A Parallel Framework For Loopy Belief Propagation

A. Mendiburu, R. Santana, J.A. Lozano, and E. Bengoetxea [*]
Intelligent Systems Group, Faculty of Computer Science
The University of the Basque Country, San Sebastian, Spain
{amendiburu,rsantana,ja.lozano,endika}@ehu.es

ABSTRACT

There are many innovative proposals introduced in the literature under the evolutionary computation field, from which estimation of distribution algorithms (EDAs) is one of them. Their main characteristic is the use of probabilistic models to represent the (in)dependencies between the variables of a concrete problem. Such probabilistic models have also been applied to the theoretical analysis of EDAs, providing a platform for the implementation of other optimization methods that can be incorporated into the EDA framework.

Some of these methods, typically used for probabilistic inference, are belief propagation algorithms. In this paper we present a parallel approach for one of these inference-based algorithms, the loopy belief propagation algorithm for factor graphs. Our parallel implementation was designed to provide an algorithm that can be executed in clusters of computers or multiprocessors in order to reduce the total execution time. In addition, this framework was also designed as a flexible tool where many parameters, such as scheduling rules or stopping criteria, can be adjusted according to the requirements of each particular experiment and problem.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Probabilistic algorithms; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Performance

Keywords

Belief propagation, Factor graphs, Parallelism

[*]This work was supported by the SAIOTEK Autoinmune (II) 2006 and Etortek research projects from the Basque Government, as well as by the Spanish Ministerio de Educación y Ciencia under grant TIN 2005-03824.

1. INTRODUCTION

Research on evolutionary computation has experienced a remarkable development with the integration of results from other research fields. In some cases, these results have contributed to overcome some of their previously acknowledged limitations and have allowed to be applied to more complex and costly problems. One relevant example of these developments is the conception of evolutionary algorithms that combine probabilistic models with machine learning techniques to guide the search. Among them, some of the best known example of such methods are estimation of distribution algorithms (EDAs) [18].

EDAs are characterized by the use of probabilistic graphical models which serve to represent relevant relationships between the variables of a concrete problem. It is generally assumed that it is possible to build a model of the search space that can be used to guide the search for the optimum solution. Several probabilistic models can be used to represent the interactions between the variables, representing different types and degree of probabilistic dependencies. The algorithms needed to learn the models and sample new solutions from them have different degrees of complexity [11].

EDAs have shown to be very efficient to solve a wide variety of optimization problems [11, 12]. But at the same time, the probabilistic graphical models usually applied in EDAs have shown to be useful for the theoretical analysis of EDAs (e.g. analyzing the relationship between the function structure and search distributions [17]) and as a framework for the implementation of other optimization methods that can be considered within the EDA paradigm.

Belief propagation (BP) algorithms are methods commonly employed in probabilistic graphical models for inference, that could also be used as optimization tools [21, 25, 26]. Given a probabilistic graphical model, BP algorithms can obtain either marginal probabilities or the most probable state of the distribution. This type of algorithms are based on applying message-passing operators until the algorithm converges –or until another stopping condition is satisfied. BP algorithms do not guarantee convergence for graphs with cycles but the literature shows many applications for which optimal solutions are reached [6, 7, 25].

BP method has been used in the optimization of additively decomposable functions from which the expression of the function is available [25]. The advantage over EDAs is that no function evaluation has to be performed (only local functions need to be calculated) and that EDA parameters such as population size and maximum number of generations are not used. In BP, the iterations of the message passing scheme substitute the generations of the evolutionary algorithm.

The limitation of BP in comparison to EDAs is that full information about the function structure must be available in order to be able to run BP. Thus, black-box optimization, as conducted by EDAs, can not be accomplished. A solution to this situation is the conception of hybrid algorithms that incorporate features from EDAs and BP as those in [9, 16, 23]. However, for this type of algorithms it is crucial to have flexible and scalable BP implementations. Flexibility is needed for allowing the manipulation of the BP components (i.e. initializations methods, scheduling policies, and stopping criteria) according to the search goals, and for the combination of these components with the probabilistic model learning and sampling steps used by EDAs. Scalability is essential for addressing hard optimization problems with many variables in reasonable time.

In this paper, we introduce a flexible parallel framework for BP over factor graphs [10]. Parallelization is shown to be a suitable alternative for achieving efficient BP implementations. Similarly to EDAs where parallelization has shown to achieve important improvements over serial implementations [14], we show that optimal solution of the optimization problems addressed can be efficiently found by using our implementation.

The rest of the paper is organized as follows: Section 2 introduces belief propagation and factor graphs. Sections 3 and 4 present an analysis of the BP algorithm and describe our parallel approach respectively. Results obtained from some preliminary experiments are discussed in Section 5. Finally, Section 6 provides general conclusions and future lines of research.

2. FACTOR GRAPHS AND BELIEF PROPAGATION

In probability theory a graphical model is a structure used to represent the (in)dependencies among random variables. In that structure (graph), variables are represented by nodes, and edges are used to express conditional (in)dependencies among variables.

Within the graphical models family, different representations exist: from models that use directed graphs, like Bayesian networks and Gaussian networks, to models that use undirected graphs like Markov random fields or factor graphs.

Thanks to the useful properties that graphical models provide –such as visual representation of dependencies and facility to obtain factorizations of the probability distribution– these are widely used in many problem domains, such as medical diagnosis, speech and image processing, or general optimization.

This paper focuses on factor graphs [10], which 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 $\{f_a, f_b, \text{and } f_c\}$.

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

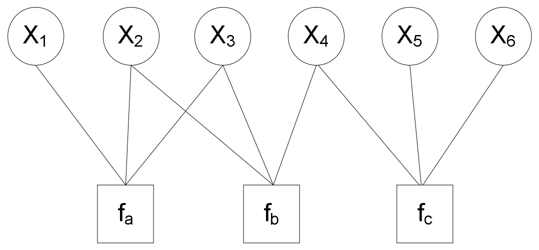

Figure 1: Example of a factor graph.

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

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:

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

Belief Propagation (BP) [21] 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 manyfold applications including low-density parity-check codes [22], turbo codes [13], image processing [8], or optimization [3].

The main characteristic of the BP algorithm is that the inference is calculated 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 kind 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.

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

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 [24].

2.1 Application of message-passing algorithms in optimization

Different variants of message-passing algorithms have been applied to function optimization. BP method has been used in the optimization of additively decomposable functions from which the expression of the function is available [25]. The idea is to associate a probability distribution to the function to be optimized in such a way that the most probable value of the distribution is reached for the solutions that optimize the function. Message passing is run until the algorithm eventually converges and the optimal solutions might be eventually recovered.

Propagation algorithms have also been employed to search for optimal solutions of constrained problems. In [5, 4], warning and survey propagation is introduced for the solution of the satisfiability problem (SAT). Warning and survey propagation belongs to a new type of propagation algorithms that intend to find satisfiable assignments to a set of clauses. The algorithm uses factor graphs to represent the structure of the problem and organizes the passing of messages.

Another proposal in the literature that relate BP and optimization algorithms in the context of EDAs can be found in [20], where the estimation of higher order marginals from small marginals is made by applying the iterative proportional fitting (IPF) procedure.

3. ANALYSIS OF THE BP ALGORITHM

As described in the previous sections, BP 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. Regarding parallel versions, to the best of our knowledge, recent works have been designed only for exact inference methods [19].

This fact encouraged us to design a parallel version of the BP algorithm for factor graphs. 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, initial message values.

3.1 Scheduling policies

An important aspect when using the BP 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 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 from 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.

3.2 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,

Table 1: Scheduling rules 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

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 cannot 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 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.

3.3 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.

4. PARALLEL DESIGN

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

The application was implemented using the widely known Message Passing Interface (MPI) paradigm [1]. MPI is an Application Programming Interface (API) that can be used by processes to communicate to each others, even when they are running in the same or in different computers. In the latter case, message exchange is performed using a network. The MPI forum –a group of academic and industrial experts– designed and standardized this paradigm to be used in very different types of parallel computers. A wide set of functions are available for managing, sending and receiving messages.

Regarding the BP algorithm, looking at its behavior, a direct parallel approach is possible: 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 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. Obviously, the size of each group will directly depend on the particular characteristics of the problem as well as on the communication - computation ratio of each process.

For the MPI implementation, the well-known manager-worker scheme was chosen. The manager-process is responsible for loading the problem structure and the user-defined settings. 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 attended 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 belongs to the same worker or not. Figure 2 illustrates this manager-worker scheme assuming that three workers are running. The node distribution presented is according to the factor graph introduced in Figure 1. In our method this distribution has been done trying to equally distribute variable and factor nodes. However, a deeper study to weigh the incidence that different distributions could have on the algorithm's performance is regarded as a future work to be done.

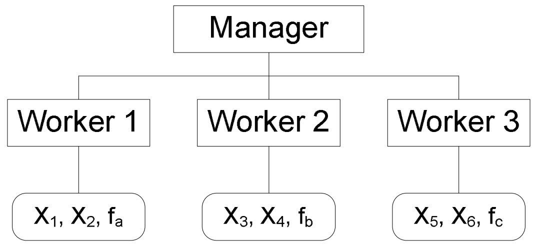

Figure 2: Manager-worker scheme. Node distribution is shown for the factor graph in Figure 1 using three workers.

Pseudo-code for the manager

```
Step 1.  Get the structure of the factor graph from file
Step 2.  Get the configuration from file
Step 3.  Send the structure of the FG to the workers
Step 4.  Send the configuration to the workers
Step 5.  numWorkersEnded = 0
Step 6.  While numWorkersEnded != numWorkers
             Wait for a worker to finish
             Store messages from the worker
             numWorkersEnded = numWorkersEnded + 1
         End While
Step 7.  Send stop order
Step 8.  Show results
Step 9.  Stop.
```

Figure 3: Pseudo-code for the manager.

Figures 3 and 4 describe the pseudo-code for the manager and workers.

5. EXPERIMENTS

In order to test the performance of our parallel approach, we have conducted a number of experiments using as a test bed a well known optimization function (checkerboard) and running the experiments using a cluster of computers.

The checkerboard function was firstly introduced in [2]. In this problem, a $s \times s$ grid is given. Each point of the grid can take a value 0 or 1. The goal of the problem is to create a checkerboard pattern of 0's and 1's on the grid. Each point with a value of 1 should be surrounded in all four basic directions by a value of 0, and vice versa. The evaluation counts the number of correct surrounding bits. The maximum value is $2s(s-1)$, and the problem dimension is $n = s^2$. If we consider the grid as a matrix $\boldsymbol{x} = [x_{ij}]_{i,j=1,\ldots,s}$ and interpret $\delta(a,b)$ as the Kronecker's delta function, the checkerboard function can be written as[1]:

$$F_{CheckerBoard}(\boldsymbol{x}) = 2s(s-1) - \sum_{i=1}^{s}\sum_{j=1}^{s-1}\delta(x_{ij}, x_{ij+1}) - \sum_{j=1}^{s}\sum_{i=1}^{s-1}\delta(x_{ij}, x_{i+1j}). \quad (7)$$

This problem is represented using factor graphs by adding factor nodes, horizontally and vertically, between each pair of points of the grid (variable nodes). The factor function takes value 0 when its arguments have the same value and 1 otherwise. A factor graph representation for the 3×3 checkerboard problem is shown in Figure 5.

[1]This is a slightly different formulation that the used in [2]

Pseudo-code for workers

```
Step 1.  Receive the structure of the factor graph
         from the manager
Step 2.  Receive the configuration from the manager
Step 3.  For each node $n$ in worker
             If startsSending
                 Look for a scheduling-rule
                 Calculate messages
                 Send messages according to the rule
             End If
         End For
Step 4.  While not allNodesFixed
             Wait for a message
             For each active node $n$ in worker
                 Look for a scheduling-rule
                 If ruleFound
                     Calculate messages
                     Check stopping criteria
                     Send messages according to the rule
                 End If
             End For
         End While
Step 5.  Send final messages to the manager
Step 6.  Wait for the stop order
Step 7.  Stop.
```

Figure 4: Pseudo-code for the workers.

Experiments were carried out in a cluster of computers with 4 nodes. Each node has two Intel Xeon processors (2.4GHz, hyper-threading disabled), 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 version 3.3.3 of GCC. Nodes are interconnected using a switched Gigabit Ethernet network.

Regarding the problem complexity, two different sizes were selected: 40 and 50 –which consequently requires a factor

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

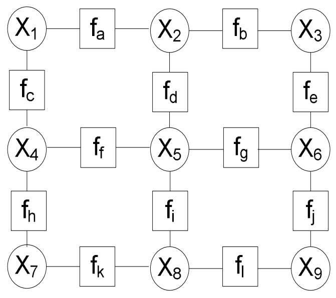

Figure 5: Factor graph representation for a 3×3 checkerboard problem.

graph structure with 4,720 and 7,400 nodes respectively. The rest of parameters were fixed as follows:

- The scheduling-policy was defined based on sets of nodes. Particularly, variable nodes start sending messages and the subsequent information-interchanging is completed for each node (either variable or factor) every time it receives a message from all its neighbors (these scheduling-rules are the same as those in Figure 1).

- 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.

- Graph is partitioned taking into account the number of MPI process available. The distribution will be completed trying to distribute in each MPI process a similar number of variable nodes and factor nodes.

It must be noted that in these experiments we do not intent to proof the suitability of the BP algorithm to solve this particular problem. These preliminary runs have been designed to study the execution time and scalability of the parallel approach.

Tables 2 and 3 show the execution time (in seconds) for the 40×40 and 50×50 problems respectively. Taking into account the nature of the BP algorithm, the parallel application has high communication requirements due to a constant message exchange until a stopping criterion is met. That is why we decided to run different experiments changing the number of MPI processes running in each node.

Looking at the results, it can be seen that our parallel proposal presents, in general, an acceptable scalability. In addition, varying the number of MPI processes executed in each CPU allows to notably reduce the execution time.

Since this is a preliminary approach, additional experiments should be conducted in order to properly tune the parallel application. However, some interesting conclusions can be obtained from our findings:

- Overcharging CPUs is useful only when non CPU intensive applications are executed. These results confirm our initial hunch, in the sense that communication plays an important role in this application. Thus, looking for communication bottlenecks will be essential to improve the general behavior of the application as well as its scalability.

- Workload distribution is really important when designing a parallel algorithm. Particularly, in our application, the way the graph is partitioned can give rise to non well balanced tasks (as seems to be the case).

- The available MPICH2 installation was configured with default options. When applying these, the communication between the MPI processes is performed though sockets, even when communicating processes are in a same machine. Obviously, this fact affects to all intra-node communications, and it is our aim to complete a deeper study using the different communication devices provided by MPICH2: only sockets, socket plus shared memory, and nemesis (a particular high-performance device).

- As mentioned when describing the characteristic of the cluster, hyper-threading was disabled. This technology provides thread-level parallelism on each processor, resulting in a more efficient use of processor resources, and a higher processing throughput. As observed in previous works [15], using this technology can be really helpful, helping to increase the performance of parallel implementations.

6. CONCLUSIONS AND FUTURE WORK

This paper introduces a parallel framework for a probabilistic inference algorithm. Among the different approaches available, we have focused on the belief propagation algorithm over factor graphs. BP is generally used to obtain marginal probabilities or the most probable state of the distribution. The algorithm is based on message-passing operations that are applied until a stopping condition is reached.

Before designing our parallel approach, BP algorithm was studied deeply trying to create a framework as flexible as possible. This solution allows the user, for example, to define the most suitable scheduling policy for a problem, to fix initial message values for a set of nodes or to decide the criterion that should be reached to stop the execution. This parallel framework has been implemented using the well-known message-passing paradigm MPI. The manager-worker scheme has been adopted, where manager loads initial information, sends it to the workers, and waits to gather the final results. On the other hand, workers manage a set of nodes (variables or factors), sending, calculating, and receiving messages according to the scheduling policies and parameters established initially.

Our experiments prove that our parallel proposal shows an acceptable scalability, being able to make a efficient use of up to 6 or 8 CPUs for the different problems defined.

Regarding the future research lines, we aim to extend this work to different scopes:

Test schedulings: once the user has designed the proper scheduling rules for the problem, it would be necessary to check if blocking situations can occur. That is, a node is waiting for messages from its neighbors, but these messages will never arrive because senders are also waiting for receiving other messages.

Optimum node distribution: taking into account that each MPI-worker executes the message-interchanging of a group of nodes (variables and/or factors), the way these nodes are distributed among the workers is really important and can condition the performance. It would be necessary to apply an optimization technique to properly distribute the nodes according to the structure of the factor graph, number of workers, cardinality of the variable nodes, and so on.

Performance analysis: We have presented some preliminary results using MPICH2's default configuration. In addition to this work, it could be interesting to extend the experiments completing an exhaustive performance analysis, taking into account the configurations available (communication devices via shared memory or nemesis).

Analysis of BP: Although some theoretical works there exist for BP algorithm, there is not a whole knowledge about its behavior. Therefore, the flexibility of

Table 2: Execution time (seconds) for the 40×40 problem, using up to 4 nodes (8 CPUs) and different number of MPI processes per node. Best results are presented in bold.

40×40							
# nodes (# CPUs)	# processes per node						
	2	4	8	10	12	14	18
1 (2)	354.05	132.64	43.65	38.69	46.61	**36.53**	39.34
2 (4)	120.76	50.11	25.01	**22.59**	22.76	23.56	24.46
3 (6)	63.45	27.89	20.16	**19.50**	21.94	24.47	24.81
4 (8)	41.57	22.55	**15.64**	16.74	19.35	20.10	23.80

Table 3: Execution time (seconds) for the 50×50 problem, using up to 4 nodes (8 CPUs) and different number of MPI processes per node. Best results are presented in bold.

50×50							
# nodes (# CPUs)	# processes per node						
	2	4	8	10	12	14	18
1 (2)	921.33	330.18	162.56	138.09	116.60	108.06	**83.03**
2 (4)	296.99	120.81	72.35	54.46	53.97	**48.98**	52.35
3 (6)	157.50	67.53	48.72	44.09	**43.75**	48.21	51.88
4 (8)	100.61	50.93	**36.05**	36.12	39.17	43.80	48.02

our tool to set different configurations can be useful to empirically study some properties of the algorithm.

EDAs: Finally, we also plan to include BP as a part of the EDA framework. Particularly, BP can be used to complete individual sampling.

7. REFERENCES

[1] Anonymous. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):159–416, 1994.

[2] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, Carnegie Mellon Report, CMU-CS-97-107, 1997.

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

[4] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.

[5] A. Braunstein and R. Zecchina. Survey and belief propagation on random k-sat. *Lecture Notes in Computer Science*, 2919:519–528, 2004.

[6] 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.

[7] 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.

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

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

[10] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

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

[12] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. Springer-Verlag, 2006.

[13] 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.

[14] 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.

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

[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, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 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. Rechenberger, and H. P. Schwefel, editors, *PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer, 1996.

[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] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.