Inducing Value Sparsity for Parallel Inference in Tree-shaped Models

Sameer Singh **Brian Martin** **Andrew McCallum**
Department of Computer Science, University of Massachusetts, Amherst, MA 01003
{sameer, martin, mccallum}@cs.umass.edu

1 Introduction

Single-core architectures are rapidly on the decline, and even the most common computational devices now contain multiple cores. With this easy access to parallelism, the machine learning community needs to go beyond treating the running time as the only computational resource and needs to study approaches that take this additional form of flexibility into account. In this work, we study inference in tree-shaped models. Specifically, we focus on balancing accuracy and efficient use of multiple cores when faced with running time constraints.

The problem of inference for tree-shaped models (such as chains) plays an important role as a large number of real-world problems are best modeled as linear-chains or trees [1, 2, 3, 4, 5]. A fast approximate inference approach can allow inference over very large models (such as those that model documents instead of sentences) for which exact inference is slow. Second, even for smaller (sentence level) models, the computational limits for certain tasks may be too severe to allow exact inference. One such example is query analysis for real-time search [6].

In this work we introduce an approach to utilize the inherent sparsity in variable marginals for faster, parallel inference. In many real-world applications, marginals of many variables at the end of inference are often peaked at a single value. We detect this value sparsity at earlier stages of inference, and utilize it to dynamically decompose the model into smaller pieces. In particular, we observe the marginal of each latent variable in the model, and as soon as the probability of the max-marginal value crosses a pre-defined threshold (ζ), we treat the variable as *deterministic* (i.e. having a fixed value). The sparsity decisions allow us to split the tree at these "islands of certainty", thereby decomposing the model into smaller, separate inference tasks. This enables us to utilize multiple cores to provide further speedups. To revisit the sparsity decisions made during earlier stages of inference, we also allow the deterministic variables to become uncertain about their values.

The proposed approach can provide a way to balance accuracy with constraints on running time and the number of cores available, which is not possible with exact inference. The running time of exact inference is fixed for a given chain, and using number of cores ≥ 2 does not provide any speedups. The approximation introduced by our approach is defined by the parameter ζ, where $\zeta = 1$ denotes exact inference, and $\zeta = 0$ denotes only propagating local factors. By lowering ζ, we improve both the running time of inference, and the speedups achieved by using multiple cores, however it also adversely affects the accuracy. Given the computational constraints on time and number of cores, we can set ζ for our inference algorithm to provide the best accuracy.

Some existing work addresses approximate inference in chains. Tsuruoka et. al. present a heuristic for performing inference in "easiest-first" order, avoiding full bidirectional inference [7]. Shen et. al. have proposed learning the order of inference instead of employing runtime heuristics [8]. These differ from our proposed approach as they do not revisit earlier decisions, address parallelism or computational resources.

Although not the focus of this work, a faster, multi-core inference method for tree-shaped models may facilitate quicker inference in loopy models. First, if the loopy model has a relatively low tree-width, our proposed approach can be applied to the resulting junction tree. Second, a number of approximate inference approaches for loopy models can be represented as using tree inference as an underlying subroutine [9] and can directly benefit from a faster, parallel tree inference algorithm.

Some parallel approximate inference approaches rely on single core tree inference [10, 11], and the ability to use multiple cores for the underlying subroutine allows additional flexibility.

2 Exact Inference

An undirected, linear-chain graphical model defines a probability distribution over a sequence of variables $\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_n\}$ where each variable Y_i can take a value $y_i \in \mathcal{L}$. The probability distribution is defined by a set of *local* factors $\psi_i : \mathcal{L} \rightarrow \mathcal{R}, i \in [1, n]$ and a set of *transition* factors $\psi_{i,j} : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{R}, i \in [1, n-1], j = i+1$. For a given assignment to the variables $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$, the probability of the configuration is:

$$p(\mathbf{Y} = \mathbf{y}) = \frac{1}{Z} \prod_{i \in [1,n]} \psi_i(y_i) \prod_{i \in [1,n-1]} \psi_{i,i+1}(y_i, y_{i+1}) \quad (1)$$

$$\text{where } Z = \sum_{\mathbf{y} \in \mathcal{L}^n} \prod_{i \in [1,n]} \psi_i(y_i) \prod_{i \in [1,n-1]} \psi_{i,i+1}(y_i, y_{i+1}) \quad (2)$$

The primary task of marginal inference is to compute the partition function Z, and involves initializing the per-variable marginals $\mathbf{m}_i, i \in [1, n]$ to that induced by the local factor ψ_i, followed by a *forward* and a *backward* pass. The forward pass computes the messages in the forward direction:

$$\mathbf{m}_i^f(k) \propto \mathbf{m}_i(k) \sum_{l \in \mathcal{L}} \psi_{i-1,i}(l, k) \times \mathbf{m}_{i-1}^f(l). \quad (3)$$

During the backward pass, messages are sent from the end of the chain as:

$$\mathbf{m}_i^b(k) \propto \mathbf{m}_i(k) \sum_{l \in \mathcal{L}} \psi_{i,i+1}(l, k) \times \mathbf{m}_{i+1}^b(l). \quad (4)$$

At the end of the passes, the marginal for each variable is set to the product of the two messages, i.e. $\mathbf{m}_i \propto \mathbf{m}_i^f \times \mathbf{m}_i^b$. The running time of this algorithm is $O(n \cdot |\mathcal{L}|^2)$ since $2n$ messages are sent, and computation of each message takes $O(\mathcal{L}^2)$. Further, the two passes can run simultaneously, leading to a trivial parallel inference algorithm on 2 cores that provides a speedup of 2. An example of the scheduling of messages for this case is shown in Figure 1a.

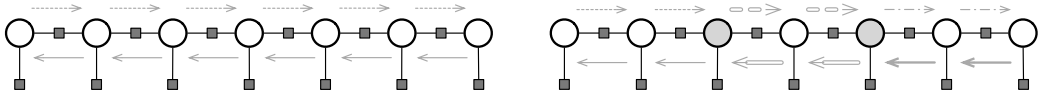
We implement the exact inference algorithm as a queue of message computations. We initialize the queue to only contain the forward message $\mathbf{m}_1^f$ and the backward message $\mathbf{m}_n^b$. At each step the message to be computed is dequeued from the queue, its value is calculated using Equation 3 or 4, and the marginal of the variable ($\mathbf{m}_i$) is updated. Then, depending on the direction of the message (forward or backward), the *next* message is added to the queue. The process continues until the $2n$ messages have been computed.

3 Utilizing Value Sparsity

In real-world applications, many of the variables, post-inference, have a low-entropy distribution. That is, the marginal is often peaked at a single value. This peaked marginal may arise during the early steps of inference, and subsequent steps either reinforce the peaky distribution or do not have a significant impact on it. For example, in a linear chain tagging task, often the local factors provide strong evidence for a single value of the variable that the transition factors do not vary during the course of inference.

If the peaked distribution over a single value is detected during the earlier stages of inference, we can approximate the marginals by replacing the distribution with a *deterministic* distribution that has probability of 1 for the mode of the distribution (and 0 for the others). Making the variable deterministic blocks the information from propagating across it ($\mathbf{m}_i^f \approx \mathbf{m}_i^b \approx \mathbf{m}_i$). The chain is effectively split into two chains at the deterministic variable. Below we describe how we implement such an approach, and how it facilitates of parallelism.

Inducing Value Sparsity: To detect peaked distributions, we recompute the per-variable marginal after every message propagation. We use parameter ζ to decide when to induce the sparsity on a variable, i.e. a variable is treated as a deterministic variable i with a single value l if the probability $\mathbf{m}_i(l) \geq \zeta$. Thus the parameter ζ defines the assumption that if the mode of the marginal has probability $\geq \zeta$, the marginal is not likely to be affected significantly by its neighboring variables.



(a) Exact Parallel Inference (b) Parallel Inference with Induced Sparsity

Figure 1: **Parallel Inference in Chains:** Arrows with the same style represent messages that are independent on each other, and are required to be assigned to the same core. By inducing sparsity on the values of the shaded variables (b), up to 6 cores can be used in parallel, as opposed to the restriction of using only 2 in exact inference (a).

To incorporate this approximation to the exact algorithm outlined in Section 2, we modify it as follows. At any point during inference, the marginal of a variable i may become peaked (according to ζ), in which case we do the following. First, the marginal $\mathbf{m}_i$ (and $\mathbf{m}_i^{f/b}$) is set to the deterministic distribution. Second, we add two messages to the queue: a *forward* pass to variable $i+1$ and a *backward* pass to variable $i-1$. Further, for every message computation, we do not add the *next* messages to the queue if the neighboring variable is deterministic. Note that when $\zeta = 1$, none of the variables become deterministic, and the approach is equivalent to exact inference.

This approach can significantly improve performance even on a single core. Even though the total number of messages is still $2n$, the computation of the messages is much faster for variables that neighbor a deterministic variable. In particular, the summation in Eq 3 (or 4) reduces to a single value since $\mathbf{m}_{i-1}^f$ (or $\mathbf{m}_{i+1}^b$) is non-zero only for a single value.

Parallelism: The computations that are added to the queue are independent of each other, and thus can be computed simultaneously using multiple threads. For exact inference, the queue is initialized with only 2 computations, and each step consumes a single message and adds a single message, thereby limiting the size of the queue to ≤ 2. This results in a speedup of 2 when using 2 cores, and increasing the number of cores does not provide a benefit.

With our approximation, steps that results in a deterministic variable consume a single computation, but potentially add two more computations. The queue can therefore grow to sizes > 2, thus allowing multiple cores can be utilized to provide speedups. Alternatively, we can view the deterministic variables as "islands of certainty" that each split the chain into two. This results in multiple forward and backward passes that are independent of each other. See Figure 1b for an example.

Revisiting Sparsity: Marginals of a variable may vary considerably over the course of inference, and committing to a deterministic value at an early stage of inference can lead to substantial errors. We allow revisiting of our sparsity decisions by storing the actual marginals as a separate value $\mathbf{m}'_i$. After every message propagation, we update $\mathbf{m}'_i$ and compare the resulting distribution probabilities with ζ. If the probability of all the values is $< \zeta$, we *unsparsify* the variable by setting $\mathbf{m}_i$ to $\mathbf{m}'_i$. We also modify the deterministic value if the mode of $\mathbf{m}'_i$ is different from $\mathbf{m}_i$ and the probability of the mode is $\geq \zeta$.

Extension to Tree-shaped Models: Although we describe this work on linear-chain models, the algorithm generalizes to the case of the tree. Exact inference in the tree is defined by selecting a root, and performing message passing in *upstream* and *downstream* passes. However, the queue size is not limited to 2; particularly the beginning of *upstream* pass and the end of *downstream* pass can grow the queue considerably. Thus we can achieve speedups > 2 even for exact inference (see [12]). When including our approximation, we expect further speedups. Specifically, as the message propagate up the tree, the size of the queue shrinks in exact inference. In our approach, we are splitting the tree into multiple smaller tress, thus providing potential for utilizing multiple cores at every stage of inference. However, we do not expect the speedups to be as significant as in chains.

4 Experiments

To study the trade-offs provided by our approach, we run the inference algorithm on synthetic and real-world problems, described in the following sections.

Synthetic Chains: As described above, the speedup provided by multiple cores depends on the size of the queue of computations, i.e. linear speedups can be obtained if the queue size is always greater than the number of cores. However, the queue size doesn't depend on ζ, instead it depends on which variables turn deterministic during inference. To study the behavior of our system as we vary the

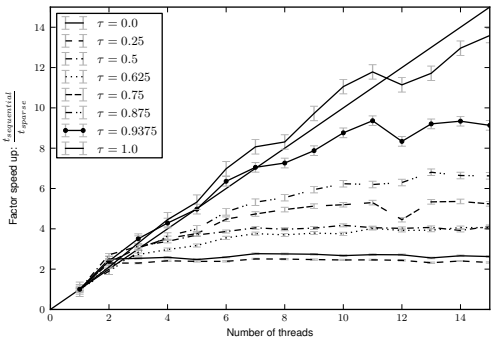
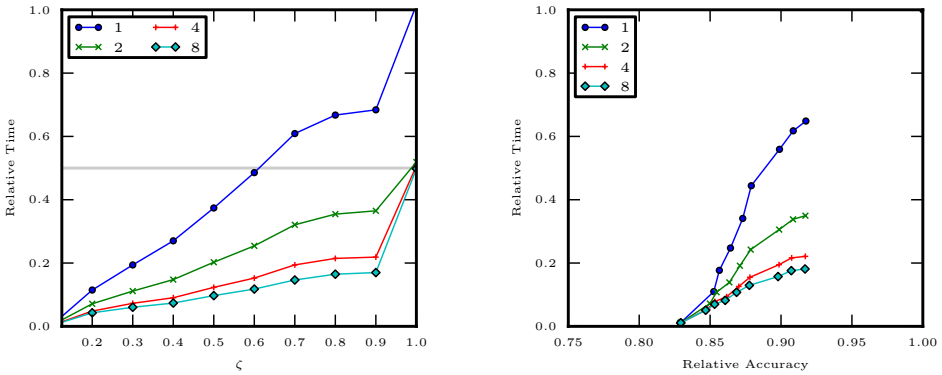


Figure 2: **Synthetic Models:** Speedups obtained when varying the position of deterministic variables from ideal ($\tau = 1$) to the worst ($\tau = 0$).



(a) **Relative Running Time** compared to sequential for different number of cores, as ζ is varied

(b) **Relative Max-marginal accuracy** compared to exact inference for different number of cores, as ζ is varied

Figure 3: **POS-tagging on CONLL data:** Varying ζ to observe the time and accuracy tradeoffs

position of deterministic variables, we create synthetic chains of length 128 with 15 deterministic variables each and random potentials. The position of the deterministic variables within the chain is controlled by τ, where $\tau = 1$ denotes ideal placement (uniformly across the chain) and $\tau = 0$ denotes the worst placement (all deterministic placed together). Figure 2 shows the speedups obtained when using multiple cores (over 100 iterations) and differing τ. We observe near-linear speedups when the placement is close to ideal, and never worse than the speedup obtained using parallel exact inference.

Part-of-speech Tagging: To observe the performance of our approach on real-world dataset, we run inference on a model of part of speech tagging on the CONLL 2003 dataset [13]. A linear chain model with features on the observed word (capitalization, lowercase, etc.), chunk, and context features is created and trained using max-likelihood with exact inference. We plot the performance of our system in Figure 3 when run on 1000 sentences from the test set (and filtered by size < 10) to measure the running time relative to a sequential exact inference implementation[1], and compute the accuracy of the resulting marginals relative to exact inference. Figure 3a shows impressive gains in running time over the sequential and the parallel exact inference (solid gray line) as ζ is varied. These speedups are accompanied by a slight dropoff in accuracy; Figure 3b shows the accuracy our method can achieve for different running times, and number of cores. Note that sequential exact inference lies at $(1.0, 1.0)$ while parallel exact inference lies at $(1.0, 0.5)$.

[1]We do not take the time to propagate the local factors into account as it is the same for both the approaches.

5 Conclusions

In this work we propose an approximate inference algorithm for tree-shaped models that allows trading of accuracy for lower running time on multiple cores. Our approach has a single parameter that controls the loss in accuracy and the gains in speedups. We show results on synthetic and real-world data that demonstrate the resulting trade-offs.

References

[1] Jie Tang, MingCai Hong, Juan-Zi Li, and Bangyong Liang. Tree-structured conditional random fields for semantic annotation. In *International Semantic Web Conference*, pages 640–653, 2006.

[2] Fuchun Peng and Andrew McCallum. Information extraction from research papers using conditional random fields. *Inf. Process. Manage.*, 42(4):963–979, July 2006.

[3] B. Settles. ABNER: An open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[4] Yan Liu, Jaime G. Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. Protein fold recognition using segmentation conditional random fields (scrfs). *Journal of Computational Biology*, 13(2):394–406, 2006.

[5] A. Torralba, K. P. Murphy, and W. T. Freeman. Using the forest to see the trees: exploiting context for visual object detection and localization. *Communications of the ACM*, 53(3):107–114, March 2010.

[6] Xiao Li, Ye-Yi Wang, and Alex Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *International ACM conference on research and development in information retrieval (SIGIR)*, pages 572–579. ACM, 2009.

[7] Yoshimasa Tsuruoka and Jun'ichi Tsujii. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 467–474, 2005.

[8] Libin Shen, Giorgio Satta, and Aravind Joshi. Guided learning for bidirectional sequence classification. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 760–767, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[9] Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *Artificial Intelligence and Statistics (AISTATS)*, 2003.

[10] Joseph Gonzalez, Yucheng Low, and Carlos Guestrin. Residual splash for optimally parallelizing belief propagation. In *Artificial Intelligence and Statistics (AISTATS)*, 2009.

[11] Joseph Gonzalez, Yucheng Low, Carlos Guestrin, and David OHallaron. Distributed parallel inference on large factor graphs. In *Uncertainty in Artificial Intelligence (UAI)*, 2009.

[12] Yinglong Xia and V.K. Prasanna. Junction tree decomposition for parallel exact inference. In *Parallel and Distributed Processing Symposium*, pages 1 –12, 2008.

[13] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL HLT)*, pages 142–147. Association for Computational Linguistics, 2003.