Prioritized Asynchronous Belief Propagation

Jiarong Jiang JIARONG@UMIACS.UMD.EDU
University of Maryland, College Park, MD 20740

Taesun Moon TSMOON@UMIACS.UMD.EDU
University of Maryland, College Park, MD 20740

Hal Daumé III HAL@UMIACS.UMD.EDU
University of Maryland, College Park, MD 20740

Jason Eisner JASON@CS.JHU.EDU
Johns Hopkins University, Baltimore, MD 21218

Abstract

Message scheduling is shown to be very effective in belief propagation (BP) algorithms. However, most existing scheduling algorithms use fixed heuristics regardless of the structure of the graphs or properties of the distribution. On the other hand, designing different scheduling heuristics for all graph structures are not feasible. In this paper, we propose a reinforcement learning based message scheduling framework (RLBP) to learn the heuristics automatically which generalizes to any graph structures and distributions. In the experiments, we show that the learned problem-specific heuristics largely outperform other baselines in speed.

1. Introduction

Probabilistic graphical models (Pearl, 1988; Wainwright & Jordan, 2008; Jordan, 1999) play an important role in representing complex distributions and dependencies between random variables for many real world applications. Many approximate inference algorithms have been proposed to solve the problem efficiently. One of the most common methods is message-passing algorithms such as loopy belief propagation (Murphy et al., 1999; Ihler et al., 2005; Yedidia et al., 2000). The idea behind this is to pass messages between adjacent nodes until the fixed points of the beliefs are achieved.

Inferning Workshop at 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. Copyright 2013 by the author(s).

It is shown in Elidan (2006) that asynchronous propagation can achieve better convergence compared to synchronous methods and thus initiated interest in studying message scheduling schemes. However, most existing research only focuses on manually designed heuristics. For example, Wainwright et al. proposed a tree reparameterization algorithm as message scheduling for asynchronous propagation. Elidan (2006) orders the messages in the order of the differences between two consecutive values of the message. Yedidia et al. (2005); Vila Casado et al. (2010) also design specific scheduling for LDPC decoding problems.

In contrast with standard scheduling algorithms, in this paper, we explore *automatically* scheduling messages for any graph structure and distribution. To achieve this, we formalized the learning of the ordering as a Markov Decision Process (MDP) and associate the final reward with the convergence speed. The ordering can be viewed as a linear policy of the MDP. The goal is trying to learn a good policy such that the corresponding scheduling can achieve a better convergence speed. In the experiments, we show that that for the graphs that the heuristics are automatically learned, the learned heuristics performs better than the baselines. We also notice that the learned weights reveal some useful features to consider in the scheduling.

2. Background

In this paper, we consider the problem of marginal inference on undirected and discrete graphical models. Formally, we denote a graph $G = (X, E)$ and let $x = \{x_1, ..., x_m\}$ to be the random variables associated with nodes in X. The exponential family distribution p over the random variables is defined as follows:

$$p_\theta(x) = \exp[\langle \theta, \phi(x) \rangle - A(\theta)] \quad (1)$$

where $\phi(x)$ is the sufficient statistics of x and θ is the canonical or exponential parameters. $A(\theta) = \log \sum_x \exp[\langle \theta, \phi(x) \rangle]$ is the log-partition function. In this paper, we will focus on solving the marginal problem which is inferring marginal distribution $p(x)$ for all x.

2.1. Belief Propagation

Belief propagation (Murphy et al., 1999; Ihler et al., 2005; Yedidia et al., 2000) (or sum-product) algorithm is the standard message-passing algorithm for inferring marginal distributions over random variables. It is a fixed point iteration algorithm where the fixed point is the exact solution to trees or polytrees structures (Pearl, 1988). On loopy graphs, they are not guaranteed to converge, but if they do, the final estimates are shown to be reasonably good (Yedidia et al., 2000).

The message M_{ts} passed from node t to one of its neighbors s is defined as:

$$M_{ts}(x_s) \leftarrow \kappa \sum_{x'_t \in \mathcal{X}_t} \Big\{ \exp[\theta_{st}(x_s, x_{t'}) + \theta_t(x_{t'})] \prod_{u \in N(t) \backslash s} M_{ut}(x_{t'}) \Big\} \quad (2)$$

where κ is the normalization constant. $N(t) \backslash s$ are the neighbors of t excluding s. When the messages converge, the belief/psuedomarginal distribution for the node s is given by

$$\mu_s(x_s) = \kappa \exp\{\theta_s(x_s)\} \prod_{t \in N(s)} M_{ts}(x_s) \quad (3)$$

For synchronous message-passing, only the old messages from the previous iteration are used to compute the messages for the current iteration. However, asynchronous message-passing uses the latest messages for updates. In Elidan (2006), it is shown that any reasonable asynchronous BP algorithm converges to a unique fixed point under the assumptions that are similar to the synchronous version. They use the residues (differences in beliefs between updates) as a priority to schedule the messages. In this paper, instead of using the greedy residual schedule, we learn a proper schedule automatically by a reinforcement learner.

3. Message Scheduling with Reinforcement Learner

We formulate the schedule learning problem as a MDP and the goal is to learn a policy for the MDP such that the corresponding ordering can improve the speed of convergence. Note that in this paper, we will only consider ordering the nodes in the graph, but the learning framework can be easily generalized to ordering each message in the graph.

3.1. MDP Formulation

A Markov Decision Process (MDP) (Bellman, 1957; Puterman, 2009) can be viewed as a memoryless search process. It consists of a state space S, an action space A, a transition function T, a reward function R and the environment $\mathcal{E}$. An agent repeatedly observes the current state $s \in S$ and takes an action $a \in A$. The environment $\mathcal{E}$ then samples the new state s' from the transition function $T(s'|s, a)$ and gives a reward $R(s'|a, s)$. A policy $\pi(a|s)$ describes how the agent chooses an action based on its current state.

More specifically, for the schedule learning problem, the state space S is the graph and its messages/beliefs. The action to take at each step is choosing a node x in the graph and compute its messages. Assume the features associated with node x is $\phi(x)$ and the transition function T is deterministic. The reward is defined with regard to the convergence speed. Here we consider a linear policy (priority) which is

$$\pi_\omega(s) = \arg\max_x \omega \cdot \phi(x) \quad (4)$$

where ω is the feature weight vector. The goal is to maximize the expected reward $R = \mathbb{E}_{r \sim \pi_\omega}[R(\tau)] = \mathbb{E}_{r \sim \pi_\omega}[\sum_{t=0}^{T} r_t]$ where $\tau = (s_0, a_0, r_0, ..., s_T, a_T, r_T)$ is a trajectory. In the current situation, this is equivalent to minimizing the number of messages passed.

At test time, the transition function is deterministic so that we always choose the node that has the highest priority in the current state s. However, at the training time, we allow the agent to explore the space by a stochastic policy:

$$\pi_\omega(x_t) = \frac{1}{Z} \exp\left(\frac{1}{T} \omega \cdot \phi(x)\right) \quad (5)$$

where T is temperature and Z is a normalization constant. When $T \to \infty$, π_ω achieves a random choice over all the nodes while $T \to 0$ only exploits the deterministic policy.

To solve this, we apply the standard policy gradient algorithm (Sutton et al., 2000).

$$\begin{aligned} \nabla_\omega \mathbb{E}_\tau[R(\tau)] &= \mathbb{E}_\tau \left[\frac{\nabla_\omega p_\omega(\tau)}{p_\omega(\tau)} (R(\tau) - b) \right] \\ &= \mathbb{E}_\tau \left[(R(\tau) - b) \sum_{t=0}^{T} \nabla_\omega \log \pi(x_t) \right] \end{aligned} \quad (6)$$

where b is some arbitrary baseline. In the last term, the gradient of each policy decision is:

$$\nabla_\omega \log \pi_\omega(x_t) = \frac{1}{T}\left[\phi(x_t) - \sum_{x'_t} \pi_\omega(x'_t)\phi(x'_t)\right] \quad (7)$$

Here we also use the optimal baseline

$$b = \frac{\left\langle \left(\sum_{t=0}^{T} \nabla_\omega \log \pi_\omega(x_t)\right)^2 R(\tau) \right\rangle}{\left\langle \left(\sum_{t=0}^{T} \nabla_\omega \log \pi_\omega(x_t)\right)^2 \right\rangle} \quad (8)$$

and the averaged gradient

$$g = \left\langle \left(\sum_{t=0}^{T} \nabla_\omega \log \pi_\omega(x_t)\right)(R(\tau) - b) \right\rangle \quad (9)$$

3.2. Algorithm

A detailed training algorithm is described in Algorithm 1. In the training, assume node and edge potentials are given for a set of training graphs. The goal is to learn the policy (feature weights) so that the ordering of the nodes will help the model convergence. One run on a graph is treated as a trajectory for the learner.

A priority queue is maintained to order the nodes of the graph. For each iteration, a node is popped from the queue according to a Boltzmann exploration policy. Then the relevant messages are updated for the node and the features (and their priorities) of the neighboring nodes are also updated. One iteration is over when there are no more nodes in the priority queue. If the beliefs are not converged, then push all the nodes to the priority queue and repeat. Continue this process until the gradient converges and update the weight vector ω. Return the final weight vector ω when the gradient $\rightarrow 0$.

During the test time, the learned deterministic policy will be used to prioritize the nodes and the node with the highest priority is updated for each step.

3.3. Features

The features we use here include both static features and dynamic features. The static features are mostly extracted according to the graph properties once per inference run. Dynamic features are associated with real-time updates and a change in the values of dynamic features for a node can impact the feature values for other nodes in the graph.

Algorithm 1 Training Algorithm

```
Input: Training set with graphs and given potentials.
Initialize the policy parameter $\omega$.
for each graph in the training set do
  Initialize the messages and priority queue $Q$ with all the nodes in the graph.
  while the beliefs are not converged or the maximum number of iterations is not reached do
    Pop a node $x$ from $Q$ according to (5).
    Compute its messages by (3).
    Compute the derivative $\nabla_\omega \log \pi(x_t|Q_t)$ by (7).
    Update the dynamic features of its neighbors.
    if the priority queue $Q$ is empty and algorithm not converged then
      Enqueue all the nodes.
    end if
  end while
  Compute the reward of the current trajectory.
  Compute the baseline by (8) and gradient by (9).
  Update the policy when the gradient converges.
end for
```

3.3.1. Static Features

The static features are:

- the degrees of the node,
- the dimension of the node,
- the max/min degrees of the neighboring nodes,
- the max/min dimension of the neighboring nodes

3.3.2. Dynamic Features

Dynamic features are those features that change between iterations or even within a single iteration. Here we use

- KL divergence and residual between the current learned belief and previous belief
- the difference between the current and last incoming messages
- the difference between the outgoing messages.

4. Experiments

To empirically evaluate the model, we randomly generate Ising models according to the experiment setups in (Elidan, 2006)[1]: The graphical representation of an

[1]As ongoing work, we only show the preliminary results here.

Ising model is a $N \times N$ random grid with a corresponding number of binary variables. The node potentials are drawn from $U[0,1]$ and the pairwised potentials

$$\phi_{s,t}(X_s, X_t) = \begin{cases} \exp(\lambda C) & \text{if } x_s = x_t \\ \exp(-\lambda C) & \text{if } x_s \neq x_t \end{cases} \quad (10)$$

where $\lambda \sim U[-0.5, 0.5]$ and C is a constant.

We evaluate our reinforcement learning based message scheduling framework (RLBP) against the standard loopy belief propagation (LBP) and residual belief propogation (RBP). We also propose and evaluate against a novel variant of the RBP where instead of measuring the L2 differences between belief updates, we use a KL divergence type of difference (KLBP).

We generate 40 5×5 grid graphs with $C = 5$. We train our scheduling policy on the same graphs for which we report convergence results. LBP converges on 19 graphs and the remaining 3 algorithms converge on 27 graphs. We only try to learn the scheduling on the 27 graphs where either RBP or KLBP converges. In table 1, we show results in terms of time reduction over LBP per model. For clarity, LBP is shown as having 0% time reduction over itself. It is clear that RLBP outperforms all other models by a wide margin.

LBP	0%	RBP	34.70%
KLBP	39.62%	RLBP	55.10%

Table 1. Time reduction in percentage over LBP by propagation framework

Figure 1 shows the number of iterations taken for each of the 27 graphs. Note some of the results for LBP are missing because it does not converge. In order to make the plot easy to read, we order the graphs on the x-axis by the number of iterations RLBP (red line) takes.

It can be seen that RBP and KLBP also show large reductions in the time to convergence and sometimes one scheme is better than the other. However, the learned heuristics (RLBP) performs the best of all most of time. One interesting observation in the final feature weights is that the degrees of the node here is a very indicative feature besides the difference or divergence of the beliefs, which supports the fact that different graph structures should have different scheduling for the messages.

5. Discussion and Future Work

In this paper, we presented a novel, effective algorithm for learning how to schedule messages automatically with a reinforcement learner. We restricted our focus

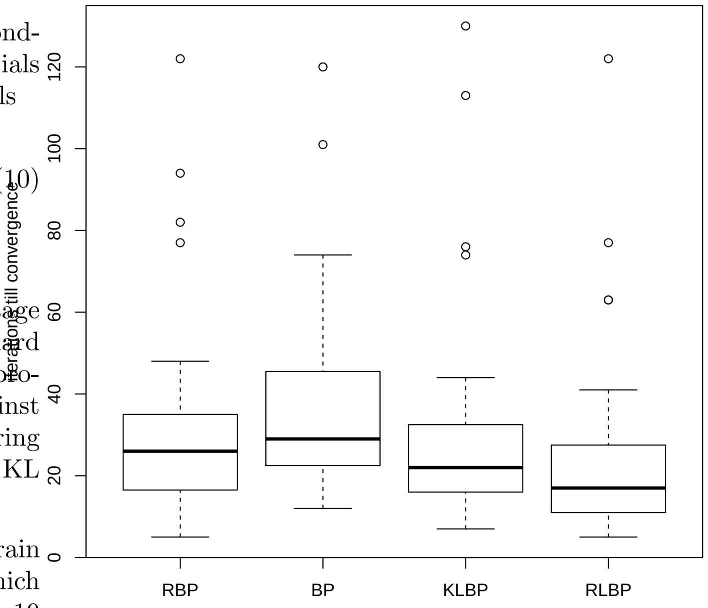

Figure 1. Plot of iterations per model until convergence

here on learning the prioritization for the nodes in the graph. However, in the general case, we can also apply the same learning framework to learning or pruning the specific messages in the belief propagation. If the graphical model is large with many variables or variables sitting in a high dimension space, the edge-based scheduling becomes harder. More complicated models should be considered such as merging variables and performing inference on different parts of the graph, coarse-to-fine message passing, etc.

References

Bellman, Richard. A markovian decision process. Technical report, DTIC Document, 1957.

Elidan, Gal. Residual belief propagation: Informed scheduling for asynchronous message passing. In *in Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI*, 2006.

Ihler, Alexander T., Iii, John W. Fisher, Willsky, Alan S., and Chickering, Maxwell. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6: 905–936, 2005.

Jordan, Michael I. An introduction to variational methods for graphical models. In *Machine Learning*, pp. 183–233. MIT Press, 1999.

Murphy, Kevin P., Weiss, Yair, and Jordan, Michael I. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of Uncertainty in AI*, pp. 467–475, 1999.

Pearl, Judea. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference*. Morgan Kaufmann Pub, 1988.

Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. Wiley-Interscience, 2009.

Sutton, Richard S, McAllester, David, Singh, Satinder, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12(22), 2000.

Vila Casado, Andres I, Griot, Miguel, and Wesel, Richard D. Ldpc decoders with informed dynamic scheduling. *Communications, IEEE Transactions on*, 58(12):3470–3479, 2010.

Wainwright, Martin J and Jordan, Michael I. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

Wainwright, MJ, Jaakkola, T, and Willsky, AS. Tree-based reparameterization for approximate estimation on loopy graphs. *Advances in Neural Information Processing Systems*, 14.

Yedidia, Jonathan S., Freeman, William T., and Weiss, Yair. Generalized belief propagation. In *IN NIPS 13*, pp. 689–695. MIT Press, 2000.

Yedidia, Jonathan S., Wang, Yige, Wang, Yige, Zhang, Juntan, Zhang, Juntan, Fossorier, Marc, and Fossorier, Marc. Reduced latency iterative decoding of ldpc codes. In *Proc. of the IEEE Global Communications Conf*, 2005.