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

Lilyana Mihalkova[1]
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
lilyanam@cs.utexas.edu

Matthew Richardson
Microsoft Research
One Microsoft Way
Redmond, WA 98052
mattri@microsoft.com

May 2008
Technical Report
MSR-TR-2008-72

Markov logic networks (MLNs) are a statistical relational learning model that consists of a set of weighted first-order clauses and provides a way of softening first-order logic. Several machine learning problems have been successfully addressed by treating MLNs as a "programming language" where a set of features expressed in first-order logic is manually engineered by the designer and then weights for these features are learned from the data. Inference over the learned model is an important step in this process both because several weight-learning algorithms involve performing inference multiple times during training and because inference is used to evaluate and use the final model. "Programming" with an MLN would therefore be significantly facilitated by speeding up inference, thus providing the ability to quickly observe the performance of new hand-coded features. This paper presents a meta-inference algorithm that can speed up any of the available inference techniques by first clustering the query literals and then performing full inference for only one representative from each cluster. Our approach to clustering the literals does not depend on the weights of the clauses in the model. Thus, when learning weights for a fixed set of clauses, the clustering step incurs only a one-time up-front cost.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

[1]A significant portion of this work was completed while the author was an intern at Microsoft Research in Summer 2007.

1 Introduction

The goal of statistical relational learning (SRL)[1] is to develop representations, as well as learning and inference methods for them, that address the problem of learning probablistic models from multi-relational data. The major challenge associated with learning from multi-relational data is that the entities in the data are usually related in a variety of different ways; as a result, individual training instances cannot be readily broken down into independent components and have varying size.

Markov logic networks (MLNs) [2] are an SRL representation that consists of a set of first-order clauses each of which has an attached weight. Roughly speaking, the higher the weight of a clause, the less likely is a situation in which an instantiation of the clause is not satisfied, over a situation in which it is satisfied. MLNs have recently been successfully applied to a variety of challenging machine learning tasks such as learning for entity resolution [3], for information extraction [4], and for ontology refinement [5]. The general approach followed in these applications is to treat MLNs as a "programming language" where first a human designer manually codes a set of features, and then weights for these features are learned from the data. This strategy is appealing because on the one hand, first-order logic can be intuitively understood by humans and thus can be used by experts from other fields; on the other hand, the clauses that comprise an MLN do not always have to be satisfied and as a result, many, possibly contradictory, "rules of thumb" can be easily encoded to improve the predictive accuracy of the final model. Furthermore, an MLN can be easily coded up using the Alchemy software package [6]. Nevertheless, producing an effective set of MLN clauses is not foolproof and involves several trial-and-error steps, such as determining an appropriate set of predicates to represent the data and tuning the parameters of the weight learner. Just as the availability of fast compilers significantly simplifies software development, "programming" with an MLN would be facilitated by speeding up inference. This is because inference is used not only to test and use the final model, but also multiple rounds of inference are performed by many of the available weight-training techniques [7]. Speeding up inference would also expand the applicability of MLNs by allowing them to be used for modelling in applications that require efficient test-time.

This paper presents a new 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 then performed using B for a single representative of each cluster, and the probability of being true of all other members of a cluster is set to the inferred probability of the cluster representative. In the restricted case when no clause in the MLN contains more than one unknown literal, our approach returns identical probability estimates to performing complete inference using B, modulo random variation of the base inference procedure B. Because our new algorithm first breaks down the inference problem to each of the query atoms and then matches, or clusters them, we will call our approach BAM for Break And Match inference.

2 Terminology and Background

For completeness, we will first review some of the logic terminology used in this paper and will then present necessary background on MLNs.

In first-order logic, a predicate represents a relationship in the domain and can be viewed as a function that returns true or false. Because in this work we assume that the domains contain no logical functions, we define an atom to be a predicate applied to constants or variables. A negative or positive literal is an atom that is or is not negated. An atom is ground if it contains only constants. A clause is a disjunction of literals. A clause is ground if it contains only ground literals. Ground clauses or literals are called groundings.

An MLN [2] consists of a set of first-order clauses, each of which has an attached weight. Let $\mathbf{X}$ be the set of all propositions describing a world (i.e. all ground atoms in the domain), $\mathcal{F}$ be the set of all clauses in the MLN, w_i be the weight of clause f_i, and $\mathcal{G}_{f_i}$ be the set of all possible groundings of clause f_i, and Z be the normalizing partition function. Then, the probability of a particular truth assignment $\mathbf{x}$ to $\mathbf{X}$ is given by the formula:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{f_i \in \mathcal{F}} w_i \sum_{g \in \mathcal{G}_{f_i}} g(\mathbf{x})\right). \tag{1}$$

The value of a particular ground clause $g(\mathbf{x})$ is 1 if the clause is satisfied on truth assignment $\mathbf{x}$ and 0 otherwise. In the above expression, Z is the normalizing partition function given by:

$$Z = \sum_{\mathbf{x}} \exp\left(\sum_{f_i \in \mathcal{F}} w_i \sum_{g \in \mathcal{G}_{f_i}} g(\mathbf{x})\right).$$

Usually, one would like to find the probability that a subset $\mathbf{Q}$ of *query* atoms have a particular truth assignment, given as *evidence* the values of a subset $\mathbf{E}$ of atoms, where $\mathbf{E} \cap \mathbf{Q} = \emptyset$. For simplicity, let us assume that $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. In this case, equation 1 can be rewritten as:

$$P(\mathbf{Q} = \mathbf{q} | \mathbf{E} = \mathbf{e}) = \frac{1}{Z} \exp\left(\sum_{f_i \in \mathcal{F}} w_i \sum_{g \in \mathcal{G}_{f_i}} g(\mathbf{q}, \mathbf{e})\right). \tag{2}$$

Here, the normalizing function is computed by holding fixed the values assigned to the atoms in $\mathbf{E}$ and summing over possible assignments to the literals in $\mathbf{Q}$.

As can be seen from this formula, ground clauses that are satisfied by the evidence, i.e. by the truth assignments to atoms in the set $\mathbf{E}$, cancel from the numerator and denominator and therefore do not contribute to the probability calculation. Thus, each ground clause G that contains one or more atoms from the set $\mathbf{E}$ falls in one of two categories. Either, (1) G is satisfied by the evidence and can therefore be ignored, or (2) 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 [8], as described by Richardson and Domingos [2]. Although several approaches to making this process more tractable have been developed (e.g. [9]), this basic approach will be most useful to understanding BAM. Given a set of constants, the ground Markov network 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, as discussed in the previous paragraph, the Markov network 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 Markov network is intractable in general. Thus approximation approaches, such as Markov chain Monte Carlo (MCMC) algorithms have been introduced [2]. In our experiments, we use MC-SAT as the base inference procedure because it has been demonstrated to give more accurate probability estimates than other methods [10]. However, BAM can be applied to speed up any inference method.

3 Speeding Up Inference using BAM

This section describes how inference can be sped up using the BAM algorithm. To aid the exposition, before addressing the general case, we will describe our approach in the restricted scenario where each of the clauses constituting the given MLN contains at most one unknown literal. In other words, this restriction requires that when grounding the given MLN, in every ground clause, all the literals, except at most one, belong to the set $\mathbf{E}$. Although this may seem like an overly restrictive case, in reality it arises in several relational applications, such as, for instance, when modelling the function of independent chemical compounds whose molecules are described in terms of relational structure but whose function is labeled by a single variable, independent of the label of the other molecules [11]. This restricted scenario also arose in recent work by Wu and Weld [5] on automatic ontology refinement where it was shown that an MLN containing clauses complying with our restricted scenario outperformed an SVM that used an identical set of features.

The consequence of this restriction is that the ground Markov network constructed from the given MLN consists of a set of disconnected query nodes, i.e. all cliques are of size 1. Thus the probability of each disconnected query literal $Q \in \mathbf{Q}$ being true can be computed independently of the other query literals and depends only on the number of groundings of each MLN clause that contain Q and fall in the second category of clauses discussed in Section 2. More precisely, this probability is given by the expression:

$$P(Q = q|E = e) = \frac{\exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(q)\right)}{\exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(0)\right) + \exp\left(\sum_{f_i \in \mathcal{F}} w_i \cdot n_{i,Q}(1)\right)} \quad (3)$$

In the above expression $n_{i,Q}(q)$ is the number of ground clauses from the second category that contain Q and are true when setting $Q = q$. We will call these counts the *query signature* of a query literal. Any two literals with the same query signature have

the same probability of being true. We can therefore partition all literals from the set **Q** into clusters of literals with equal probability of being true. The actual probability of only one representative from each cluster needs to be calculated and can then be simply assigned to all other members of the cluster. This is, in fact, the approach taken by BAM, which we describe in detail in Algorithm 1. In the restricted case the query

Algorithm 1 Break and Match Inference (BAM)

1: **Q**: set of query literals in ground Markov network
2: B: Base inference algorithm
3: **for each** $Q \in$ **Q do**
4: SIG_Q = calculateQuerySignature(Q) (Algorithm 2)
5: **end for**
6: Partition **Q** into clusters of queries with identical signatures.
7: **for each** Cluster C found above **do**
8: Randomly pick a representative query literal R
9: Calculate $P(R = true)$ using algorithm B on the subset of the Markov network used to calculate SIG_R.
10: **for each** $Q \in C$ **do**
11: Set $P(Q = true) = P(R = true)$
12: **end for**
13: **end for**

signature calculation in line 4 is simple (Algorithm 2).

Algorithm 2 `calculateQuerySignature(Q)`

1: Input: Q, current query literal
2: **for each** Unground clause c in the MLN **do**
3: n = number of groundings of c containing Q that are not satisfied by the evidence.
4: Include a pair c, n in the signature
5: **end for**

When BAM is used as part of weight-learning, all query signatures can be computed up-front because the signature of a node does not change when the weight of a clause is adjusted. This is indeed the reason why we avoid further simplifying the computation in equation 3 by summing the weights of all ground clauses that become identical after the evidence is taken into consideration.

Observation 1 *In the restricted case when each clause in the MLN contains at most one unknown (query) predicate, the results obtained by* BAM *are identical to those obtained by the base inference algorithm B up to random variations of B.*

The proof of this observation follows directly from equation 3. Any two query literals Q and Q' with identical signatures will have equal counts $n_{i,Q}$ and $n_{i,Q'}$ for all clauses f_i in the MLN, and therefore their probability of being true will be the same. Note that this probability can be computed exactly in closed form. Because this

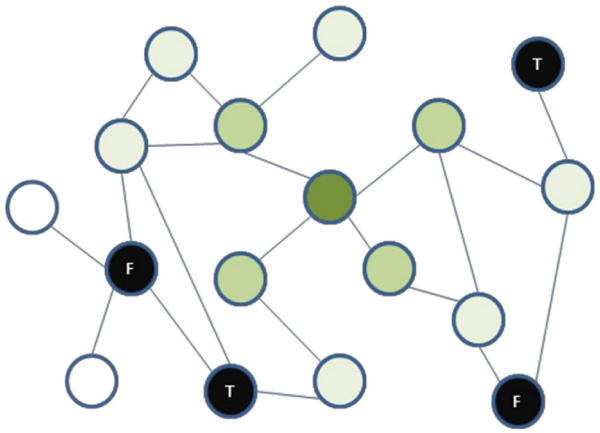

Figure 1: Computing query signature in the genral case. The darkest node in the middle is the query literal Q whose signature is being computed. The black nodes have their values set to true or false using MaxWalkSat [12]. The two white nodes attached to the left-most black "F" node are unknown but they are no longer considered in the computation.

calculation is very efficient, using BAM in the restricted case is most beneficial when inference is performed as part of weight learning.

3.1 BAM in the General Case

BAM would be most useful if it also applies to the general case in which the clauses in the MLN are not restricted to any particular form. It is frequently useful to include clauses that relate the values of the unknown query literals to one another and thus contain more than one unknown literal when ground. For example, Wu and Weld [5] found that including such clauses gave further performance advantages.

The algorithm in the general case differs only in the way query signatures are computed. This process 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, as we discussed in Section 2. 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. As an illustration, consider Fig. 1. In this figure, we are trying to compute the query signature of the node in the middle that is colored in the darkest shade. Let us call this node q^*. It participates in clauses with four nodes, 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 Markov network until it reaches a pre-defined depth $maxDepth$. At this point, it cuts off the

expansion by assigning to the outermost nodes their most likely values found using the MaxWalkSat algorithm [12]. Figure 1 shows these nodes in black, with 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 Algorithm 1, inference to determine its probability of being true will be carried out only over this portion of the Markov network.

Algorithm 3 describes the query signature calculation process more precisely. For simplicity, rather than returning the signature itself, this algorithm returns a unique identifier associated with each 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. More precisely, once a new signature is computed, it is compared to each unique signature that has already been computed. If it is found to be identical to one of the pre-existing signatures SIG, it is assigned the same identifier as the one assigned to SIG; otherwise, it is assigned a unique identifier. The identifier assigned to nodes at depth $maxDepth$ from q^* is either 1 or 0, depending on whether MaxWalkSat assigns a value of $true$ or $false$ to them (line 4). In lines 6-10, the signatures of nodes adjacent to q^* are recursively computed. Once all nodes adjacent to q^* have their signatures computed, in lines 11-16 the algorithm considers the groundings of each MLN clause that contain q^*. Each such grounding is characterized by the signature identifiers assigned to each of the *other* literals (i.e. other than q^*). The query signature consists, for each MLN clause C, of the number of times a particular way of assigning signature identifiers to the other literals in C is obseved in the ground Markov network. For the general case, Algorithm 1 is modified to call `calculateQuerySignature(Q, 0)` in line 4. If an MLN that complies with the restricted case is provided, Algorithm 3 returns the same result as Algorithm 2 when the value of $maxDepth$ is set to be strictly positive.

3.2 Further Optimization

In the general case, 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 Markov network over which we perform inference in order to compute the probability of the cluster representative R contains additional query literals that may themselves be chosen as cluster representatives or may appear in the Markov networks of multiple cluster representatives. One solution to this problem that we implemented is to perform inference for more than one cluster representative at a time. This is done as follows. Suppose we would like to perform inference for literal $L1$ from cluster $C1$, but this would involve inference over literal $L2$ from cluster $C2$, i.e., $L2$ is closer to $L1$ than $maxDepth$. If we have not yet performed inference for any representative of $C2$, we include in the Markov network all the literals up to the desired depth necessary for inference over $L2$ as well. We do this for all literals included in the network. If a representative of a cluster is already included, further representatives are not considered. Doing this also improves the quality of the approximation obtained by BAM.

Algorithm 3 `calculateQuerySignature(Q, d)` (Computing query signatures in general case)

Input: q^*, query literal whose signature is being computed
Input: d, depth of literal
if $d == maxDepth$ **then**
    Return value (0 or 1) assigned to q^* by MaxWalkSat
end if
for each Grounding G of a clause in the MLN that contains q^* **do**
    for each Unknown literal U in G whose signature is not yet computed, $U \neq q^*$ **do**
        SIG_U =`calculateQuerySignature(U, d + 1)`
    end for
end for
for each Unground clause C in the MLN **do**
    for each Distinct way a of assigning signature identifiers to the *other* unknown literals in a grounding of C that contains q^* **do**
        Include a triple C, a, n in the signature where
        n is the number of times the particular assignment a was observed
    end for
end for
Return the unique identifier for the signature

4 Experiments

The goal of our experiments was to determine the situations in which BAM is likely to be the most effective and to verify that the approximate probability estimates returned are not too far from the ones output by the base inference algorithm.

To be able to control the size and complexity of the MLN over which inference is performed, we used synthetically generated MLNs and corresponding datasets in which we varied the number of objects, the number of clauses, and the complexity of the clauses. As also noted by Poon and Domingos [10], the problem of generating data from a given MLN is difficult; like these authors, we therefore used a heuristic procedure to generate MLNs, as follows. In each case we designated one predicate P_{tar} as the target predicate whose groundings were the query literals, while 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 contained only clauses that mention P_{tar} once. In non-restricted MLNs, half of the clauses mentioned P_{tar} once and the other half mentioned it twice. A dataset/MLN pair was generated as follows. We first randomly assigned truth values to the evidence ground literals. For each evidence predicate/constant pair (P, C), we drew the number of literals of P that have C as the first argument from a geometric distribution with success probability 0.5, thus simulating the fact that relational domains usually contain

very few true ground literals. The necessary number of constants for the second argument were then randomly selected. 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. A grounding of P_{tar} was assigned a value of true 90% of the time a grounding of a type 1 clause implied it. Additional truth assignments were then carried out analogously using clauses of type 2, if applicable. Finally, 1% of the ground literals of P_{tar} had their truth values flipped. We used the default discriminative weight learning procedure in Alchemy [6] to learn weights for the synthetic MLNs. We tested with models containing 5, 10, and 15 clauses and 50, 100, 150, and 200 objects.

We implemented BAM within Alchemy and used the implementation of MC-SAT provided with the package. 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: we kept the defaults of Alchemy, except that we set the number of sampling steps performed to 10000. The goal of this high setting was two-fold. On the one hand, we wanted to minimize the amount of variation due to sampling. On the other hand, we wanted to better simulate a scenario in which BAM is used as part of weight-learning. In this latter case, a very large number of infrence iterations are performed (several for each weight adjustment).

We compared the performance of BAM using MC-SAT as a base algorithm to that of the original MC-SAT in terms of the time it took for inference to complete and the conditional log-likelihood (CLL) of the output predictions. The accuracy results are averages over 5 random runs. To ensure a fair comparison for the timings, for each MLN and each system, one of the 5 runs over which we averaged was performed on the same dedicated machine. Thus the results on running time are based on a single representative run.

4.1 Results

Table 1 shows the difference between the CLL of MC-SAT and the CLL of BAM for each of the models on which we tested. A positive value indicates a slight advantage of MC-SAT, whereas a negative value indicates a slight advantage of BAM. As can be seen, the differences in predictive accuracy are tiny. This demonstrates that the approximation obtained by BAM is comparable to that of MC-SAT. The values in three of the cells are missing because for these models, BAM did not have enough memory to store all the query signatures.

The comparison between the running times is shown in Figures 2-4. As can be seen from these figures, in the cases when BAM manages to store its query signatures in memory, it gives substantial improvements in inference speed. Thus BAM is able to output probability estimates of similar quality faster than the base inference method it uses.

Number of Clauses	Clause Type	Number of Constants	CLL Difference
5	1	50	-0.00044432
5	1	100	0.00025094
5	1	150	-5.326E-05
5	1	200	-0.00012052
5	2	50	-0.0070392
5	2	100	-0.0175504
5	2	150	-0.0032412
5	2	200	
10	1	50	-0.0004864
10	1	100	-0.00016842
10	1	150	0.0002683
10	1	200	-5.6E-07
10	2	50	0.1451082
10	2	100	-0.0015038
10	2	150	0.0105902
10	2	200	-0.0032196
15	1	50	-0.0006408
15	1	100	3.134E-05
15	1	150	-3.454E-05
15	1	200	0.00019144
15	2	50	-0.8244162
15	2	100	-0.0013468
15	2	150	
15	2	200	

Table 1: Differences between the CLL of MC-SAT and that of BAM. Positive numbers indicate an advantage of MC-SAT, whereas negative numbers indicate an advantage of BAM.

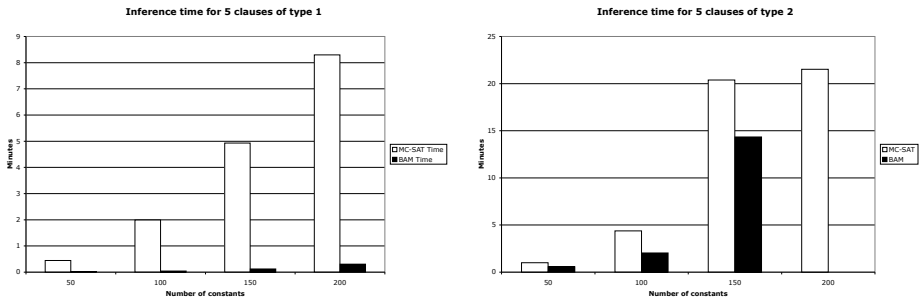

Figure 2: Inference running time in minutes on MLNs containing 5 clauses.

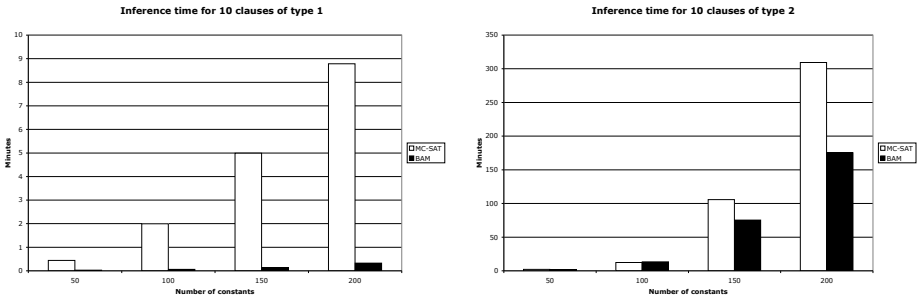

Figure 3: Inference running time in minutes on MLNs containing 10 clauses.

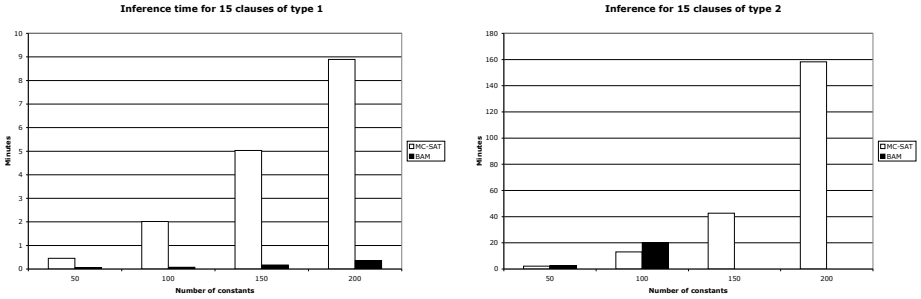

Figure 4: Inference running time in minutes on MLNs containing 15 clauses.

5 Related Work

Several researchers have worked on the problem of speeding up inference. One avenue that has been extensively explored is lifted inference in which variable elimination is used to eliminate all, or a large number of instantiations of variables at once [13, 14, 15]. Jaimovich *et al.* [16] 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 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 recently been extended to the case when evidence is present [17]. All the 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.

Because BAM conditions on the most likely assignments to some of the variables in order to make inference faster, it is also related to recursive conditioning [18]. In this approach, one conditions on variables chosen in such a way, so as to break down the netwok into smaller pieces. Results from inference on the sub-pieces are cached and reused, thus introducing a tradeoff between the time it takes to perform inference and the amount of memory that is used. A related technique is that of Rao-Blackwellised particle filters, which similarly combines conditioning with sampling to improve inference [19].

6 Conclusion and Future Work

This paper presents BAM, a meta-inference algorithm that can speed up any MLN inference technique by first clustering the query literals based on the effect of evidence, and then performing inference for only one representative from each cluster. Our experiments in synthetic domains demonstrate that BAM can successfully decrease inference time while maintaining a level of accuracy similar to that achieved by the base inference algorithm.

In the future, we plan to experiment with real datasets. The applicability of BAM to a variety of problems, particularly ones that are modeled by MLNs containing large numbers of clauses, will be extended if we use "soft" query signature matching techniques. This would allow for larger numbers of queries to be clustered together. Further improvements include developing a lazy version of BAM, analogous to the work of Singla and Domingos [9], that would not require the MLN to be grounded to its corresponding Markov network ahead of time.

Acknowledgment

We would like to thank Tuyen Huynh for helpful discussions regarding the relevance of the restricted scenario and Eyal Amir for pointing out the relationship to Rao-Blackwellised particle filters. Some of the experiments were run on the Mastodon Cluster, provided by NSF Grant EIA-0303609, at The University of Texas at Austin.

References

[1] Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA (2007)

[2] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62** (2006) 107–136

[3] Singla, P., Domingos, P.: Entity resolution with Markov logic. In: Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM-06). (2006)

[4] Poon, H., Domingos, P.: Joint inference in information extraction. In: Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-2007), Vancouver, Canada (2007)

[5] Wu, F., Weld, D.: Automatically refining the Wikipedia infobox ontology. In: Proceedings of the Seventeenth International World Wide Web Conference (WWW-2008), Beijing, China (April 2008)

[6] 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`.

[7] 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)

[8] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo,CA (1988)

[9] Singla, P., Domingos, P.: Memory-efficient inference in relational domains. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI-2008), Boston, MA

[10] Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006), Boston, MA (July 2006)

[11] Frasconi, P., Passerini, A.: Learning with Kernels and Logical Representations. In: Probabilistic Inductive Logic Programming: Theory and Applications. Springer (2008) 59—61

[12] Kautz, H., Selman, B., Jiang, Y. In: 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) 573–586

[13] Poole, D.: First-order probabilistic inference. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003), Acapulco, Mexico (2003)

[14] de Salvo Braz, R., Amir, E., Roth, D.: Lifted first-order probabilistic inference, Edinburgh, Scotland (2005)

[15] de Salvo Braz, R., Amir, E., Roth, D.: MPE and partial inversion in lifted probabilistic variable elimination. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006), Boston, Massachusetts (2006)

[16] 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-2007), Vancouver, Canada (2007) 191–199

[17] Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI-2008). To appear.

[18] Allen, D., Darwiche, A.: New advances in inference by recursive conditioning. In: Proceedings of 19th Conference on Uncertainty in Artificial Intelligence (UAI-2003), San Francisco, CA (2003) 2–10

[19] Doucet, A., de Freitas, N., Murphy, K., Russell, S.: Rao-Blackwellised particle filtering for dynamic bayesian networks. In: Proceedings of 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000). (2000)