

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

Lilyana Mihalkova^{*1} and Matthew Richardson²

¹ Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 USA

`lilyanam@cs.utexas.edu`

² Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
`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) [1] represent knowledge as a set of weighted first-order clauses. Roughly speaking, the higher the weight of a clause, the less likely is a situation in which a grounding of the clause is not satisfied. MLNs have been successfully applied to a variety of challenging tasks, such as information extraction [2], and ontology refinement [3], among others. The general approach followed in these applications, is to treat MLNs 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.

^{*} A significant portion of this work was completed while the author was an intern at Microsoft Research in summer 2007.

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 [4]. Therefore, just as the availability of fast compilers significantly simplifies software development, “programming” with an MLN would be facilitated by speeding up inference. Speeding up inference would also expand the applicability of MLNs by allowing them to be used for modeling in applications that require efficient test-time, or are too large for training to be done in a reasonable amount of time.

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 . Because our new algorithm first breaks down the inference problem to each of the query atoms and then matches, or clusters them, we call it BAM for Break And Match inference.

The remainder of this paper is organized as follows. In Section 2 we introduce necessary background and terminology. We then describe the main contribution of this paper, the BAM algorithm, in Section 3. Section 4 presents our experimental results, showing that BAM maintains the accuracy of the base inference algorithm while achieving large improvements in speed. Finally, we conclude with a discussion of related and future work.

2 Background on MLNs

An MLN [1] 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 $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. If there are atoms with unknown truth values in whose probabilities we are not interested, they can be added to \mathbf{Q} and then simply ignored in the results. 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} cancel from the numerator and Z and do not affect the probability. Thus, a ground clause G containing one or more

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 this case the truth value of G hinges on the assignment made to the remaining literals, which are from the set \mathbf{Q} .

In its most basic form, inference over an MLN is performed by first grounding it out into a Markov network (MN) [5], as described by Richardson and Domingos [1]. Although several approaches to making this process more efficient have been developed (e.g. [6], which reduces the memory requirement, and [7], 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. In the presence of evidence, the MN contains nodes only for the unknown query atoms and cliques only for ground clauses that are not made true by the evidence.

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 [8]. However, BAM is independent of the base inference algorithm used and can in principle be applied to speed up any inference method.

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 modeling the function of independent chemical compounds whose molecules have relational structure [9]. This restricted scenario also arose in work by Wu and Weld [3] on automatic ontology refinement, and, as we will see in Section 4, also in our experimental study.

3.1 BAM in the Restricted Case

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 denominator of the expression, we have set q to 0 and 1 in turn. In the restricted case, these counts constitute the *query signature*

Algorithm 1 Break and Match Inference (BAM)

- 1: \mathbf{Q} : set of ground query literals, B : Base inference algorithm
 - 2: **for each** $Q \in \mathbf{Q}$ **do**
 - 3: $SIG_Q = \text{calculateQuerySignature}(Q, 0)$ (Alg. 2 in general case)
 - 4: Partition \mathbf{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 $P(R = \text{true})$ using B on the portion of the MN used to calculate SIG_R .
 - 8: **for each** $Q \in K$ **do**
 - 9: Set $P(Q = \text{true}) = P(R = \text{true})$
-

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.

3.2 BAM in the General Case

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 perform a simple approximation to the value of the distant nodes, while only slightly affecting the accuracy of inference. The computation begins by first grounding the given MLN to its corresponding Markov network, simplifying it by removing any evidence nodes and any clauses satisfied by the evidence. All nodes in the resulting Markov network are unknown query nodes from the set \mathbf{Q} . 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. Figure 1 provides an illustration. In this figure, we are trying to compute the query signature of the node in the middle that is labeled with q^* . It participates in clauses with four nodes, its immediate neighbors, which are colored in slightly lighter shades in the figure, and thus the probability that is computed for q^* being true will depend on the probabilities computed for these adjacent nodes. 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 [10]. Figure 1 shows these nodes in black, with

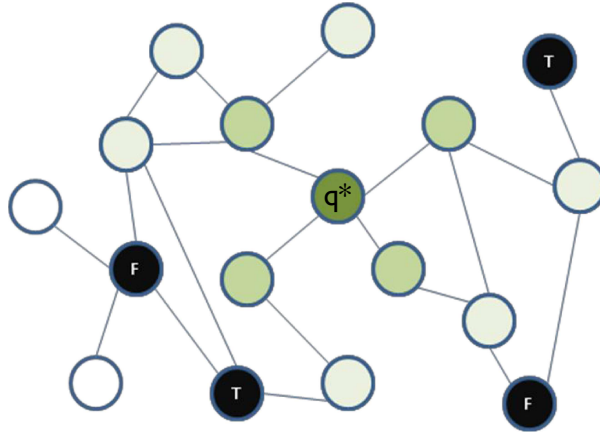


Fig. 1. Illustrating the operation of BAM.

their most likely assignment (**True** or **False**) written inside them. In this way, q^* is rendered conditionally independent from nodes that are further than depth $maxDepth$ from it in the Markov network, given the most likely values of nodes that are exactly at depth $maxDepth$.

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.

3.3 Using BAM in the Loop of Weight Learning

When we use BAM for weight learning, we would like to compute all signatures up-front, rather than having to re-cluster the nodes for each iteration. For this to be possible, the query signature of a node must not depend on the weights of the clauses. Upon inspection of Alg. 2, we notice that the only aspect of this algorithm that depends on the clause weights is in line 4, where we use MaxWalkSat to set the outer-most nodes to their most likely values. A simple solution is to assign arbitrary values to the outer-most nodes instead of using MaxWalkSat. We experimented with setting the values of all outer-most nodes to **False** and observed that the accuracy of inference degrades only slightly. A further discussion of this issue is provided in Sect. 4.

Algorithm 2 calculateQuerySignature(Q, d) (General case)

```
1: Input:  $Q$ , query literal whose signature is being computed
2: Input:  $d$ , depth of literal
3: if  $d == \text{maxDepth}$  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:      $SIG_U = \text{calculateQuerySignature}(U, d + 1)$ 
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
```

3.4 Further Optimizations

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

The goal of our experiments was to answer the following questions:

1. Does BAM produce correct probability estimates?
2. Are the probability estimates output by BAM close to those output by the base inference algorithm?
3. Does BAM provide an advantage in inference time over the base inference algorithm?
4. What is the effect of setting the outer-most nodes to **false** rather than to their most likely values?

There is a subtle difference between the first two questions. The first is concerned solely with BAM's accuracy of inference, whereas the second one focuses on how well BAM approximates the behavior of the base inference algorithm. To answer these questions, we implemented BAM within Alchemy [11] 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. The same parameter settings were used for all systems: 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.

To answer the fourth question, we ran two flavors of BAM, which differed in how the values of the outer-most nodes were set. The original algorithm that uses MaxWalkSat to set the outer nodes to their most likely values is called BAM. The version of the algorithm that simply sets the outer nodes to false is called BAM-False. Note that, barring random variation of the base inference algorithm, those two versions of the algorithm are indistinguishable in the restricted case.

We compared the systems in terms of three measures:

- **CLL** Average conditional log-likelihood, which helps answer the first question above and lends evidence towards answering the second question.
- **MAE** Mean absolute error, measured as the absolute difference between the probability output by BAM and that output by MC-SAT and averaged over all query atoms. This measure provides a direct comparison between BAM and the base inference algorithm.
- **Time** Inference time, which helps answer the third question. To ensure a fair comparison for the timings, all the experiments of the two systems within the same run were performed on the same dedicated machine. Furthermore, when performing inference for cluster representatives, BAM executes the same code as that executed by MC-SAT.

We performed experiments on two sets of data: a synthetically generated one that allowed us to control the complexity and size of the models, as well as the UW-CSE domain [1], which contains information on social interactions in an academic setting.

4.1 Synthetic Data Sets

We first describe how the synthetic data was generated and then discuss the experimental results.

Synthetic Data Generation To generate synthetic MLNs and corresponding datasets, we used a heuristic procedure in which we varied the number of objects and clauses and the complexity of the clauses. This procedure, which we describe next, was designed to model the sparsity typical in relational data. In each case we designated one predicate P_{tar} as the target predicate, all of whose groundings were the query literals. The remaining predicates provided evidence. All predicates were binary. A synthetic dataset and its MLN are specified by three parameters: the number of constants, the number of clauses, and the complexity of the clauses. We considered two levels of clause complexity: restricted (type 1) and non-restricted (type 2). MLNs with restricted complexity contain

Table 1. Possible values characterizing synthetic data.

Model Complexity	Num Clauses	Num Objects
Type 1, Type 2	5,10	100, 150, 200

only clauses that mention P_{tar} once. In non-restricted MLNs, half of the clauses mention P_{tar} once and the other half mention it twice. A dataset/MLN pair was generated as follows. We first randomly assigned truth values to the evidence ground literals. To simulate the fact that relational domains usually contain very few true ground literals, for each evidence predicate/constant pair (P, C) , we drew n_{true} , the number of true literals of P that have C as the first argument, from a geometric distribution with success probability 0.5. We then randomly selected n_{true} constants to serve as second arguments in the true literals. Each clause was of the form (a conjunction of two literals) \Rightarrow (conclusion), where the conjunction in the antecedents contained two evidence literals different from P_{tar} in models with type 1 complexity, and one evidence literal and one literal of P_{tar} in models with type 2 complexity. The conclusion was always a literal of P_{tar} . Truth values were assigned to the ground literals of P_{tar} by first using the clauses of type 1 and then the clauses of type 2 (if applicable). A grounding of P_{tar} was assigned a value of true 90% of the time a grounding of a clause implied it. Finally, to add more noise 1% of the ground literals of P_{tar} had their truth values flipped. We used the default discriminative weight learning procedure in Alchemy [11] to learn weights for the synthetic MLNs. Table 1 summarizes the possible values for the various characteristics of a dataset/MLN pair. In this way, we obtained $2 \times 2 \times 3 = 12$ distinct dataset/MLN pairs.

Results on Synthetic Data For each of the 12 dataset/MLN pairs generated above, we performed 5 random runs with each of the systems (BAM and MC-SAT), using the same random seed for both systems within a run.

Table 2 displays the results in terms of CLL. The “Experiment” column of this table describes the experiment as a (number of objects), (number of clauses), (clause complexity type) tuple. Each “Difference” column gives the difference in CLL between the corresponding version of BAM and MC-SAT. A positive (negative) value shows a slight advantage of BAM(MC-SAT). The results presented in this table allow us to draw three conclusions. First, both systems give good accuracy in terms of CLL, as all values are close to 0. Second, the difference in the CLL scores of MC-SAT and BAM is very small; thus, BAM is able to mirror the quality of probability estimates output by MC-SAT. Third, BAM-False leads to only a very slight degradation, as compared to BAM.

Further evidence supporting the conclusion that BAM’s probability estimates are very close to those of MC-SAT is presented in Table 3. This table displays the mean absolute differences between the probability output by BAM and that

Table 2. Average CLL of MC-SAT, BAM and BAM-False on synthetic data.

Experiment	MC-SAT	BAM	Difference	BAM-False	Difference
100,5,1	-0.067	-0.067	0.000	-0.067	0.000
150,5,1	-0.064	-0.064	-0.000	-0.064	-0.000
200,5,1	-0.061	-0.061	0.000	-0.061	0.000
100,5,2	-0.338	-0.316	-0.021	-0.302	-0.036
150,5,2	-0.344	-0.340	-0.004	-0.333	-0.011
200,5,2	-0.220	-0.207	-0.014	-0.205	-0.016
100,10,1	-0.080	-0.080	0.000	-0.080	0.000
150,10,1	-0.069	-0.069	0.000	-0.069	0.000
200,10,1	-0.068	-0.068	0.000	-0.068	0.000
100,10,2	-0.491	-0.489	-0.001	-0.489	-0.002
150,10,2	-0.598	-0.614	0.015	-0.619	0.020
200,10,2	-0.668	-0.668	0.000	-0.674	0.006

output by MC-SAT, over all test atoms. The fact that these differences are tiny indicates that BAM’s performance is very close to that of its base algorithm.

Moreover, BAM runs consistently faster than MC-SAT, and the improvement in speed increases as the number of constants in the domain grows. This can be observed in Fig. 2, which shows only the running times of BAM. The running time of BAM-False is always at most that of BAM because the potentially time-consuming MaxWalkSat step is omitted. On average, BAM performed inference for 37% of the query atoms in domains with 100 constants; 42% in domains with 150 constants; and 31% in domains with 200 constants.

4.2 UW-CSE Domain

We additionally tested BAM on the UW-CSE domain [1]³ using models⁴ learned with BUSL, which gave good predictive accuracy on this data set [12]. 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. We found that all of these models fall in the restricted case; i.e. they contain at most one mention of the `advisedBy` predicate per clause. This provides an example of a situation in which the restricted case arose naturally in practice. Thus, on this domain the behavior of BAM is identical to that of BAM-False; hence we report results only for the former.

Table 4 compares the CLL scores of MC-SAT and BAM. The “Difference” column provides the difference, where a positive (negative) value shows a slight

³ Available from <http://alchemy.cs.washington.edu/> under “Datasets.”

⁴ Available from <http://www.cs.utexas.edu/~ml/mlns/> under “BUSL.”

Table 3. Mean Absolute Error (MAE) of BAM and BAM-False compared to MC-SAT on synthetic data.

Experiment	BAM	BAM-False
100, 5, 1	0.003	0.003
150, 5, 1	0.002	0.002
200, 5, 1	0.002	0.002
100, 5, 2	0.036	0.041
150, 5, 2	0.020	0.025
200, 5, 2	0.008	0.007
100, 10, 1	0.003	0.003
150, 10, 1	0.003	0.003
200, 10, 1	0.002	0.002
100, 10, 2	0.008	0.009
150, 10, 2	0.021	0.055
200, 10, 2	0.033	0.060

Table 4. The first four columns compare the average CLL of MC-SAT and BAM on each of the test examples in UW-CSE. The last column lists the MAE of BAM compared to MC-SAT.

Example	MC-SAT	BAM	Difference	MAE
1 (AI)	-0.045	-0.045	0.000	0.002
2 (Graphics)	-0.044	-0.052	-0.008	0.004
3 (Language)	-0.060	-0.060	0.000	0.002
4 (Systems)	-0.040	-0.055	-0.015	0.006
5 (Theory)	-0.031	-0.031	0.000	0.001

advantage of BAM (MC-SAT). As on the synthetic data, we see that both systems have very high scores, and that their scores are of similar quality.

The MAE is shown in the last column of Table 4, and the inference time is shown in Fig. 3. From these experiments, we can conclude that MC-SAT and BAM produce probability estimates of similar quality and 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 both the synthetic and UW-CSE experiments, the results of inference exhibited very little variance across the random runs. Variance is therefore not reported to reduce clutter.

5 Related 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

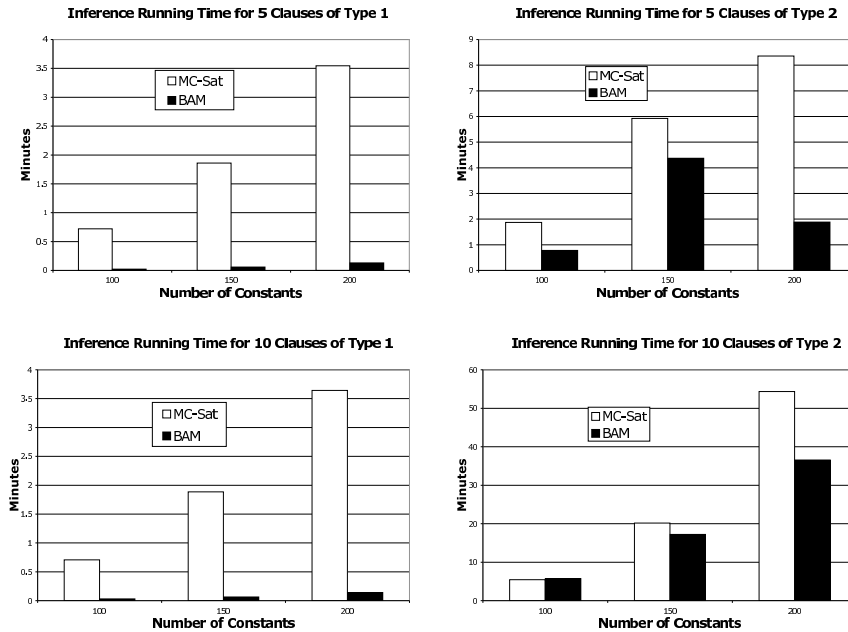


Fig. 2. Average inference running time in minutes.

number of instantiations of variables at once [13–15]. Sen *et al.* [16] 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 [17], 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.* [18] introduce an algorithm based on belief propagation in which inference is performed on the template, i.e. variabilized, level. Their approach targets the case when no evidence is present and takes advantage of the fact that in this case, the same inference steps are carried out for all literals in the model. This algorithm has been extended to the case when evidence is present [19]. Furthermore, a recent extension [20] allows for belief propagation to be carried out in an on-demand fashion, grounding out the model as time allows and maintaining bounds on the probability estimates. 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.

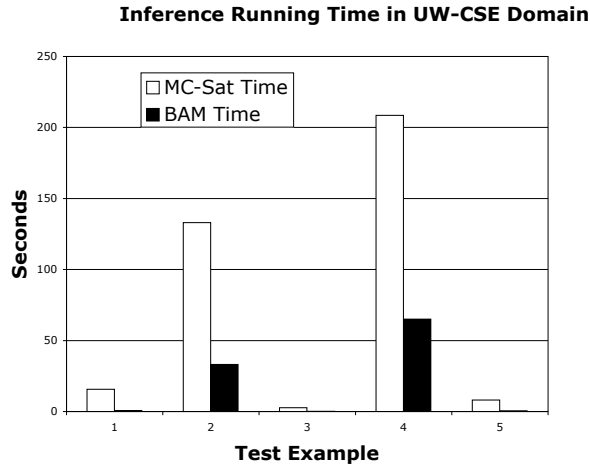


Fig. 3. Average inference time in seconds. All times are plotted, although some are extremely small.

6 Future Work

An interesting future extension to BAM is to allow “soft” query signature matching, so that signatures need only be very similar to each other to be placed in the same cluster. It would also be interesting to provide a method for quickly recomputing query signatures as the clauses of the MLN are refined, added, or deleted, allowing efficient use of BAM for structure learning. Another possible improvement includes developing a lazy version of BAM, analogous to the work of Singla and Domingos [6], that would not require the MLN to be grounded to its corresponding Markov network ahead of time.

Acknowledgment

We would like to thank Raymond Mooney for suggesting the use of mean absolute error; 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. Richardson, M., Domingos, P.: Markov logic networks. *Machine Learning* **62** (2006) 107–136
2. Poon, H., Domingos, P.: Joint inference in information extraction. In: *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, Vancouver, Canada (2007)

3. Wu, F., Weld, D.: Automatically refining the Wikipedia infobox ontology. In: Proceedings of the Seventeenth International World Wide Web Conference (WWW-08), Beijing, China (2008)
4. Lowd, D., Domingos, P.: Efficient weight learning for Markov logic networks. In: Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-07), Warsaw, Poland (2007)
5. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA (1988)
6. Singla, P., Domingos, P.: Memory-efficient inference in relational domains. In: Proceedings of the Twenty-First Conference on Artificial Intelligence (AAAI-06), Boston, MA (2006)
7. Shavlik, J., Natarajan, S.: Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In: Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, CA (2009)
8. Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: Proceedings of the Twenty-First Conference on Artificial Intelligence (AAAI-06), Boston, MA (2006)
9. Frasconi, P., Passerini, A.: Learning with kernels and logical representations. In De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming: Theory and Applications. Springer (2008) 59–61
10. Kautz, H., Selman, B., Jiang, Y.: A general stochastic approach to solving problems with hard and soft constraints. In Gu, D., Du, J., Pardalos, P., eds.: DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Volume 35. American Mathematical Society (1997) 573–586
11. Kok, S., Singla, P., Richardson, M., Domingos, P.: 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>.
12. Mihalkova, L., Mooney, R.J.: Bottom-up learning of Markov logic network structure. In: Proceedings of 24th International Conference on Machine Learning (ICML-07), Corvallis, OR (2007)
13. Poole, D.: First-order probabilistic inference. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico (2003)
14. de Salvo Braz, R., Amir, E., Roth, D.: MPE and partial inversion in lifted probabilistic variable elimination. In: Proceedings of the Twenty-First Conference on Artificial Intelligence (AAAI-06), Boston, Massachusetts (2006)
15. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted probabilistic inference with counting formulas. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI-08). (2008)
16. Sen, P., Deshpande, A., Getoor, L.: Exploiting shared correlations in probabilistic databases. In: Proceedings of the 34th International Conference on Very Large Data Bases (VLDB-08), Auckland, New Zealand (2008)
17. Sen, P., Deshpande, A., Getoor, L.: Bisimulation-based approximate lifted inference. In: Proceedings of 25th Conference on Uncertainty in Artificial Intelligence (UAI-09). (2009)
18. Jaimovich, A., Meshi, O., Friedman, N.: Template based inference in symmetric relational Markov random fields. In: Proceedings of 23d Conference on Uncertainty in Artificial Intelligence (UAI-07), Vancouver, Canada (2007) 191–199

19. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI-08), Chicago, IL (2008)
20. de Salvo Braz, R., Natarajan, S., Bui, H., Shavlik, J., Russell, S.: Anytime lifted belief propagation. In: International Workshop on Statistical Relational Learning (SRL-09), Leuven, Belgium (2009)