the population and solution generated by LBP at each generation of EBNA-LBP for the first (left) and second (right) instances. The LBP solutions that improved the so far best solution found by the algorithm are represented with circles.

Table 3: Execution times and efficiency for different problem sizes (100, 256, and 324).

		# nodes (# CPUs)			
	Seq	1(2)	2(4)	3(6)	4(8)
		$n = 100$			
Time (s)	1,611	983	713	595	550
Efficiency		0.82	0.57	0.45	0.37
		$n = 256$			
Time (s)	31,712	17,139	9,516	6,714	5,373
Efficiency		0.93	0.83	0.79	0.74
		$n = 324$			
Time (s)	89,186	47,480	25,335	17,975	14,357
Efficiency		0.93	0.88	0.83	0.78

Xeon processors (2.4GHz), with 512KB of cache memory each and 2GB of (shared) RAM, all under Linux. The chosen MPI implementation is MPICH2[2] (version 1.0.5), installed using default parameters. C++ compiler is Intel's version 9.1. Nodes are interconnected using a switched Gigabit Ethernet network.

The results of the experiments are presented from the point of view of speed up and scalability of the parallel algorithm. Note that the parallel version has exactly the same behavior of the sequential algorithm but run faster, allowing it to solve harder problems more quickly. In Table 3, execution time and efficiency are presented. In addition, Figure 8 shows the speed up for the different problems and cpu combinations.

These two measures, speed up and efficiency, have been calculated as:

- Speed up = $\frac{sequential\ time}{parallel\ time}$,
- Efficiency = $\frac{speed\ up}{number\ of\ processors}$.

Looking at the results, it can be seen that, in general terms, the behavior of the parallel approach is satisfactory. However, we have selected different problem sizes to point out that scalability depends clearly on the complexity of the problem. For example, for a small problem size ($n = 100$), the scalability is quite poor, as the ratio communication/computation increases with the addition of more processors (less work for each worker while maintaining similar communication needs). In the other cases, it can be seen that when using medium problem sizes, the workload is big enough to be spread through more processors maintaining a notable scalability.

[2]http://www-unix.mcs.anl.gov/mpi/mpich2

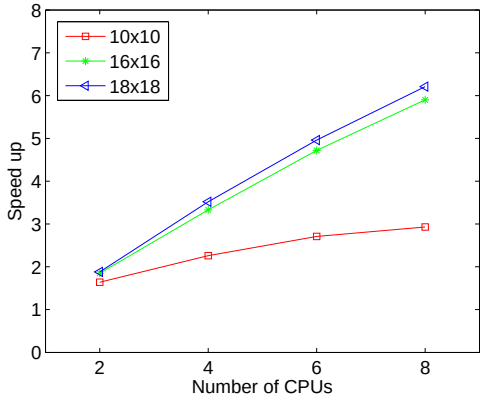

Figure 8: Speed up values for different problem sizes (100, 256, and 324).

5 Conclusions and future work

In this paper LBP has been added in the sampling phase of EBNA. This modification has allowed an improvement in the optimization capabilities of EBNA.

As the complexity and size of the problems is growing notably, and taking into account that the use of clusters of computers and/or multi-processors is generalizing, we designed a flexible and parallel version of the LBP algorithm that was included (as an additional module) in the parallel framework developed for EDAs. In this way, parallelism allows to face harder problems or reduce execution times. From the point of view of parallelism, good efficiency and scalability values are obtained using up to eight processors.

5.1 Discussion and future work

There are many open problems related to the use of LBP. Although we had a satisfactory behavior of LBP in the experiments carried out, this does not need to be always the case. As we have pointed out in the paper, the results of LBP can be unpredictable. For instance it could be the case that LBP does not converge, oscillating between different values. In that case it is possible to obtain an individual but there is not guarantee of being the point with the highest probability. Another problem can come from the max-marginals. If there are ties in the max-marginals, it is sometimes impossible to recover the highest probability point. Finally, we could consider the sensitivity of the algorithm to the parameters. In any case all these problems need to be addressed by the research community working in probabilistic graphical models.

The work presented in this paper is a first step in the use of LBP in EBNAs. An obvious way to continue this work is by using the k points with the highest probability. These points can be calculated using algorithms proposed by [20] and [32]. In this case the computational cost is higher and the use of the parallel version of LBP is crucial. Therefore, a balance should be found between the time spent in using LBP and that of learning the Bayesian network.

Furthermore, recent research on EDAs [28] has paid attention to the use of the models learned

during the search as a source of previously unknown information about the problem. In our case, an open question is to determine the possible impact of LBP in the accuracy (in terms of the mapping between the problem interactions and the dependencies learned by the model) of the models learned by EBNA. We have conducted initial experiments on this research trend. Preliminary results of EBNA-LBP for the Ising problem show that the factor graph accurately maps the underlying structure determined by the grid.

More research is needed to characterize the situations (e.g. period of the evolution) in which LBP produces more gains over PLS. The capacity of LBP to generate high fitness solutions seems to depend on the accuracy of the Bayesian network to represent the relevant features of the problem. Similarly, the probability given by the Bayesian network to the most probable configuration exerts an influence on the likelikood of obtaining this solution by using PLS. If this probability is too low, LBP is expected to exhibit a clear advantage over PLS. Both issues could be investigated in the context of EBNAs that use exact learning techniques [5].

References

[1] M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. *IEEE Transactions on Information Theory*, Accepted for publication.

[2] D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA, 1994.

[3] J. M. Coughlan and S. J. Ferreira. Finding deformable shapes using loopy belief propagation. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III*, pages 453–468, London, UK, 2002. Springer-Verlag.

[4] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, pages 159–166. Morgan Kaufmann Publishers, 2003.

[5] C. Echegoyen, J. A. Lozano, R. Santana, and P. Larrañaga. Exact Bayesian network learning in estimation of distribution algorithms. In *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pages 1051–1058. IEEE Press, 2007.

[6] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[7] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In J. F. Lemmer and L. N. Kanal, editors, *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence*, pages 149–164. Elsevier, 1988.

[8] R. Höns. *Estimation of distribution algorithms and minimum relative entropy.* PhD thesis, University of Bonn, Bonn, Germany, 2006.

[9] R. Höns, R. Santana, P. Larrañaga, and J. A. Lozano. Optimization by max-propagation using Kikuchi approximations. Submitted for publication, 2007.

[10] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 343–352, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[11] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.

[12] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng. Turbo Decoding as an Instance of Pearl's "Belief Propagation" Algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[13] A. Mendiburu, J. Lozano, and J. Miguel-Alonso. Parallel implementation of EDAs based on probabilistic graphical models. *IEEE Transactions on Evolutionary Computation*, 9(4):406–423, 2005.

[14] A. Mendiburu, J. Miguel-Alonso, J. A. Lozano, M. Ostra, and C. Ubide. Parallel edas to create multivariate calibration models for quantitative chemical applications. *J. Parallel Distrib. Comput.*, 66(8):1002–1013, 2006.

[15] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[16] H. Mühlenbein and R. Höns. The factorized distributions and the minimum relative entropy principle. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 11–38. Springer-Verlag, 2006.

[17] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.

[18] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin, 1996. Springer Verlag. LNCS 1141.

[19] V. K. Namasivayam and V. K. Prasanna. Scalable Parallel Implementation of Exact Inference in Bayesian Networks. In *ICPADS (1)*, pages 143–150. IEEE Computer Society, 2006.

[20] D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 2:159–173, 1998.

[21] A. Ochoa, R. Höns, M. R. Soto, and H. Mühlenbein. A maximum entropy approach to sampling in EDA- the single connected case. In *Progress in pattern recognition, speech and image analysis*, volume 2905 of *Lectures Notes in Computer Science*, pages 683–690, 2003.

[22] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, Palo Alto, CA, 1988.

[23] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms*. Studies in Fuzziness and Soft Computing. Springer, 2005.

[24] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, volume I, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

[25] T. J. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2001.

[26] R. Santana. *Advances in Probabilistic Graphical Models for Optimization and Learning: Applications in Protein Modelling*. PhD thesis, 2006.

[27] R. Santana, P. Larrañaga, and J. A. Lozano. Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 2007. In Press.

[28] R. Santana, P. Larrañaga, and J. A. Lozano. The role of a priori information in the minimization of contact potentials by means of estimation of distribution algorithms. In E. Marchiori, J. H. Moore, and J. C. Rajapakse, editors, *Proceedings of the Fifth European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 4447 of *Lecture Notes in Computer Science*, pages 247–257, 2007.

[29] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464, 1978.

[30] M. R. Soto. *A Single Connected Factorized Distribution Algorithm and Its Cost of Evaluation*. PhD thesis, University of Havana, Havana, Cuba, July 2003. In Spanish.

[31] M. Wainwright, T. Jaakkola, and A. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14:143–166, 2004.

[32] C. Yanover and Y. Weiss. Finding the M most probable configurations using loopy belief propagation. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[33] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.