

# Implementing Belief Propagation on P-Grid

Nicolas Bonvin, Damien Auroux and Grégoire Montavon

**Abstract**—Some applications require to solve a Bayesian network in a distributed fashion. It can lead to a large amount of messages exchanged between hosts if the variables of the Bayesian network are not allocated to good hosts. The technique we implement uses an algorithm that allocates variables to hosts so that the *tension* of the Bayesian network is minimized. We present and discuss our results.

## I. INTRODUCTION

A Bayesian network is a set of statistically dependent random variables. It is represented as a directed graph where nodes represent variables and edges are the dependence link between two variables. For example, if we have an independent random variable  $x$  with distribution  $p(x)$ , a random variable  $y$  with distribution  $p(y|x)$  and a random variable  $z$  with distribution  $p(z|y)$ , the corresponding Bayesian network will be  $x \rightarrow y \rightarrow z$ .

Belief propagation, also known as the sum-product algorithm, is an iterative algorithm for computing marginal probability distributions. It is described in algorithm 1. This algorithm converges for loop-free Bayesian networks. Convergence for Bayesian networks with loops is still an open question.

---

### Algorithm 1 Belief propagation

---

```

for  $t = 1$  to  $n$  do
   $V =$  set of leaves of the Bayesian network
  while  $V \neq \emptyset$  do
     $temp = \emptyset$ 
    for all  $v \in V$  do
       $W =$  set of neighbors of variable  $v$ 
       $v$  sends distribution  $p_v(\mathbf{x})$  to  $W$ 
      each  $w \in W$  computes distribution  $p_w(\mathbf{x})$ 
       $temp = temp + W$ 
    end for
     $V = V - temp$ 
  end while
end for

```

---

This algorithm has been extensively used in a

single-host environment. However, some applications require that the variables are not stored on a single machine. There are two reasons for that: (1) variables should not be stored on a single machine for privacy issues and (2) running belief propagation on a very large Bayesian network using a single machine does not scale. Therefore, we need an efficient way to run the belief propagation in a distributed fashion.

In algorithm 1, probability distributions of the variables must be advertised to other variables. In a distributed system where variables are stored on different hosts that are located at different physical places, these probability distributions must be sent over the network. For large and well-connected Bayesian networks, this could lead to a starvation of network resources.

A solution to reduce the network consumption of algorithm 1 is to reallocate variables to hosts in a way that (1) reduces the network consumption and (2) balances the load between hosts. We implement an algorithm for redistributing variables, run it on a set of machines and compare the obtained results with previous work.

## II. DESIGN

Each host of the network has an initial set of variables. Algorithm 2 reallocates variables to different hosts in a distributed fashion. A more complex algorithm has been proposed in [1] and tested on a simulator, but it is dependent on the structure of P-Grid, and we prefer implementing a simpler version of this algorithm as a starting point, making it also more suitable for comparison with future systems.

### A. Computing the tension

Let  $d(X, Y)$  be the distance between host  $X$  and host  $Y$ . We have  $d(X, Y) = d(Y, X)$  and  $d(X, X) = 0$ . The tension between two variables is the distance between the two hosts on which these variables are located. Tension is zero when the two variables are on the same host. The tension of the Bayesian network is the sum of the tensions of all connected pairs of

**Algorithm 2** Distributed tension minimization

---

```

V = set of local variables
while true do
  if size(V) > minLoad then
    N = neighbor hosts with load < maxLoad
    (v, n) = argmin(v,n) ∈ V × N ΔT(v, n)
    send v to n
    V = V - {v}
  end if
end while

```

---

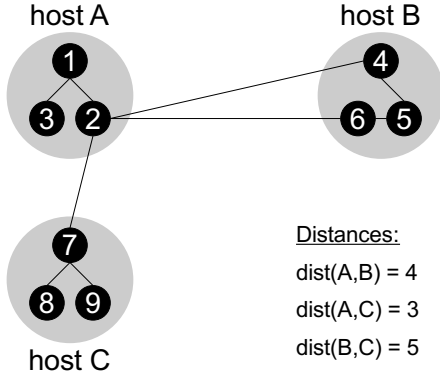


Fig. 1. Bayesian network distributed on several hosts

variables. The function  $\Delta T(v, n)$  is the difference of tension resulting from sending a variable  $v$  to host  $n$ . It can be computed as the tension of the Bayesian network after sending the variable minus the tension of the Bayesian network before sending the variable.

Suppose we have the network configuration presented in figure 5 and that we consider moving variable 2 from host  $A$  to host  $B$ . The difference of tension is:

$$\begin{aligned} \Delta T(2, B) &= (d(B, A) + d(B, C)) - \\ &\quad (d(A, B) + d(A, B) + d(A, C)) = -2 \end{aligned}$$

Since it is negative, it is worth moving variable 2 from  $A$  to  $B$ . But in practice, it is impossible to keep track of the location of all the variables. Therefore, we use an approximation: we consider only the variables that are located at the source and at the destination. host  $A$  and  $B$ . Therefore, the only distance that we need to know is  $d(A, B)$ . We obtain the following approximate difference of tension:

$$\Delta T(2, B) = d(B, A) - (d(A, B) + d(A, B)) = -4$$

The approximation of the difference of tension is  $-4$  which is slightly different from the the real difference of tension. The approximation is equal to the real difference of tension if hosts are equidistant.

*B. Push/pull strategies and load balancing*

There are two possible strategies for exchanging variables: (1) *push strategy*: hosts select among all possible actions (sending a variable  $v$  to neighbor  $n$ ) the one that gives the highest reduction of tension and perform the selected action (sending the variable). (2) *pull strategy*: hosts select among all possible actions (receiving a variable  $v$  from neighbor  $n$ ) the one that gives the highest reduction of tension and perform the selected action (requesting the variable).

We also need to design a mechanism to control the minimal and maximal load that hosts can have. Suppose we want the load of every host to be in the interval  $[l^-, l^+]$ . We describe the mechanisms for the two strategies we presented in the previous paragraph: (1) load balancing for *push strategy*: hosts send variables only if their load is greater than  $l^-$ . Variables are sent only to neighbors that have a load smaller than  $l^+$ . (2) load balancing for *pull strategy*: hosts request variables only if their load is smaller than  $l^+$ . Variables are requested only to neighbors that have a load greater than  $l^-$ .

Both strategies require asking neighbors for their load. Once hosts have information about their neighbors, they have to proceed to the exchange of the variable. If the host is using a push strategy, it simply needs to send a message to the chosen host with the chosen variable. On the other hand, if the host is using the pull strategy, it needs to perform a request to the chosen host for the chosen variable and then, the latter will send back the variable. Therefore, we prefer using the push strategy, since it requires one less message compared to the pull strategy.

## III. IMPLEMENTATION

Implementing this algorithm requires an overlay network and a DHT (distributed hash table). The overlay network is used to maintain for each host a set of neighbors to which variables can be sent. The DHT is used to maintain the average number of variables per host (the *average load* of the system). We implemented algorithm 2 on top of P-Grid which builds an overlay network and provides a DHT on top of it.

We adopt a push strategy for exchanging the variables. The load interval allowed for hosts is  $[0.5 \cdot avgLoad, 2 \cdot avgLoad]$ . This is controlled by the mechanisms we described in the design part.

Every 500 milliseconds, an attempt to send a variable is performed by every host. The neighbors are contacted and when information about their variables

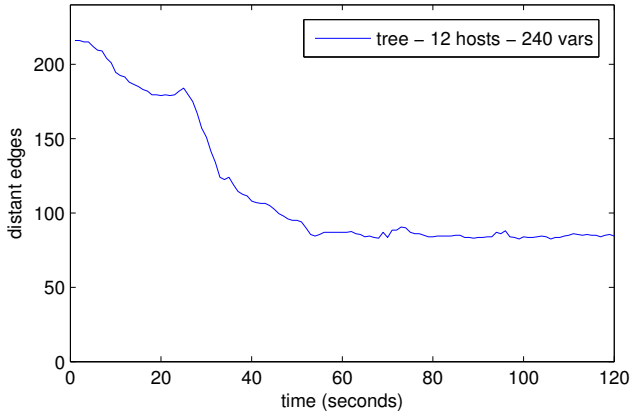


Fig. 2. Evolution of distant edges for a tree Bayesian network

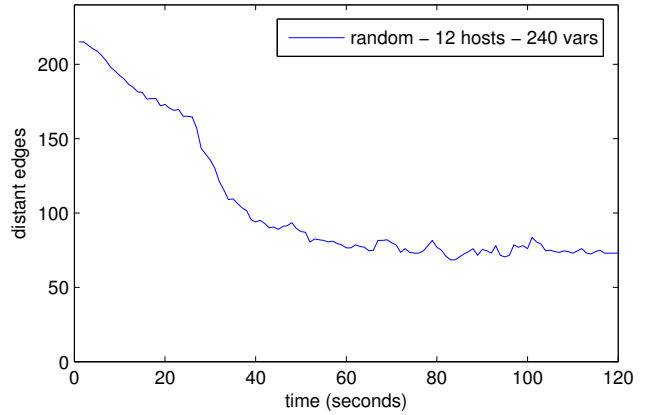


Fig. 3. Evolution of distant edges for a random Bayesian network

is collected, the best action to perform is chosen. If one of the neighbors fails to reply, it is discarded from the set of possible hosts to which variables can be sent. In the current implementation, actions are considered independently. An optimization could be to consider the combinatorial problem of finding an optimal consistent subset of actions and perform them all at the same time.

To implement the exchange of variables, we created our own P-Grid messages and used the routing facilities provided by P-Grid. Messages are serialized, encoded in base-64, and included in a compressed XML message.

We created Bayesian networks to test our implementation with three different topologies: binary tree-based, random and scale-free. Bayesian networks are exploded in a collection of stubs (variables with their relations) that are randomly distributed to the hosts.

#### IV. RESULTS AND ANALYSIS

We ran our program on a set of 12 machines with a Bayesian network of 240 variables. The structure of the Bayesian network is either a tree, a random graph or a scale-free graph.

We take a snapshot of the current allocation of variables every second and study the evolution over time. The two metrics we use to evaluate our system are the number of distant edges and the load distribution. We plot the results for the three different types of Bayesian networks we generated.

The algorithm takes around 60 seconds to converge. The first results we obtained had a much slower convergence time because of the presence of some waiting times for debugging purposes and stability issues.

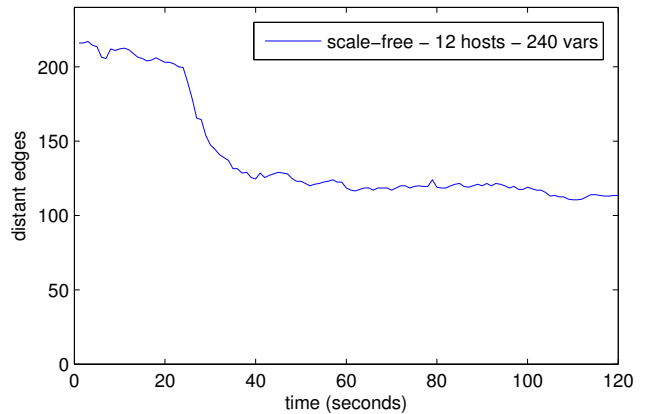


Fig. 4. Evolution of distant edges for a scale-free Bayesian network

Once the stability issues were solved, we reduced the waiting times to the maximum and obtained faster convergence.

We can observe two distinct phases in the evolution of distant edges over time. One where the number of distant edges decreases slowly and one where the number of distant edges decreases much faster to converge to a fixed value. This is explained by the fact that once a fraction of the hosts gets more populated, they tend to attract variables of other nodes as much as they are allowed to, which leads to a fast reduction, but this set of more populated nodes becomes significant only after some time.

We can also observe that the reduction of the distant edges is not monotonic. The explanation is that hosts only have a partial knowledge of the distribution of variables on the hosts. This is also due to the fact that our measuring system does not take into account variables that are being sent from one node to

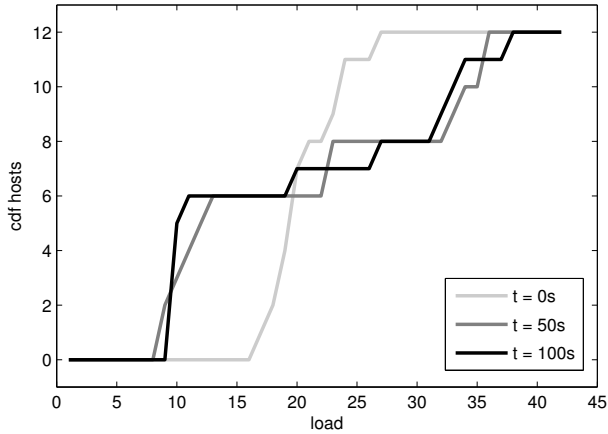


Fig. 5. Distribution of the load between the hosts for different times in the case of a scale-free graph

another, but it represents on average only 3% of the total number of variables.

## V. FUTURE WORK

Due to the fixed neighbors, search of the optimal allocation of variables can be stuck in a local optima. The current implementation of the algorithm sometimes performs bad actions to get out of these local optima. By using a random subset of neighbors that is recomputed regularly (the fidget list in P-Grid), the geometry of the space allocation of variables is constantly modified, thus, removing the problem of local optima. The convergence time might therefore be improved.

## VI. CONCLUSION

We showed that it is possible to solve Bayesian network in a distributed fashion in a reasonable time. We implemented a novel algorithm to decrease the number of messages that must be sent over the network to solve the Bayesian network. Our algorithm balances the load between the hosts in the system, has a satisfactory convergence time and works for various topologies of Bayesian networks.

## REFERENCES

- [1] Roman Schmidt and Karl Aberer. Efficient peer-to-peer belief propagation. In *Cooperative Information System conference (CoopIS)*, Montpellier, France, 01-03.11.06, 2006.