Inference in Probabilistic Graphical Models by Graph Neural Networks - Reproduction and Extension

Clemence Granade [1] **Anita Kriz** [1] **Alireza Dizaji** [1] **Anthony Gosselin** [1] **Jeremy Qin** [1]

Abstract

Although exact inference in Probabilistic Graphical Models (PGMs) is possible on relatively small graphs and in tree-like graphs by using message passing algorithms such as Belief Propagation (BP), performing these tasks on graphs of arbitrary size and connectivity is computationally expensive and often intractable. Methods like BP can be used for approximate inference but generally struggle in loopy graphs or graphs with far-reaching node dependencies. Although used in different applications, there is a striking parallel between the fundamental concept of message passing in PGMs and Graph Neural Networks (GNN). This naturally gives way to using GNNs for approximate inference in PGMs by mapping nodes in PGMs to nodes in GNNs. In this work, we compare BP with GNN methods both in tree and non tree graph structures. To extrapolate the need for importance weighing in graph structures, we explore the use of soft attention in GNNs. Expanding on the works of Yoon et al. (2019), our work demonstrates that using this mechanism in the GNN outperforms its variants and BP on various graphs. Our code is publically available here[1].

1. Introduction

A common task when using a probabilistic graphical model (PGM) is to compute marginal probability distributions $p_i(x_i)$ at one (or each) node x_i of the graph. This value can provide insights into the uncertainty and likelihood of that node on its own, regardless of other nodes in the graph. For example, given a graph containing information about a diagnostic test, disease, and symptoms, the marginal probability of the test results is crucial in understanding outcomes in the context of disease prevalence and symptom onset.

However, as the complexity of a graph increases, exact inferences of this kind become computationally intractable. Consider having to marginalize out $|N|$ nodes, each that can take on k values. This requires the summation of $k^{|N|}$ values, which even for relatively small graphs becomes computationally expensive. For this reason, message passing algorithms that build on dynamic programming methods are required - one of which is the belief propagation (BP) algorithm. Although developed for exact inference on tree graphs, the BP algorithm can only be used as an approximation on non-tree graphs.

There is a remarkable parallelism between the message passing algorithms used in PGMs and Graph Neural Networks (GNNs), a deep learning framework that can perform prediction tasks on graph structures (Scarselli et al., 2009). Notably, both share the fundamental concept of passing messages between nodes. Given the end-to-end capabilities, GNNs that use probabilistic information as features hold significant potential in being able to approximate inferences in PGMs. More specifically, the incorporation of non-linear functions in its NN formulation has the ability to represent complex relationships between nodes. Thus, by mapping the nodes of a PGM to the nodes of a GNN, running approximate inference tasks can become feasible. Yoon et. al (2019) use two different mappings from PGM to GNN nodes, one which uses the factor nodes as the GNN nodes and the other which uses the variable nodes themselves. The intuition with the latter is that it may alleviate the representational power required when additionally mapping factor nodes, allowing for greater flexibility.

Our main contributions are the following:

1. Firstly, we provide an in depth quantitative analysis of BP compared to approximate inference via GNNs for both tree and non-tree graph structures. We highlight the theoretical reasoning of using these techniques and their respective trade-offs in terms of accuracy and computation time.

2. Secondly, we extend the work of (KiJung Yoon, 2019) by considering different GG-NN (Zemel, 2017) architectures in order to find a balance between long-term dependencies and importance of neighboring nodes. We compared the performances of the models with GRU and LSTM update functions and when incorporating a soft attention mechanism.

[1] GitHub Repository

Overall, our results establish the potential of rethinking BP by using GNNs in PGM inference tasks.

2. Background

2.1. Undirected Probabilistic Graphical Models

Probabilistic graphical models are graph stuctures that enable the representation of joint dependencies through conditional distributions between random variables. Undirected graphical models $G = (V, E)$ can be represented through factor graphs. These undirected, bipartite graphs connect individual variable nodes $i \in V$ to factor nodes $\alpha \in \mathcal{F}$. The former encode x_i individual nodes and the latter encode the interactions $\psi_\alpha(x_\alpha)$ between variable node groups $\boldsymbol{x}_\alpha$. $\boldsymbol{x}_\alpha$ contains all individual nodes i connected to the factor node α. The product of these interactions defines the probability distribution of the graph:

$$\frac{1}{Z} \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{x}_\alpha)$$

with Z is the normalizing constant.

2.2. Binary Markov Random Fields

Our experiments were run on binary Markov random fields, more particularly Ising models. These are undirected graphs encoding random variables that follow the Markov property with binary random variable nodes selected $\boldsymbol{x} \in \{-1, +1\}^{|V|}$. Ising models were one of the first Markov models to model the energy in a system according to the atom interactions. Each variable node x_i has it's intrinsic energy function as well the energy from each interacting connected atom (Friedman, 2009). Restricting to Ising models, the energy function is defined as $\psi_i(x_i) = \exp\{b_i x_i\}$ and associated edge energy function as $\psi_{i,j}(x_i, x_j) = \exp\{J_{i,j} x_i x_j\}$. Deriving the exact joint distribution:

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp(\boldsymbol{b}x + xJx)$$

with b composed of biases b_i for all $i \in V$ and J a symmetric matrix composed of $J_{i,j}$ for all $\{i, j\} \in E$. For the following experiments, b and J are selected at random.

2.3. Belief Propagation

Computing marginals (and thus conditionals) of a graphical model using naive algorithms is extremely computationally expensive. Consider a graph with N nodes $X = \{X_1, X_2, ..., X_N\}$ such that each node X_i can take on k values. To obtain the marginal of a node X_i, the intuitive solution is to marginalize out all other nodes from the joint distribution:

$$P(X_i) = \sum_{x_1} \sum_{x_2} \dots \sum_{x_{i-1}} \sum_{x_{i+1}} \dots \sum_{x_n} P(X_1, X_2, \dots, X_n) \tag{1}$$

By examination of this equation, the problem becomes clear. Computing the marginal of a single node requires computing N sums with k values to iterate through in each sum, yielding a compute complexity of $k^{|N|}$. Even for reasonably low values of both parameters, computation quickly becomes expensive, and as the number of nodes grows, intractable. Thus, there is a need for methods to compute and approximate the marginal distributions that can reduce this computation time.

Belief propagation, otherwise known as the sum-product algorithm in coding theory, is a message-passing algorithm that operates by retrieving messages associated with edges in a graph and updating them recursively via local computations done at the vertices (Montanari, 2009). In other words, BP uses dynamic programming to perform the recursion, a method that can greatly reduce computation time. This algorithm extends from the simpler elimination algorithm for single queries by using a key insight: when passing messages from one node to another to compute their marginals, we are re-using messages that will be used to compute another marginal distribution. Therefore, if instead of finding the marginal distribution of a single node X_i we want to know how *all* the nodes are distributed, we do not have to run the elimination algorithm N times. Instead, if there exists an edge between nodes X_i and X_{i+1}, we simply compute two messages, the message $\mu_{i \to i+1}$ and $\mu_{i+1 \to i}$. If we store all these messages throughout the graph, we can compute any marginal of any node that we want.

By operating BP on factor graphs, a bipartite representation where edges are only between factors and variables, there are two kinds of messages that need to be constructed, variable-to-factor $\mu_{i \to \alpha}$ and factor-to-variable $\mu_{\alpha \to i}$:

$$\mu_{i \to \alpha}(X_i) = \prod_{\beta in N_i \setminus \alpha} \mu_{\beta \to i}(X_i) \tag{2}$$

$$\mu_{\alpha \to i}(X_i) = \sum_{X_\alpha \setminus X_i} \psi_\alpha(X_\alpha) \prod_{j \in N_\alpha \setminus i} \mu_{j \to \alpha}(X_j) \tag{3}$$

where N_i are the neighbors of variable node X_i (these are the factors that involve X_i) and N_α are the neighbors of factor nodes α (these are the variables that are directly coupled by $\psi_\alpha(X_\alpha)$ (KiJung Yoon, 2019). These two equations can be jointly interpreted as messages sent out by factor nodes to other factor nodes. Firstly, the incoming messages to a factor node α is computed as $\mu_{i \to \alpha}$ by multiplying all the incoming messages from the connected factor nodes. In the next step, these incoming messages are multiplied by the

factor node α_s, and marginalized over all the variable nodes involved in that factor.

However, there is one significant caveat: BP can only extract *exact* marginals on tree structures (Montanari, 2009). This is possible due to the non-cyclic nature of trees that allow messages to be passed unidirectionally and without conflicting information. Although the elimination algorithm itself can be generalized to any graphical model, it can only compute a single query. Moreover, the complexity of the algorithm is completely controlled by the elimination ordering of the variables, and this in itself is a computationally intensive process. Due to its efficiency and intuitive formulation, recent work has extended BP to approximate marginals for graphs with cycles, otherwise known as loopy belief propagation. This general case algorithm can be summarized with two steps:

1. Initialize the messages between variables and factors with a uniform distribution
2. At every step t, compute an updated message using the previous messages, until convergence (further iteraions no longer change the messages significantly):

$$\mu_{i\to\alpha}^{(t)}(X_i) = \prod_{\beta\in N_i\setminus\alpha} \mu_{\beta\to i}^{(t-1)}(X_i) \tag{4}$$

$$\mu_{\alpha\to i}^{(t)}(X_i) = \sum_{X_\alpha\setminus X_i} \psi_\alpha(X_\alpha) \prod_{j\in N_\alpha\setminus i} \mu_{j\to\alpha}^{(t-1)}(X_j) \tag{5}$$

In other words, messages are being passed around until convergence. However, convergence is not guaranteed in non-tree graphs. The intuition for this is that since BP is a local algorithm, it should be successful whenever the underlying graph is locally a tree. However, the trade-off with being able to define distributions locally is indeed that far apart variables become uncorrelated. Thus, in graphs where far apart variables are important in understanding the underlying distributions, BP will perform poorly (Montanari, 2009). Moreover, cycles can introduce ambiguity in message passing, and the iterative updates may not converge or may converge to incorrect results.This analysis of BP naturally leads to the need for alternate methods that can improve approximation performance on non-tree graphs while remaining computationally tractable.

2.4. Graph Neural Networks

Graph Neural Networks, a deep learning framework that uses the structure and feature information in a graph, allows for complex transformations between nodes. Therefore, GNN can encode probabilistic information about variables in the graphical model as nodes and send and receive messages (that are learned via non-linear transformation) about the probabilities. Finally, after training, a nonlinear decoder could be used to approximate the marginal probabilities from each node. Initially aligned with the methods from (KiJung Yoon, 2019), we trained a Gated Graph Neural Network (GG-NN) (Zemel, 2017) updating nodal hidden states at each time (t). The non-linearities are applied to obtain the hidden state vectors $h_i^{(T)}$ for each of the GNN node v_i. The hidden state vectors are initialized at 0 and updated each time step t, with the messages received from neighboring nodes. For each set of connected $\{v_i, v_j\}$ in the GNN, a message $\boldsymbol{m}_{i\to j}^{(t+1)}$ from v_i to v_j is sent at time $t+1$:

$$\boldsymbol{m}_{i\to j}^{(t+1)} = \mathcal{M}\left(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, e_{ij}\right) \tag{6}$$

with $\mathcal{M}$ here is a multi-layer perceptron (MLP) with rectified linear units (ReLU), $e_{i,j}$ edge labels/properties between the nodes v_i, v_j and $N(j)$ the neighboring nodes of v_j. At time t, the messages received by a node v_i is the sum of all incoming messages:

$$\boldsymbol{m}_i^{(t+1)} = \sum_{j\in N(i)} \boldsymbol{m}_{j\to i}^{t+1} \tag{7}$$

with $N(i)$ the indices of neighboring nodes for node v_i. The update function for a hidden state vector at time t for a node v_i is defined by:

$$\boldsymbol{h}_i^{(t)} = \mathcal{U}(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{m}_i^{(t)})$$

with $\mathcal{U}$ a specified update function, either a Gated Recurrent Unit (GRU) or an Long-Short Term Memory (LSTM). At the final state T the output marginals $\hat{y}$ are obtained from:

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{h}^{(T)})$$

This method proceeds from a variation of (KiJung Yoon, 2019) derived from (Zemel, 2017)'s GG-NNs. The training is done using backpropagation minimizing the loss function $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$ (in this work, we use cross-entropy loss which will be detailed in a later section). Given the architecture of GNNs, it is natural to think of how to extend it to the task of inference in PGMs. GNN nodes can encode probabilistic information about variables in the graphical model by sending and receiving messages. These messages are learned via nonlinear transformations that allow for complex relationships between nodes. With these parallels between message passing algorithms in PGMs and GNNs, and the potential of GNNs to learn probabilistic information in a computationally efficient way, the next point of discussion would be: *how can we map the nodes of a PGM to nodes in GNNs*? As shown in Figure 1a, (KiJung Yoon, 2019) describe two different mappings that can be used.

2.4.1. Message Mapping

The first mapping as shown in Figure 1b more closely resembles the structure of conventional BP, where the node in

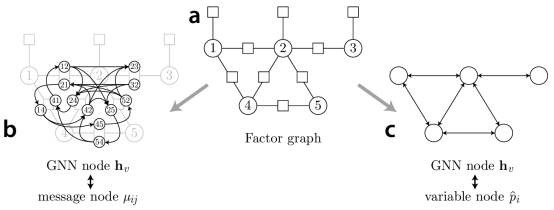

Figure 1. Mappings of PGM nodes into GNN nodes

the GNN corresponds to the message $\mu_{i,j}$. Thus in a factor graph, this is just the factor itself. As demonstrated in 1b, since messages flow bidirectionally, there are two messages per pairwise factor. To compute the message from variable node v_i to v_j in the PGM using the GNN updates described, we have the following updates at each iteration t:

$$m_{i\to j}^{(t+1)} = \mathcal{M}\left(\sum_{k\in N_i\setminus j} h_{k\to i}^{t}, e_{ij}\right) \tag{8}$$

Where the hidden states are aggregated by summing over all hidden states involving variable node v_i (i.e. its neighbors). The hidden state is then updated by:

$$h_{i\to j}^{(t+1)} = \mathcal{U}(h_{i\to j}^{(t)}, \boldsymbol{m}_{i\to j}^{(t+1)}) \tag{9}$$

After convergence, the node marginals can be readily extracted. For example, to find the marginal of a variable node X_i, the hidden states of all its neighbors are summed over to obtain the hidden state of node X_i, representing its marginal distribution in this case:

$$\hat{p}_i(X_i) = \mathcal{R}\left(\sum_{j\in N_i} h_{j\to i}^{T}\right) \tag{10}$$

2.4.2. Variable Mapping

The second possible mapping, as shown in Figure 1c, uses the variable nodes themselves as the GNN nodes. Given this formulation, there will be no hidden states that correspond to the factor nodes (and thus they will not be updated). The idea is that the factor nodes are still influencing the inference as they are embedded in the edges of the GNN. By relying on these parameters being passed into the message function on each iteration, we can avoid the representational power on the underlying factor nodes. Thus, the equations for updates are:

$$m_{i\to j}^{(t+1)} = \mathcal{M}\left(h_i^t, h_j^t, e_{ij}\right) \tag{11}$$

Where the hidden states now directly correspond to their variable node in the PGM. The messages are then aggregated into a single message for the destination node;

$$m_i^{(t+1)} = \sum_{j\in N_i} m_{j\to i}^{(t+1)} \tag{12}$$

Finally, the hidden state is updated by:

$$h_{i\to j}^{(t+1)} = \mathcal{U}(h_i^{(t)}, \boldsymbol{m}_i^{(t+1)}) \tag{13}$$

Given the one to one mapping of variable node to GNN node, the readout can be obtained readily from the corresponding GNN node, $\mathbf{h}_v$:

$$\hat{p}_i(X_i) = \mathcal{R}(h_i^T) \tag{14}$$

2.5. Integrating Information from Past States

The learning method for our model is done through a time-sequential update of each node, using the update function $\mathcal{U}$ with which certain information from previous states is conserved, forgotten and combined with the new message information. To do this, we used a special type of GNN, Gated Graph Neural Networks. More specifically, we considered using both a Long Short Term Memory (LSTM) cell as well as a Gated Reccurent Unit cell. In addition to this, we also explore the use of soft attention in our work.

We implemented the LSTM and GRU as update function to regulate the information selection from time t to $t+1$, as update functions. LSTMs were developped to solve the vanishing gradient problem for traditional recurrent neural networks (RNN) by incorporating specialized memory cells and gating mechanisms (an input gate, a forget gate and an output gate). These gates allow LSTMs to selectively retain information over time, enabling the model to capture and remember long-term dependencies within the sequential data in a selective manner. The GRUs, used in the implementations done by Yoon (KiJung Yoon, 2019) were built as a variant of the LSTM architecture. They also exploit the gating mechanism to selectively include and pass on information. The memory cell is composed of an update gate, that combines the forget gate and the input gate, and of a reset gate. Both are known for solving the vanish gradient problem for short-term context and perform similarly well (Chung et al., 2014).

While both were a great advance from traditional RNNs, they struggle with long-term contextual learning and long-run vanishing gradient. Both benefited from attention mechanisms which enable the model to focus on specific elements within the graph, assigning varying degrees of importance to different nodes during the sequential update process through a trained weighting process. We do this by using a MLP layer with LeakyRelu activation function that learns the importance of each messages and scale them with a Softmax

function so that the weights sum to one. Through a weighted combination of these components, attention mechanisms allows the model to selectively consider relevant nodes. This adaptability and selective attention is useful in scenarios where certain nodes or connections play a more important role in the sequential evolution of the node data update.

3. Experiments

3.1. Graph Selection and Task Settings

To understand how the two GNN methods work for approximate inference compared to BP, we strategically selected graph structures to emphasize their differences by choosing both sparse and highly connected graphs including grid and tree structures.

Each graph structure was generated with $|N| = 9$ and $|N| = 16$ nodes, where each node $x_i \in \{+1, -1\}$. The 6 graph structures of interest are shown in Figure 2. For data generation, we used the code-base provided by (Lingxiao Zhao, 2019) that can be found here. To compare the performance and generalization capability of the different methods experimented on, we defined two different task settings.

- In-sample: each GNN implementation was trained and evaluated on each graph structure type individually.
- Combined: each GNN implementation was trained on a combined dataset of 13 graph structures (structures defined in (KiJung Yoon, 2019)), and evaluated on the 6 graphs of interest.

With regards to the in-sample experiments, for each graph structure and size set, we generated 5000 training and 1000 testing samples. For the combined experiments, we generated and combined 100 training samples of each graph structure (13 total structures) and 100 test samples. Using their codebase, the true marginal values were generated with exhaustive enumeration of states.

3.2. Tested Algorithms

We compared the BP algorithm to different implementations of GNNs with both the Message and the Variable Mapping. As presented above, the GNNs followed GG-NN architectures and we compared for each mapping a GNN with a GRU update function, a GNN with an LSTM update function. Additionally, for Message mapping we implemented a GNN with GRU update function coupled with an additional attention layer.

The GNNs where trained using the cross entropy loss: $L(\boldsymbol{p}, \hat{\boldsymbol{p}}) = -\boldsymbol{p}_i(x_i) \log(\hat{\boldsymbol{p}}_i(x_i))$ between exact and estimated marginals.

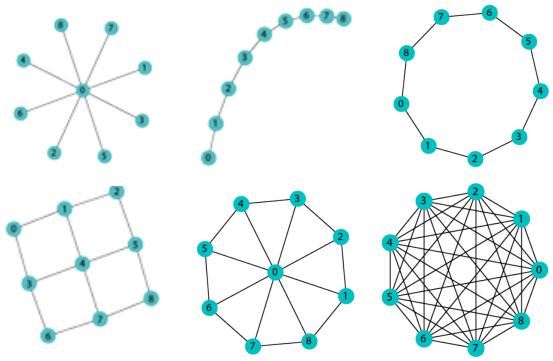

Figure 2. Explored Graph Structures. From left to right starting from the top: "star", "path", "cycle", "grid", "wheel" and "FC".

3.3. Computational Complexity Analysis

Additionally, evaluate the GNNs robustness with regards to inference time complexity. In addition to performance saw it valuable to track and compare the total inference time complexity of these models, in comparison to BP. All inference experiments run on a single laptop (cpu: AMD Ryzen 7 5700u).

4. Results

We obtain the 'In-sample' and 'Combined' model performance results as well as the time complexity for 'Combined' inference, presented respectively in 3,

4.1. In-Sample Performance

The in-sample results are presented in Figure 3. The GNNs with variable mapping and message mapping are identified as "Var" and "Msg", respectively. We quantify performance by taking the negative log10 of the average Kullback-Leibler divergence between the exact and estimated marginals:

$$performance = -\frac{1}{N}\sum_{i=1}^{N} log_{10}(D_{KL}[p_i(x_i)||\hat{p}_i(x_i)]) \tag{15}$$

We observe that the best performance results across all graph structures of all sizes are shared between two of our proposed extended GNN models: the message mapping GNN with LSTM update function and the message mapping GNN with additional attention layer: "Msg_lstm" and "Msg_attn", respectively. Our intuition behind this is that the LSTM and attention layer can increase the capacity of GNNs for modeling the dependencies, specially between long and complex graphs.

We note that BP is not exact even on tree graphs. The authors of the BP implementation that we based our tests on observed similar results and stated: "We thoroughly checked

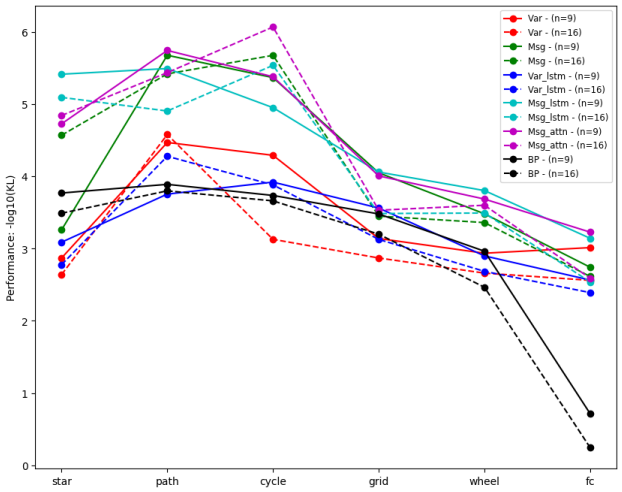

Figure 3. In-sample experiment: GNNs trained on separate individual graph structures and sizes (n=9 & n=16). Performance reported as negative log10(KL) w.r.t. ground-truth marginals.

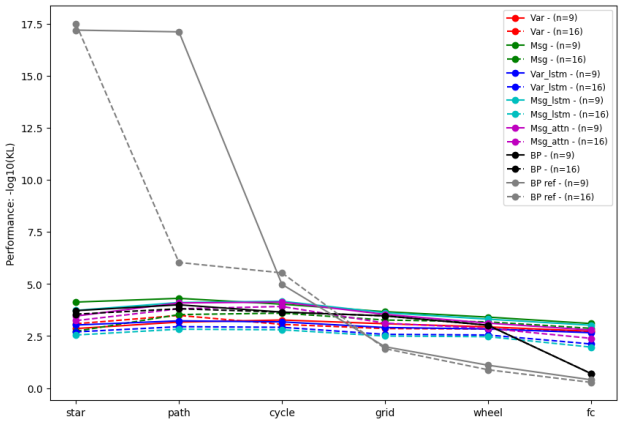

Figure 4. Combined experiment: GNNs trained on combined dataset. Performance (- log10(KL)) on different graph types (n=9 & n=16). BP performance as reported in reference paper labeled as "BP ref"

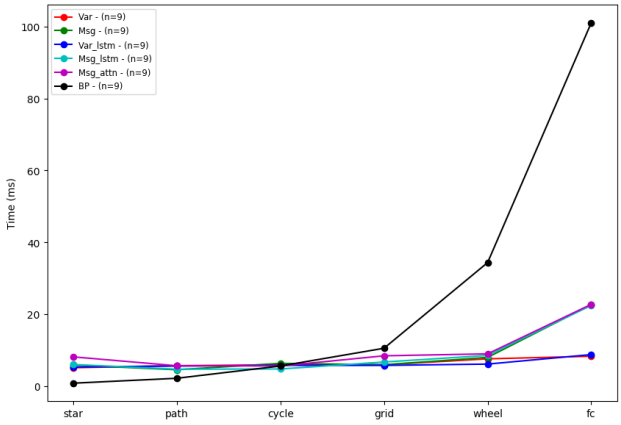

Figure 5. Average graph inference time (in milliseconds) on small (n=9) graph structures for each method.

our implementation and attribute this to accumulation of numerical error" (Lingxiao Zhao, 2019). We have also carefully verified the implementation without finding any errors, so we conclude likewise.

4.2. Combined Performance

The "combined" results are presented in Figure 4. The GNNs trained on the combined dataset of 13 graph structures were able to generalize to the different graph types and demonstrate similar performance results as the in-sample case (Figure 3). Figure 4 also showcases the reported BP performance from the reference paper (KiJung Yoon, 2019) ("BP ref" in the figure). Comparing with the reference BP performance, we note that BP performs best on simple tree-like graphs, but underperforms relative to the GNNs on the loopier graph structures.

4.3. Computation Time

Here we compare inference times of the various methods on the small graphs in Figure 5. We observe that BP performs faster on the sparse graphs ("star" and "path"), but as the graphs get denser, its computation grows exponentially, while the GNNs perform increasingly and significantly faster. Akin to the performance results presented in the previous section, the GNNs scale better than BP to complex graphs.

5. Conclusion and Future Directions

In this work, we experimented with how GNNs perform at inference tasks, particularly taking the marginal distribution of each variable, compared to belief propagation. To make a fair comparison, we thoroughly experimented with different aspects, including different graph dependencies and complexities. We observed that the GNNs perform far better than belief propagation whenever we increase the complexity of graphical models all the while remaining computationally efficient. For BP, on the other hand, the time complexity grows exponentially and the performance drops significantly. We integrated GNNs with different update functions, particularly LSTMs and attention layers, which boost the performance on different inference tasks, an idea that was not examined in the original paper. For future development, we are interested in reformulating other inference tasks, such as MAP and MLE, with GNNs. Moreover, the main focus of this work was on the binary graphical models,

extending to non-binary ones is also a logical and interesting next step.

References

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014.

Friedman, D. K. N. *Probabilistic Graphical Models, principles and techniques*. Oxford University Press, 2009.

KiJung Yoon, e. a. Inference in probabilistic graphical models by graph neural networks. 2019.

Lingxiao Zhao, Ksenia Korovina, W. S. M. C. Approximate inference with graph neural networks. 2019.

Montanari, M. M. A. *Information, Physics, and Computation*. Oxford University Press, 2009.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Zemel, Y. L. R. Gated sequential neural networks. 2017.