Efficient Peer-to-Peer Belief Propagation *

Roman Schmidt and Karl Aberer

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Abstract. In this paper, we will present an efficient approach for distributed inference. We use belief propagation's message-passing algorithm on top of a DHT storing a Bayesian network. Nodes in the DHT run a variant of the spring relaxation algorithm to redistribute the Bayesian network among them. Thereafter correlated data is stored close to each other reducing the message cost for inference. We simulated our approach in Matlab and show the message reduction and the achieved load balance for random, tree-shaped, and scale-free Bayesian networks of different sizes.
As possible application, we envision a distributed software knowledge base maintaining encountered software bugs under users' system configurations together with possible solutions for other users having similar problems. Users would not only be able to repair their system but also to foresee possible problems if they would install software updates or new applications.

1 Introduction

Peer-to-peer systems currently share local information by pairwise interactions in a cooperative way. The most popular application to date is file-sharing such as Gnutella and BitTorrent providing search functionality respectively efficient content distribution. Shared data is usually file-based and files are not correlated with each other, i.e., it is sufficient to find a desired file and to be able to download it. More sophisticated applications rely on correlated data probably spread out among several nodes and downloading each part for local processing can be too expensive. Another solution is to perform distributed inference directly in the network so that data remains at providing nodes and only small messages to process the inference are exchanged.

Distributed inference is already applied for various applications in other networks such as sensor networks [1] where network limitations are probably more

* The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project NEPOMUK No FP6-027705.

obvious. We envision applications on top of a peer-to-peer system relying on knowledge provided by nodes and used to solve inference problems. A practical example is a distributed knowledge base for software bugs observed by users. Currently, bug reports are submitted on user acceptance to a central knowledge base for further processing. We assume that most of the bug reports are not submitted because users are afraid to reveal their identity. In our distributed scenario, the reports are inserted into a peer-to-peer system concealing the users identity. Distributed inference is then used to suggest solutions for occurred errors as known from central knowledge bases.

Belief propagation [2] enables distributed inference by a simple message-passing algorithm between nodes in a Bayesian network modeling correlations between variables. A node can represent any kind of variable, be it an observed measurement, a parameter, a latent variable, or a hypothesis. Belief propagation was first successfully applied in the domain of error correcting codes (Turbo Codes [3]), speech recognition, image processing and medical diagnosis. Recently, it was used in peer-to-peer systems in the context of content distribution [4] and in sensor networks [5]. The simplicity of the message-passing algorithm holds the risk of being not scalable towards large-scale networks because many small messages have to be sent between nodes. Approaches to reduce communication costs such as Generalized Belief Propagation [6] cluster nodes and build a hierarchy based on common variables of clusters. The message reduction comes with the drawback that the size of sent messages increases exponentially ($number\ of\ states^{nodes\ in\ the\ cluster}$) because the exchanged messages now contain the joint probabilities of all nodes and states in the cluster. What remains unsolved is how nodes are clustered in a distributed way requiring no global knowledge and coordination so that the communication costs are minimized.

In this paper, we will present a decentralized algorithm to cluster variables at nodes to reduce the number of physical messages sent over the network by not increasing message sizes. The overall number of messages to run Pearl's belief propagation algorithm remains the same but most of them are sent node-internally which does not induce any bandwidth nor latency costs. Our clustering algorithm is based on the spring relaxation technique used for example in peer-to-peer systems for virtual coordinate systems [7] and for path optimization in stream-based overlays [8] to find minimal energy configurations. In our case, we try to find the minimal configuration for variables stored on nodes organized in an P-Grid [9] overlay network. P-Grid provides us a distributed index of the Bayesian network and efficient lookup mechanisms.

In the following, we will first explain briefly the background and the basis of our approach, belief propagation in Section 2 and P-Grid in Section 3. Afterwards, we will present peer-to-peer blief propagation in Section 4 before we evaluate our approach in Section 5. The paper discusses related work in Section 6 and future work in Section 7. We conclude in Section 8.

2 Belief Propagation

Pearl's belief propagation [2], also known as the sum-product algorithm, is an iterative algorithm for computing marginal probabilities, "beliefs" about possible diagnoses, of nodes on a probabilistic graphical model such as Bayesian networks. A Bayesian network is an directed acyclic graph of nodes representing variables and edges representing dependence relations among the variables. If there is an edge from node A to node B, then node B's state depends on node A's state. This is specified by a conditional probability distribution for node B, conditioned on the state of node A. A Bayesian network is a representation of the joint distribution over all the variables represented by nodes in the graph. We assume that the joint probability distribution factors into a product of terms involving node pairs and single nodes. These factors are called edge potentials $\psi_{ij}(x_i, x_j)$ and local potentials $\phi_i(x_i)$. Evidence nodes are nodes with a known value. A node can represent any kind of variable, e.g., an observed measurement, a parameter, a latent variable, or a hypothesis. For example, consider the simple Bayesian network in Figure 1 consisting of 3 variables OS1, Driver1 and App1. The dependencies are as follows: if the hardware driver Driver1 is installed on the operating system OS1, the application App1 is likely to run smoothly with 90% probability. If the driver is missing, the application runs only to 40% and if OS1 is not installed, then the application does not run at all independent of the driver. If it is known that OS1 is installed, then its probability would be set to 1 and the probabilities for App1 to run would only depend on Driver1 thereafter.

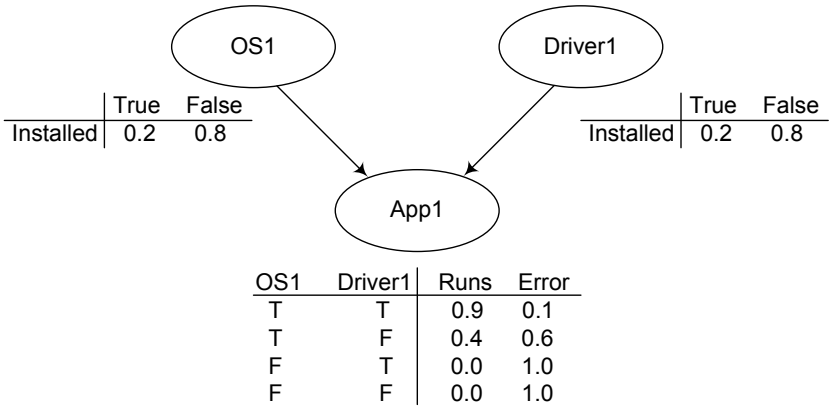


Fig. 1. Bayesian network example

The belief propagation algorithm is provably efficient on trees and experiments demonstrate its applicability to arbitrary network topologies using loopy belief propagation for loopy networks [10], which we will present in the following.

The algorithm is currently used with success in numerous applications including low-density parity-check codes, turbo codes, free energy approximation, and computer vision.

2.1 The Message Passing Algorithm

The algorithm passes messages across the edges in the graphical model, i.e., in each iteration, a node sends a message to an adjacent node if it has received messages from all of its other adjacent nodes at the previous iteration. In the first iteration, nodes send an initial message, usually set to 1, to all adjacent nodes. In subsequent iterations, messages passed from node x_i to node x_j are updated using the following rule:

$$m_{ij}(x_j) = \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \neq j} m_{ki}(x_i)$$

where $\phi_i(x_i)$ are the local potentials of node x_i and $\psi_{ij}(x_i, x_j)$ are the edge potentials. The product of messages excludes the message received in the previous iteration from node j, the node we are passing the message to. The messages $m_{ij}(x_j)$ and the local potentials $\phi_i(x_i)$ are vectors whose length corresponds to the number of states a node x_i can be in. The edge potentials $\psi_{ij}(x_i, x_j)$ are N x M matrices where N is the number of states node x_j can be in and M is the number of states for node x_i. Therefore, the message size of the belief propagation algorithm grows exponentially with the number of states of nodes.

Finally, the marginal probabilities of nodes, called the beliefs, can be computed by multiplying all received messages by the local potentials:

$$b_i(x_i) = \alpha\phi_i(x_i) \prod_{k} m_{ki}(x_i)$$

The beliefs are normalized by α to avoid numerical underflow. The algorithm converges if none of the beliefs in successive iterations changes by more than a small threshold. For singly connected graphs, it is proven [2] that beliefs at nodes converge to the marginal probability at that node, which is:

$$b_i(x_i) = \alpha \sum_{x_j / x_i} p(x) = p_i(x_i)$$

In networks with loops, evidence is counted multiple times. As all evidence is double counted in equal amounts, Pearl's belief propagation also provides good approximations of the marginal probabilities in loopy networks.

3 The P-Grid Overlay

The approach presented in this paper uses the P-Grid [9] distributed hash table (DHT). We assume that the reader is familiar with the general concepts of DHTs and will thus only address the specific and relevant properties of P-Grid.

In P-Grid peers refer to a common underlying binary trie structure to organize their routing tables. Data keys are computed using an order-preserving hash function to generate keys. Without constraining general applicability binary keys are used in P-Grid. Each peer constructs its routing table such that it holds peers with exponentially increasing distance in the key space from its own position. This technique basically builds a small-world graph [11], which enables search in $O(\log N)$ steps. Each peer $p \in P$ is associated with a leaf of the binary trie, i.e., a key space partition, which corresponds to a binary string $\pi(p) \in \Pi$ called the peer's *path*. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the trie the peer has references to some other peers that do not pertain to the peer's subtrie at that level which enables the implementation of prefix routing.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ $key(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing, and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain multiple references $\sigma(p)$ to peers with the same path (data replication).

Figure 2 shows a simple example of a P-Grid tree consisting of 6 peers responsible for 4 partitions, e.g., peer F's path is 00 leading to two entries in its routing table: peer E with path 11 at the first level and peer B with path 01 at the second level. Further, peer F is responsible for all data with key prefix 00. A search initiated at peer F for key 100 would first be forwarded to peer E because it is the only entry in F's routing table at level 1*. As peer E is responsible for 11 and not for the key 100, peer E further forwards the query to peer D, which can finally answer the query.

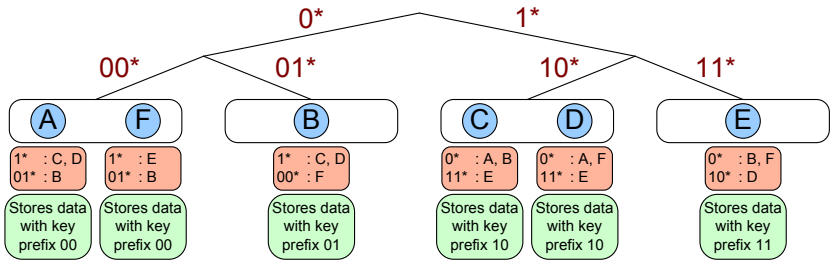


Fig. 2. P-Grid overlay network

4 Peer-to-Peer Belief Propagation

So far, we presented two independent approaches, on the one hand a distributed inference algorithm based on a simple message-passing algorithm and on the other hand an overlay system to store and retrieve data. At first sight, those two systems have not much in common but we will show in this section that both can benefit from each other. Our focus lies thereby on P-Grids ability to improve the scalability of belief propagation as the application of belief propagation to improve P-Grid's load balancing is already shown in [12].

4.1 Distributed Knowledge Base Scenario

To motivate our idea of providing a large-scale peer-to-peer inference system, we will describe in this section our distributed knowledge base scenario for software dependencies. Examples for centralized knowledge bases are the one from Microsoft [13] and Mozilla [14] providing comprehensive information about their products and support for encountered bugs. Both use a bug report and tracking system, for example BugZilla [15] from Mozilla, to collect bug reports from users with their permission. In our opinion and from our personal experience, we assume that most users do not permit submitting bug reports to a central authority such as Microsoft or Mozilla because they do not want to reveal their identity and system configuration. A distributed solution would allow users to conceal their identity because bug reports are inserted and stored at various nodes of the peer-to-peer system making it more difficult to track user identities. Therefore, we hope to be able to collect more reports from users with our decentralized solution leading to a more comprehensive knowledge base.

A knowledge base stores data together with their dependencies usually for the purpose of having automated deductive reasoning applied to them. Belief networks are one way to define those dependencies and belief propagation is an appropriate probabilistic reasoning method. In our scenario, any kind of software such as operating systems, device drivers and applications are nodes in our Bayesian network and their dependencies and distributions are learned from bug reports. A simplified bug report could look like:

```
IF
    OS = ['Linux', '2.4.12.18'] AND
    PACKAGE = ['MySQL-server', '4.0.20-0'] AND
    PACKAGE = ['MyODBC', '2.50.39-18.1']
    ...
THEN
    ERROR = 'MySQL'
END
```

A bug report starts with the used operating system, in this case with Linux and the kernel version 2.4.12.18, followed by a list of installed packages on the system. The second part consists of the affected application (MySQL) causing the error. The bug report enables us already to learn that the given system

configuration leads to problems for the MySQL application. Therefore, each bug report allows us already to create a small dependency graph, i.e., a Bayesian network, but we are still not able to identify responsible packages causing the problem. Therefore, we need a larger number of bug reports with probably varying system configurations to identify the strength of package dependencies, e.g., if a version of a package always occurs in a bug report for an application, it is very likely to cause the problem. A solution proposal for our example bug could be to install a different version of MyODBC as this version is listed in many bug reports for the MySQL application.

4.2 The Inference Architecture

Our idea of providing a generic distributed inference system is based on two fundamental design decisions: (i) no central coordination of the variables in the system and their dependencies; (ii) no global knowledge and only pair-wise interactions between nodes. Both requirements are satisfied by the P-Grid overlay infrastructure and Bayesian networks and belief propagation. P-Grid is first used to maintain the Bayesian network by indexing all variables in the system and all dependencies between them. In our scenario, variables would be software components with their version number, and their dependencies would be derived from bug reports at their insertion. At this stage, nodes are able to derive small dependency graphs from the bug reports they store and all the variables they maintain locally. Those dependencies are already represented by Bayesian network and have now be connected with each other. Learning a Bayesian network structure and probabilities from distributed data is studied in various papers [16–18]. The bug reports itself are also stored in P-Grid together with solutions for bugs provided by users. Therefore, users have the possibility to help each other with solutions and if no solution exists, inference can help to restrict the cause of error.

So far, we have a system storing a Bayesian network derived from bug reports. Belief propagation requires multiple message-passing iterations between all nodes of the Bayesian network which are currently spread over physical P-Grid nodes. On a global scale, this can lead to scalability problems for our system because messages would be sent around the globe multiple times. To tackle this problem, we uncouple variable values, the local potentials, from the P-Grid index and allow them to be stored at different physical P-Grid nodes to improve the efficiency of belief propagation. The current location of a variable's local potential is stored with the variable's index entry. The problem remains how those local potentials are stored close to each other, in the best case even on the same physical P-Grid node, without central coordination and knowledge. Our proposed solution is based on the spring relaxation technique and presented in more details in the following section.

4.3 The Relaxation Algorithm

In this section we describe the developed relaxation algorithm based on the spring relaxation technique. In our case, Bayesian variables are connected by springs and the Bayesian network forms a spring network which has to be relaxed, i.e., the network has to be in a state requiring least possible energy. The energy a spring requires is directly proportional to the distance between the two P-Grid nodes the Bayesian variables are stored at. The spring between two variables remaining at the same node requires no energy. Therefore, the optimal solution of the spring relaxation algorithm would be to place all variables at one node. This is of course not desirable because peer-to-peer systems are based on the idea of load sharing which is in contradiction with the optimal solution mentioned before. Thus, the spring relaxation algorithm also has to consider load balancing of variables among participating nodes. P-Grid provides already heuristic statistics about the current load of each level of the trie represented by a peer's routing table. These statistics are required by P-Grid itself to provide load-balancing of stored index information and are used in the following for our approach too. The statistics are based on periodic interactions with random peers of the routing table to sample the current load distribution. The periodic sampling enables peers to estimate the current load of a routing table level and the global average load.

The developed algorithm used to relax the Bayesian network is shown in Algorithm 1. The algorithm is executed by each node iteratively till no improvement is achieved anymore or a maximum number of iterations is reached. The following list provides an overview of the used variables in the algorithm:

- **localVars**: list of variables the local node maintains
- **avgLoad**: local estimate of the global average load
- **currentLoad**: the current load of the local node
- **routingTable**: the routing table of the local node
- **routingTable.levels**: the number of levels in the local routing table
- **candidate(j).tension(i)**: the tension at level i for candidate variable j
- **candidate(j).tension**: all tensions at all levels for candidate variable j

First, in line 1 to 4, each node checks if it has "free" variables it can move to other nodes or not. Currently, nodes are allowed to move variables as long as they have more than $avgLoad/2$ variables. P-Grid obtains an estimate for the current average load in the system but the accuracy of this estimate is not crucial for the algorithm. In line 5, nodes determine those local variables which have a tension to other nodes remaining at the same level of the local routing table leading to one tension at one level. Ideally, variables have a tension to only one node and not to different nodes at the same level. If the local node can move variables and it found such unidirectional variables, it moves them directly to the corresponding level or node (line 6 to 10). Moving a variable always requires only one message between the two involved peers.

A node can try to balance the load in the system if it maintains above average many variables. It therefore uses all non-unidirectional variables, i.e., variables

Algorithm 1 The spring relaxation algorithm

```
 1: freeVars = length(localVars) − avgLoad/2;
 2: if (freeVars <= 0) then
 3:    return;
 4: end if
 5: undirVars = variables having a tension only at one level;
 6: while ((freeVars > 0) AND (length(unidirVars) > 0)) do
 7:    move variable to a peer from the level with the tension;
 8:    removeFirst(unidirVars);
 9:    freeVars = freeVars − 1;
10: end while
11:
12: multidirVars = variables having tensions to multiple levels;
13: while ((currentLoad > avgLoad) AND (length(multidirVars) > 0)) do
14:    for i = routingTable.levels to 1 do
15:       if (level i is underpopulated) then
16:          candidates = variables having a tension at level i;
17:          for j = 1 to length(candidates) do
18:             if (candidate(j).tension(i) >= max(candidate(j).tension)) then
19:                move variable to a peer from level i;
20:                remove(multidirVars, candidate(j));
21:                currentLoad = currentLoad − 1;
22:                if (currentLoad <= avgLoad) then
23:                   break;
24:                end if
25:             end if
26:          end for
27:       end if
28:    end for
29: end while
```

which have tensions at multiple levels (line 12 and 13). Next, the node tries to balance each level of its routing table, starting with the highest level, i.e., its closest neighbors (line 14). Starting with the closest neighbors allows nodes to balance load first locally before they try to balance load on peers further away from them, i.e., on peers stored in lower levels. If a level is underpopulated (line 15), i.e., a level maintains below average many variables, then the node first selects candidate variables out of its local variables (line 16). Candidates are all variables which have a tension at the current level. Next, starting from line 17, the node checks if the tension at the current level for the candidate variable is the strongest tension the variable has considering all levels. This ensures that variables are moved to levels with their strongest tension. This process continues as long as candidates are available and the node has enough variables to move.

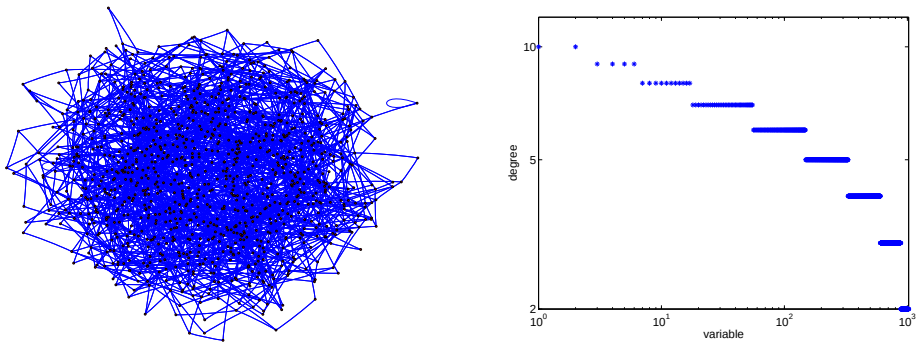
5 Evaluation

The algorithm presented in the previous section was implemented in Matlab and evaluated with diverse networks. We present results for random networks, binary trees and scale-free networks with up to 2048 variables in the Baysian network and 512 nodes in the P-Grid network. Considering our scenario we have in mind for our system, tree-based belief networks and scale-free networks are the most realistic network topologies. The network size and the number of variables is difficult to estimate but the evaluation shows that our approach scales well even though no proof can be given so far. All experiments were repeated 10 times and the figures show the average of those 10 repetitions with their standard deviation. Each time a new belief network was created and variables were assigned randomly to nodes.

5.1 Network Topologies

We briefly describe some properties of the network topologies we used for our evaluation. The networks were visualized with the Pajek tool [19] using the 2D Fruchterman Reingold layout for random networks and the Kamada-Kawai layout for the others. Additionally, we show the node degree distribution by sorting nodes according to their node degree and plotting their degree in log-log scale.

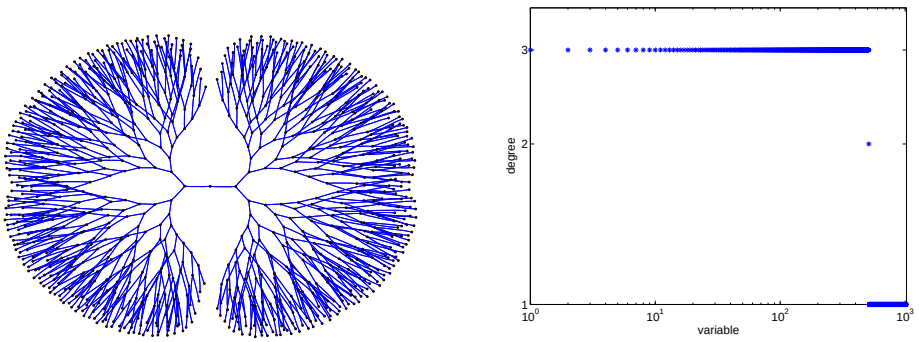
Random Networks We constructed random networks by adding for each node degree/2 edges to other nodes with equal probability to reach the desired average node degree. Figure 3 shows a network of 1024 nodes with an average node degree of 4, nodes have between 2 and 10 edges. The degree distribution indicates that most of the nodes have a degree around the average.



(a) Network visualization (b) Node degree distribution

Fig. 3. A random network: 1024 nodes with average node degree 4

Binary Trees The second used topology is a binary tree with each node having exactly two children excluding leaf nodes. Each node has exactly one parent excluding the root of the tree. Therefore, the node degree varies between 1 and 3 with an average around 2. Figure 4 shows a binary tree with 1023 nodes. The degree distribution shows the leave nodes (half of the nodes) at the bottom with 1 edge, the root with 2 edges in the middle and the intermediate nodes with 3 edges at the top.



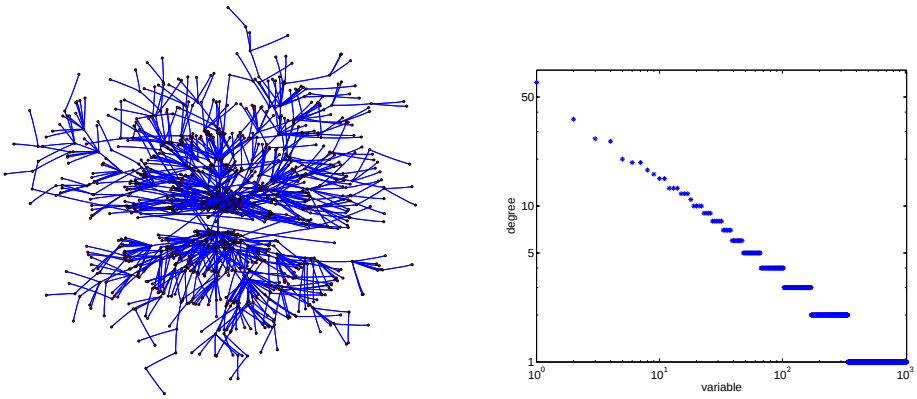
(a) Network visualization (b) Node degree distribution

Fig. 4. A binary tree: 1023 nodes

Scale-Free Networks The last used network topology is a scale-free network with the property that the number of links k originating from a given node exhibits a power law distribution $\mathbb{P}(k) \sim k^{-gamma}$. The network is constructed by progressively adding nodes to an existing network and introducing links to existing nodes with preferential attachment so that the probability of linking to a given node i is proportional to the number of existing links k_i that that node has, i.e.,

$$\mathbb{P}(\textit{linking to node } i) \sim \frac{k_i}{\sum_j k_j}$$

Scale-free networks occur in many areas of science and engineering, e.g., including the topology of web pages (where the nodes are individual web pages and the links are hyper-links), and are therefore a good model for our scenario. Figure 5 presents a scale-free network on the left side with highly connected nodes in the center and loosely connected nodes at the periphery. The node degree varies between 1 and 62 with an average around 4. The node degree distribution follows a power-law distribution.



(a) Network visualization (b) Node degree distribution

Fig. 5. A scale-free network: 1024 nodes with average node degree 4.

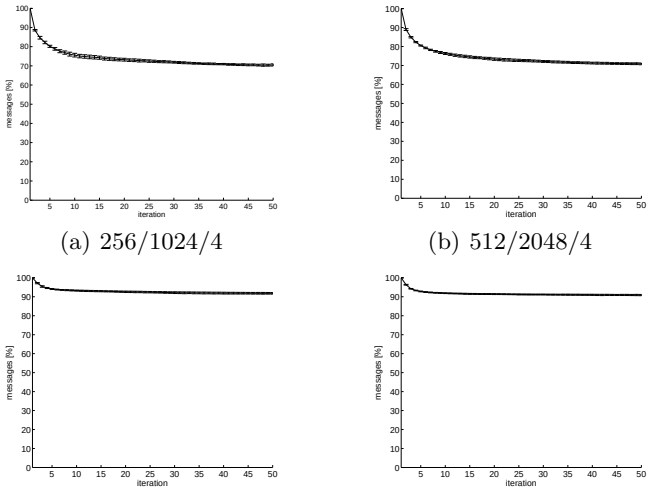
5.2 Message reduction

The most interesting evaluation criterion is of course the message reduction achieved by redistributing the variables close to each other in the P-Grid network. Figures 6 – 8 present the results obtained for the three network topologies. The plots show the achieved message reduction after each iteration of the spring relaxation algorithm by relating the number of required messages to run one iteration of the belief propagation algorithm. At the beginning, 100% of the messages are required, while after each iteration of the spring relaxation algorithm, less messages are required. The message reduction is given with the standard deviation of 10 repeated simulations for each setup.

Figure 6 shows that the algorithm does not perform well for any evaluated random network as expected. The random correlations of variables in these networks makes it difficult for the spring relaxation algorithm to cluster variables close to each other to reduce the message effort. The average node degree seems to have the strongest influence on the achieved message reduction which is not larger than 25%. As random networks are not considered as the most realistic model for our use case, this result is tolerable in our opinion. For binary trees, see Figure 7, the relaxation algorithm is already able to reduce the number of required messages to around 35% of the initially required number before running the relaxation algorithm. The obtained results seem to be independent of the number of nodes and variables. Finally, we observe similar results for the scale-networks as shown in Figure 8. The relaxation algorithm is able to reduce the message cost by about 75% independent of the number of nodes in the P-Grid network and the number of variables in the Bayesian network.

The standard deviation is small for all network topologies and network sizes which is an indicator that the algorithm scales well. In all experiments, the algorithm was iterated 50 times but the main reduction is achieved in the first

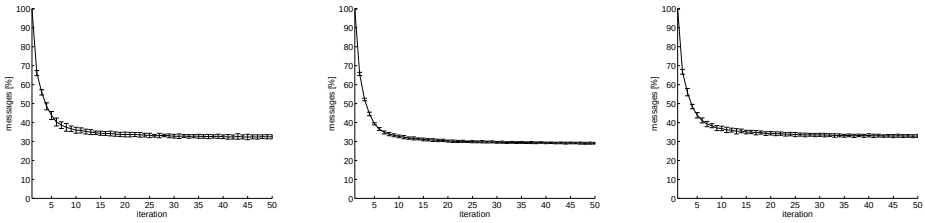
10 iterations. Again, this seems to be independent of the number of nodes and number of variables in the networks.



(a) 256/1024/4 (b) 512/2048/4

(c) 256/2048/8 (d) 512/2048/8

Fig. 6. Message reduction for random networks with different numbers of nodes/variables/degree.

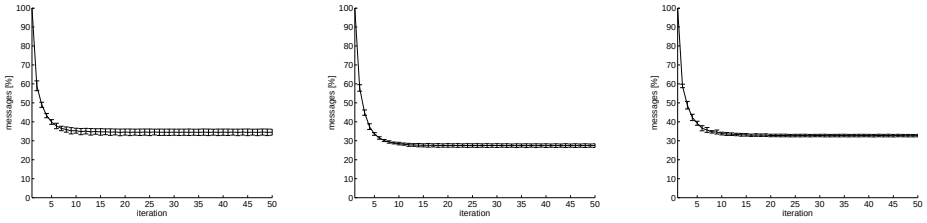


(a) 256/1024 (b) 256/2048 (c) 512/2048

Fig. 7. Message reduction for binary tree-based networks with different numbers of nodes/variables.

5.3 Load-balancing

Apart from the reduction of required messages for the message-passing algorithm, it is important that the load of variables is balanced among the participating nodes. Figures 9 – 11 present the corresponding results obtained again



(a) 256/1024 (b) 256/2048 (c) 512/2048

Fig. 8. Message reduction for scale-free networks with different numbers of nodes/variables.

for random networks, binary trees and scale-free networks. All figures show the average variable load which remains constant over all iterations as the number of variables and nodes does not change. The standard deviation indicates the load balance in the system. Additionally, the maximum load of nodes is given by the dotted line.

Whereas the relaxation algorithm did not perform well for random networks to reduce the number of required messages, it was more successful to balance the load among the nodes, as shown in Figure 9. The standard deviation is decreasing for all network sizes as well as the maximum number of variables per node (dotted line). Similar results were obtained for the binary tree-based networks (see Figure 10). Figure 11 shows that scale-free networks cause a slight increase of unbalance and a large increase in the maximum load for nodes. We think this is due to the fact that 1 or 2 nodes usually have very high degrees and therefore cause an overload at the P-Grid node they are currently maintained. Our relaxation algorithm is currently not able to handle this problem and we leave this as future work. A simple solution could be to allow nodes to decline maintaining further nodes by introducing an upper bound for the load.

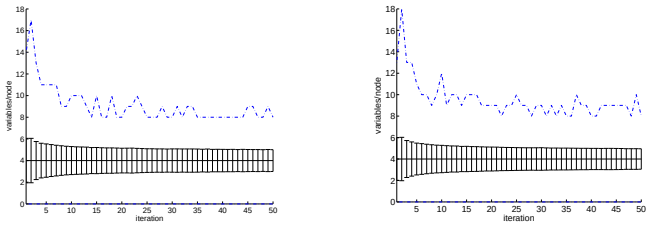
5.4 Discussion

The results obtained from the first evaluations look very promising. One way to probably further reduce the number of messages apart from the relaxation algorithm is to combine messages to one large message if local variables have a relation with variables at the same remote node. It is usually more efficient to send less large messages than more small messages in a peer-to-peer system.

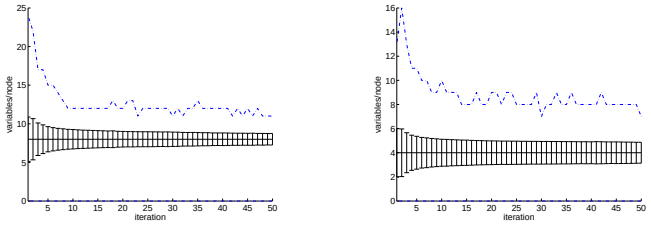
6 Related Work

6.1 Belief Propagation

Generalized Belief Propagation [6] reduces the number of messages by clustering correlated variables together and sending only one message between those



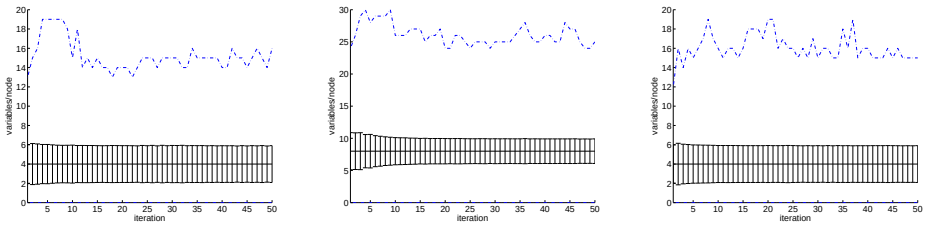
(a) 256/1024/4 (b) 512/2048/4



(c) 256/2048/8 (d) 512/2048/8

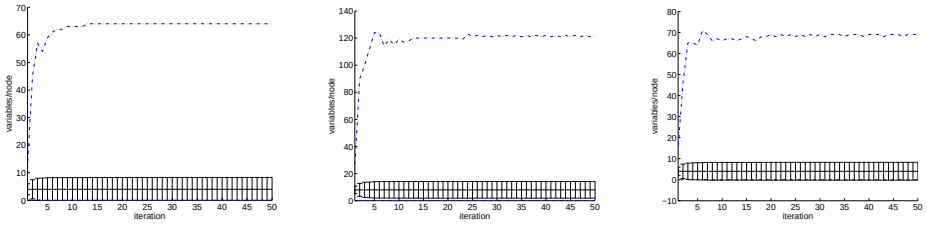
Fig. 9. Variables per node for random networks with different numbers of nodes/variables/degree.

clusters. This approach has three drawbacks: (i) the message sizes increase exponentially ($number\ of\ states^{nodes\ in\ the\ cluster}$) because the exchanged messages now contain the joint probabilities of all nodes and states in the cluster; (ii) the complexity of processing the messages and beliefs at nodes also increases considerable with increasing number of nodes in a cluster; (iii) it is not obvious for us how clusters are formed in a distributed way without central coordination and knowledge which is essential in peer-to-peer systems. Though Generalized Belief Propagation provides more accurate beliefs than Pearl's belief propagation, it is currently not applicable for large-scale networks.



(a) 256/1023 (b) 256/2047 (c) 512/2047

Fig. 10. Variables per node for binary tree-based networks with different numbers of nodes/variables.



(a) 256/1024 (b) 256/2048 (c) 512/2048

Fig. 11. Variables per node for scale-free networks with different numbers of nodes/-variables.

Reference [1] presents an inference architecture for sensor networks based on message-passing on a junction tree. For this approach, a distributed algorithm is first used to form a spanning tree of nodes which is used later to construct the junction tree for inference. Junction trees group variables into cliques and their size determines the computation costs at nodes whereas the separator size between cliques determines the communication costs. The approach was evaluated with 54 sensor motes in a local experiment showing spanning tree optimizations and the communication costs of the junction tree. Inference on junction trees is exact and always results in the exact marginals at the cost of requiring building a tree with larger messages and higher computation costs. Belief propagation only provides approximate inference on lower overheads.

6.2 Spring Relaxation

Spring relaxation is used in various domains and we will only present two examples for peer-to-peer systems. Vivaldi [7] is a decentralized network coordinate system using a spring-mass model to position nodes in a virtual coordinate system according to their latencies. Nodes run the distributed spring relaxation algorithm as soon as a new latency measurement was performed to reduce the distance error between nodes. An application of Vivaldi is described in [8] to optimize the path in stream-based overlay networks. Services are placed on nodes close to each other in the virtual latency space.

7 Future Work

An open issue is the introduction of a stop criteria for the relaxation algorithm so that nodes detect that further iterations will not reduce the number of messages noticeably any more. This is crucial because a constant maximum number of iterations may influence the scalability of our approach. Further, the algorithm currently runs absolutely synchronized at all nodes. As this is not realistic in peer-to-peer systems, the influence of an asynchronous execution has to be investigated. We plan to implement our algorithm in P-Grid to evaluate it on PlanetLab, a global-scale testbed with real network characteristics.

8 Conclusions

We presented a relaxation algorithm making large-scale distributed inference possible in peer-to-peer systems. Our approach is based on belief propagation's simple message-passing algorithm to perform inference and the P-Grid overlay network to store and maintain the required Bayesian network. Nodes of the Bayesian network are redistributed among P-Grid nodes to cluster correlated nodes together to minimize the required message costs for inference. Our purely distributed approach does not require any central coordination nor global knowledge. Matlab evaluations show promising results with message reductions up to 70% for various network topologies and network sizes.

References

1. Paskin, M.A., Guestrin, C.E., McFadden, J.: A robust architecture for distributed inference in sensor networks (2005)
2. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, CA, USA (1988)
3. Berrou, C., Glavieux, A., Thitimajshima, P.: Near shannon limit error-correcting codes and decoding: Turbo codes. In: Proceedings of the IEEE International Communications Conference. (1993)
4. Bickson, D., Malkhi, D., Rabinowitz, D.: Efficient large scale content distribution. In: Proceedings of the Workshop on Distributed Data and Structures (WDAS), Lausanne, Switzerland (2004)
5. Ihler, A.T., John W. Fisher, I., Moses, R.L., Willsky, A.S.: Nonparametric belief propagation for self-calibration in sensor networks. In: Proceedings of the Third international symposium on Information processing in sensor networks, New York, NY, USA, ACM Press (2004) 225–233
6. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Generalized belief propagation. In: Advances in Neural Information Processing Systems (NIPS). Volume 13. MIT Press (2000) 689–695
7. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: Proceedings of ACM SIGCOMM. (2004)
8. Pietzuch, P., Shneidman, J., Welsh, M., Seltzer, M., Roussopoulos, M.: Path optimization in stream-based overlay networks. Technical Report TR26-04, Harvard University, Cambridge, Massachusetts (2004)
9. Aberer, K.: P-grid: A self-organizing access structure for p2p information systems. In: Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS), London, UK, Springer-Verlag (2001) 179–194
10. Weiss, Y.: Correctness of local probability propagation in graphical models with loops. Neural Computation **12**(1) (2000) 1–41
11. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: ACM STOC. (2000)
12. Bickson, D., Dolev, D., Weiss, Y., Aberer, K., Hauswirth, M.: Indexing data-oriented overlay networks using belief propagation. In: Proceedings of the Workshop on Distributed Data and Structures (WDAS), Santa Clara, CA, USA (2006)
13. Corporation, M.: Microsoft help and support (2006) http://support.microsoft.com/.

14. Organization, T.M.: Mozillazine knowledge base (2006) http://kb.mozillazine.org/.
15. Organization, T.M.: Bugzilla (2006) http://www.bugzilla.org/.
16. Yamanishi, K.: Distributed cooperative bayesian learning strategies. In: COLT '97: Proceedings of the tenth annual conference on Computational learning theory, New York, NY, USA, ACM Press (1997) 250–262
17. Heckerman, D.: A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, USA (1995)
18. Chen, R., Sivakumar, K., Kargupta, H.: Collective mining of bayesian networks from distributed heterogeneous data. Knowledge and Information Systems **6**(2) (2004) 164–187
19. Batagelj, V.: Pajek - program for large networks analysis and visualization (2001)