Speeding up Inference in Statistical Relational Learning by Clustering Similar Query Literals

Lilyana Mihalkova*[1] and Matthew Richardson[2]

[1] The University of Texas at Austin, lilyanam@cs.utexas.edu
[2] Microsoft Research, mattri@microsoft.com

Abstract. Markov logic networks (MLNs) have been successfully applied to several challenging problems by taking a "programming language" approach where a set of formulas is hand-coded and weights are learned from data. Because inference plays an important role in this process, "programming" with an MLN would be significantly facilitated by speeding up inference. We present a new meta-inference algorithm that exploits the repeated structure frequently present in relational domains to speed up existing inference techniques. Our approach first clusters the query literals and then performs full inference for only one representative from each cluster. The clustering step incurs only a one-time up-front cost when weights are learned over a fixed structure.

1 Introduction

Markov logic networks (MLNs) [13] represent knowledge as a set of weighted first-order clauses and have been successfully applied to a variety of challenging tasks, such as information extraction [12], and ontology refinement [19], among others. In these applications, MLNs are treated as a "programming language" where a human manually codes a set of formulas, for which weights are learned from the data. This strategy takes advantage of the relative strengths of humans and computers: human experts understand the structure of a domain but are known to be poor at estimating probabilities. By having the human experts define the domain, and the computer train the model empirically from data, MLNs can take advantage of both sets of skills.

Nevertheless, producing an effective set of MLN clauses is not foolproof and involves several trial-and-error steps, such as determining an appropriate data representation and tuning the parameters of the weight learner. Inference features prominently throughout this process. It is used not only to test and use the final model, but also multiple rounds of inference are performed by many popular weight learners [6]. Therefore, just as the availability of fast compilers significantly simplifies software development, "programming" with an MLN would be facilitated by speeding up inference.

This paper presents a novel meta-inference approach that can speed up any available inference algorithm B by first clustering the query literals based on the evidence that affects their probability of being true. Inference is performed using B for a single representative of each cluster, and the inferred probability of this representative is assigned to all other cluster members. In the restricted case, when clauses in the MLN each contain at most one unknown literal, our approach returns the same probability estimates as performing complete inference using B, modulo random variation of B. We call our approach BAM for Break And Match inference.

* A significant portion of this work was completed at Microsoft Research.

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

2 Background on MLNs

An MLN [13] consists of a set of weighted first-order clauses. Let $\mathbf{X}$ be the set of all propositions describing a world, $\mathbf{Q}$ be a set of query atoms, and $\mathbf{E}$ be a set of evidence atoms. Without loss of generality, we assume that $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. Further, let $\mathcal{F}$ be the set of all clauses in the MLN, w_i be the weight of clause f_i, and $n_i(\mathbf{q}, \mathbf{e})$ be the number of true groundings of f_i on truth assignment $(\mathbf{q}, \mathbf{e})$. The probability that the atoms in $\mathbf{Q}$ have a particular truth assignment, given as evidence the values of atoms in $\mathbf{E}$ is $P(\mathbf{Q} = \mathbf{q} | \mathbf{E} = \mathbf{e}) = \frac{1}{Z} \exp\left(\sum_{f_i \in \mathcal{F}} w_i n_i(\mathbf{q}, \mathbf{e})\right)$. Ground clauses satisfied by the evidence $\mathbf{E}$ do not affect the probability. Thus, a ground clause G containing atoms from $\mathbf{E}$ falls in one of two categories: **(A)** G is satisfied by the evidence and can be ignored, or **(B)** all literals from $\mathbf{E}$ that appear in G are false and G can be simplified by removing these literals.

In its most basic form, inference over an MLN is performed by first grounding it out into a Markov network (MN) [9], as described by Richardson and Domingos [13]. Although several approaches to making this process more efficient have been developed (e.g. [17], which reduces the memory requirement, and [16], which speeds up the process of grounding the MLN), this basic approach is most useful to understanding BAM. Given a set of constants, the ground MN of an MLN is formed by including a node for each ground atom and forming a clique over any set of nodes that appear together in a ground clause. Inference over the ground MN is intractable in general, so MCMC approaches have been introduced. We use MC-SAT as the base inference procedure because it has been demonstrated to be faster and more accurate than other methods [11]. However, BAM is independent of the base inference algorithm.

3 Speeding Up Inference using BAM

We first describe BAM in the case where each of the clauses in the MLN contains at most one unknown literal. This case, which we call *restricted*, arises in several applications, such as when modelling the function of independent chemical compounds whose molecules have relational structure [2]. In the restricted case, the ground MN constructed from the given MLN consists of a set of disconnected query nodes. Thus the probability of each query literal $Q \in \mathbf{Q}$ being true can be computed independently of the rest and depends only on the number of groundings of each MLN clause that contain Q and fall in category **(B)** described in Sect. 2. This probability is given by: $P(Q = q | E = e) = \frac{\exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(q)\right)}{\exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(0)\right) + \exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(1)\right)}$, where $n_{i,Q}(q)$ is the number of groundings of clause i that contain Q and are true when setting $Q = q$. In the restricted case, these counts constitute the *query signature* of a literal, i.e., the signature for a literal Q consists of a set of (C_i, n_i) pairs where, for each clause C_i, n_i is the number of groundings of C_i containing Q that are not satisfied by the evidence. Literals with the same query signature have the same probability of being true. We can therefore partition all literals from $\mathbf{Q}$ into clusters of literals with identical signatures. The probability of only one representative from each cluster is calculated and can be assigned to all other members of the cluster. This is formalized in Alg. 1.

The algorithm in the general case differs only in the way query signatures are computed. The intuition behind our approach is that the influence a node has on the query node diminishes as we go further away from it. So, by going enough steps away, we can

Algorithm 1 Break and Match Inference (BAM)

1: **Q**: set of ground query literals, B: Base inference algorithm
2: **for each** $Q \in$ **Q do**
3: SIG_Q = calculateQuerySignature(Q,0) (Alg. 2 in general case)
4: Partition **Q** into clusters of queries with identical signatures.
5: **for each** Cluster K found above **do**
6: Pick an arbitrary query literal from K as the representative R
7: Calculate $P(R = true)$ using B on the portion of the MN used to calculate SIG_R.
8: **for each** $Q \in K$ **do**
9: Set $P(Q = true) = P(R = true)$

Algorithm 2 `calculateQuerySignature`$(\mathtt{Q}, \mathtt{d})$ (General case)

1: Input: Q, query literal whose signature is being computed
2: Input: d, depth of literal
3: **if** $d == maxDepth$ **then**
4: Return value (0 or 1) assigned to Q by MaxWalkSat
5: **for each** Grounding G of a clause in the MLN that contains Q **do**
6: **for each** Unknown literal U in G whose signature is not yet computed, $U \neq Q$ **do**
7: $SIG_U =$`calculateQuerySignature`$(\mathtt{U}, \mathtt{d} + \mathtt{1})$
8: **for each** Unground clause C in the MLN **do**
9: **for each** Distinct way a of assigning signature identifiers to the *other* unknown literals in groundings of C that contain Q **do**
10: Include a triple C, a, n in the signature where n is the number of times the particular assignment a was observed
11: Return the unique identifier for the signature

perform a simple approximation to the value of the distant nodes, while only slightly affecting the accuracy of inference. The signature of each node is computed using a recursive procedure based on the signatures of the nodes adjacent to it. The probability that a node Q is true depends on the probabilities that its adjacent nodes are true. The adjacent nodes are those with which Q participates in common clauses. The probability that each adjacent node is true, on the other hand, depends on the probabilities that *its* adjacent nodes are true and so on. In this way, BAM expands into the ground MN until it reaches a pre-defined depth $maxDepth$. We used $maxDepth = 2$ in the experiments. At this point, it cuts off the expansion by assigning to the outermost nodes their most likely values found using the MaxWalkSat algorithm [4]. If Q is selected as a cluster representative in Alg. 1, inference to determine its probability of being true will be carried out only over this portion of the MN (line 7 of Alg. 1). Alg. 2 formalizes the query signature calculation process. Rather than returning the signature itself, Alg. 2 returns a unique identifier associated with each unique signature. In this way, the clustering of nodes occurs alongside the calculation of their signatures, and signatures can be efficiently compared once their identifiers are determined. The identifiers of the outermost nodes whose values are set using MaxWalkSat are 1 (0) for true (false) assignments.

When BAM is used for weight-learning, all signatures can be computed up-front because the signature does not depend on clause weights. In this case, MaxWalkSat cannot be employed because it needs the clause weights. A simple solution is to assign arbitrary values to the outer-most nodes. We experimented with setting the values of all

outer-most nodes to `false` and observed that the accuracy of inference degrades only slightly. These experiments, omitted for space, will appear in the long version.

The running time of BAM may suffer because inference may be performed several times for the same query literal. This happens because the portion of the MN over which we perform inference in order to compute the probability of the cluster representative R contains additional literals that may themselves be chosen as representatives or may appear in the MNs of multiple representatives. To address this problem, we modified the algorithm to perform inference for more than one representative at a time: suppose we would like to perform inference for literal $L1$ from cluster $C1$, but this would involve inference over literal $L2$ from cluster $C2$. If $C2$ is not yet represented, we include in the MN all the literals up to the desired depth necessary for inference over $L2$ as well. If a cluster is already represented, further representatives are not considered.

4 Experimental Set-Up and Results

We implemented BAM within Alchemy [5] and used the implementation of MC-SAT provided with it. MC-SAT was run as a stand-alone inference algorithm and as the base inference algorithm of BAM. In both cases, the same parameter settings were used: all of Alchemy's defaults were kept, except that the number of sampling steps was set to $10,000$ in order to decrease the amount of variation due to sampling and to better simulate a scenario in which BAM is used in the loop of weight-learning. We compared the systems in terms of inference time and average conditional log-likelihood (CLL).

To control the size and complexity of the models, we used a heuristic procedure[3] to generate synthetic MLNs and corresponding datasets in which we varied the number objects and clauses and the complexity of the clauses. We considered 2 levels of clause complexity. In type 1, all clauses mention the unknown (target) predicate just once. In type 2, half of the clauses mention the unknown predicate once, and the rest mention it twice. We considered models that contained 5 or 10 clauses and domains that contained 100, 150, or 200 constants. For each dataset/MLN pair generated above, we performed 5 random runs with each of the two systems, using the same random seed and the same dedicated machine for each system within a run. When performing inference for cluster representatives, BAM executed the same code as that executed by MC-SAT.

We additionally tested BAM on the UW-CSE domain [13][4] using models[5] learned with BUSL, which gave good predictive accuracy on this data set [7]. We used the MLNs from the last point on the learning curve, which were trained on all but one of the available examples. Thus, there are five possible models, one for each of the examples left out for testing. The goal of inference was to predict the `advisedBy` relation.

Fig. 1 summarizes the results on synthetic data. As can be seen from the table in this figure, the difference in the CLL values of MC-SAT and BAM is very small; thus, BAM is able to mirror the quality of probability estimates output by MC-SAT. Moreover, BAM runs consistently faster than MC-SAT, and the improvement in speed increases as the number of constants in the domain grows. On average BAM performed inference over 37% of the query atoms in domains with 100 constants; 42% in domains with 150 constants; and 31% in domains with 200 constants.

[3] The procedure, which models sparsity in relational data, will be described in the long version.

[4] Available from `http://alchemy.cs.washington.edu/` under "Datasets."

[5] Available from `http://www.cs.utexas.edu/~ml/mlns/` under "BUSL."

Exp.	mcsat	bam	Diff.
100,5,1	-0.067	-0.067	-0.000
150,5,1	-0.064	-0.064	0.000
200,5,1	-0.061	-0.061	-0.000
100,5,2	-0.338	-0.316	0.021
150,5,2	-0.344	-0.340	0.004
200,5,2	-0.220	-0.207	0.014
100,10,1	-0.080	-0.080	-0.000
150,10,1	-0.069	-0.069	-0.000
200,10,1	-0.068	-0.068	-0.000
100,10,2	-0.491	-0.489	0.001
150,10,2	-0.598	-0.614	-0.015
200,10,2	-0.668	-0.668	-0.000

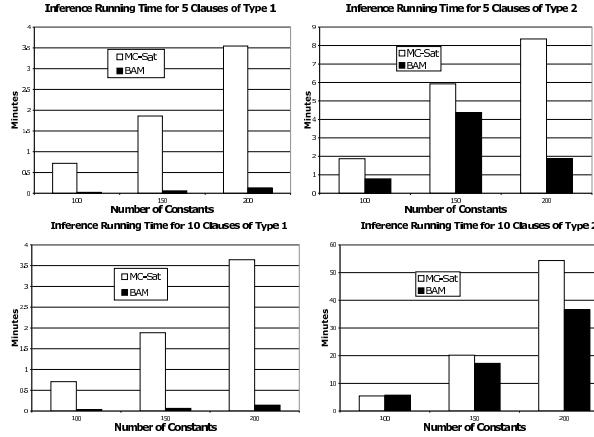

Fig. 1. The table above shows the average CLL of MC-SAT and BAM. The "Exp." column describes the experiment as a (number of objects), (number of clauses), (clause complexity type) tuple. The difference is shown in the last column. A positive (negative) value shows a slight advantage of BAM (MC-SAT). The bar graphs show the avg inference running time in minutes.

Example	MC-SAT	BAM	Diff
1 (AI)	-0.045	-0.045	0.000
2 (Graphics)	-0.044	-0.052	-0.008
3 (Language)	-0.060	-0.060	0.000
4 (Systems)	-0.040	-0.055	-0.015
5 (Theory)	-0.031	-0.031	-0.000

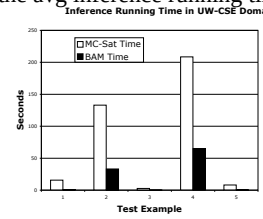

Fig. 2. The table above shows the average CLL of MC-SAT and BAM on each of the test examples, with the difference in CLL shown in the last column. A positive (negative) value shows a slight advantage of (BAM) MC-SAT. The bar graphs show the average inference time in seconds. All times are plotted, although some are extremely small.

Fig. 2 shows the comparison in the UW-CSE domain. The table on the left of this figure shows that MC-SAT and BAM produce probability estimates of similar quality, and the bar plot demonstrates that BAM is consistently faster than MC-SAT. On average, BAM was 12.07 times faster than MC-SAT and performed actual inference only for 6% of the unknown query atoms on average over the five test examples.

In all experiments, the results of inference exhibited very little variance across the random runs. Variance is therefore not reported to reduce clutter.

5 Related And Future Work

BAM is related to work on lifted inference in which variable elimination (VE), aided by clever counting and ordering heuristics, is used to eliminate a large number of instantiations of variables at once [10, 1, 8]. Sen *et al.* [14] introduce an algorithm, based on VE, that constructs a graph whose nodes represent the original and intermediate factors used by VE. By inspecting this graph and carefully computing node labels, factors that carry out identical computations are identified. In a recent extension, [15], to allow for approximate inference, the authors exploit the idea that the influence between two nodes diminishes as the distance between them increases, analogous to the idea exploited in the present work. Jaimovich *et al.* [3] introduce an algorithm based on belief propagation in which inference is performed on the template, i.e. variablized, level.

Their approach targets the case when no evidence is present and has been extended to the case when evidence is present [18]. The approaches based on belief propagation calculate exact probabilities when belief propagation would, but suffer from the same limitations as ordinary belief propagation in the presence of loops. All above techniques are tied to a particular inference approach and are therefore better viewed as stand-alone inference algorithms, in contrast to BAM, which is a meta-inference technique in that it can be applied to any existing inference algorithm.

In the future, we plan to extend BAM to allow "soft" query signature matching, so that signatures need only be very similar to each other to be placed in the same cluster. We would also like to provide a method for quickly recomputing query signatures as the clauses of the MLN are refined, allowing BAM to be used for structure learning.

Acknowledgment: We would like to thank Tuyen Huynh for helpful discussions and the anonymous reviewers for their comments. Some of the experiments were run on the Mastodon Cluster, provided by NSF Grant EIA-0303609, at UT Austin.

References

1. R. de Salvo Braz, E. Amir, and D. Roth. MPE and partial inversion in lifted probabilistic variable elimination. AAAI-06.
2. P. Frasconi and A. Passerini. *Probabilistic Inductive Logic Programming: Theory and Applications*, chapter Learning with Kernels and Logical Representations. Springer, 2008.
3. A. Jaimovich, O. Meshi, and N. Friedman. Template based inference in symmetric relational Markov random fields. UAI-07.
4. H. Kautz, B. Selman, and Y. Jiang. *A General Stochastic Approach to Solving Problems with Hard and Soft Constraints*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.
5. S. Kok, P. Singla, M. Richardson, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, 2005. `http://www.cs.washington.edu/ai/alchemy`.
6. D. Lowd and P. Domingos. Efficient weight learning for Markov logic networks. PKDD-07.
7. L. Mihalkova and R. J. Mooney. Bottom-up learning of Markov logic network structure. ICML-07.
8. B, Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling. Lifted probabilistic inference with counting formulas. AAAI-08.
9. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo,CA, 1988.
10. D. Poole. First-order probabilistic inference. IJCAI-03.
11. H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. AAAI-06.
12. H. Poon and P. Domingos. Joint inference in information extraction. AAAI-07.
13. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
14. P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. VLDB-08.
15. P. Sen, A. Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. UAI-09. To appear.
16. J. Shavlik and S. Natarajan. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. IJCAI-09. To appear.
17. P. Singla and P. Domingos. Memory-efficient inference in relational domains. AAAI-06.
18. P. Singla and P. Domingos. Lifted first-order belief propagation. AAAI-08.
19. F. Wu and D. Weld. Automatically refining the Wikipedia infobox ontology. WWW-08.