Hybrid Markov Logic Networks

Jue Wang **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{*juewang, pedrod*}*@cs.washington.edu*

Abstract

Markov logic networks (MLNs) combine first-order logic and Markov networks, allowing us to handle the complexity and uncertainty of real-world problems in a single consistent framework. However, in MLNs all variables and features are discrete, while most real-world applications also contain continuous ones. In this paper we introduce hybrid MLNs, in which continuous properties (e.g., the distance between two objects) and functions over them can appear as features. Hybrid MLNs have all distributions in the exponential family as special cases (e.g., multivariate Gaussians), and allow much more compact modeling of non-i.i.d. data than propositional representations like hybrid Bayesian networks. We also introduce inference algorithms for hybrid MLNs, by extending the MaxWalkSAT and MC-SAT algorithms to continuous domains. Experiments in a mobile robot mapping domain—involving joint classification, clustering and regression—illustrate the power of hybrid MLNs as a modeling language, and the accuracy and efficiency of the inference algorithms.

Introduction

Probabilistic models typically assume that objects are i.i.d. (independent and identically distributed), but in many domains this is far from the case. Taking object dependencies into account can greatly improve predictive accuracy and yield new insights, but it also greatly increases the complexity of inference and learning. This problem has received much attention in recent years, and many approaches have been proposed (Bakir *et al.* 2007; Getoor & Taskar 2007). Unfortunately, they have focused almost entirely on discrete domains. The state-of-the-art approach for hybrid domains is the hybrid Bayesian Network (Murphy 1998; Lerner & Parr 2001), but it is propositional and has additional limitation on how the continuous and discrete variables could interact. In this paper we extend one of the leading statistical relational learning approaches, Markov logic networks (Richardson & Domingos 2006), to the general hybrid case. MLNs use first-order logic to specify features of Markov networks, allowing us to model complex non-i.i.d. domains very compactly. We call the generalized representation *hybrid Markov logic networks (HMLNs)*.

We develop efficient algorithms for inference in HMLNs, combining ideas from satisfiability testing, slice-sampling MCMC, and numerical optimization. Because most probabilistic models are easily translated into HMLNs, our algorithms are a powerful general-purpose tool for inference in structured domains. Weight learning algorithms are straightforward extensions of existing ones for MLNs. We demonstrate the power of HMLNs and the associated algorithms in a robot mapping domain, where the goal is to identify and locate features of indoor environments (walls, doors, etc.) from laser range data.

Background

Markov Networks

Graphical models compactly represent the joint distribution of a set of variables (or nodes) $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ as a product of non-negative *potential functions* (Pearl 1988): $P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}})$, where each potential ϕ_k is over a subset of the variables $x_{\{k\}}$, and Z is a normalization constant. Under appropriate restrictions, the model is a *Bayesian network* and $Z = 1$. A *Markov network* or *Markov random field* can have arbitrary potentials. As long as $P(X = x) > 0$ for all x, the distribution can be equivalently represented as a *log-linear model*: $P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(x)\right)$, where the $f_i(x)$ are arbitrary *feature functions*.

Given the observed values of some variables, two key inference problems in probabilistic models are (a) finding the most probable joint state of the unobserved variables (MPE inference), and (b) computing conditional probabilities of unobserved variables (conditional inference). Both of these problems are computationally intractable, and are often solved approximately. Methods for MPE inference include simulated annealing, iterated conditional modes, belief propagation (max-product), and graph cuts. Simulated annealing is probably the most widely used method. It consists of repeatedly proposing a state change, accepting it if the new state x' is more probable than the old one x, and otherwise accepting it with probability $(P(x')/P(x))^{1/T}$, where $T > 0$ decreases over time. Methods for conditional inference include Markov chain Monte Carlo (MCMC), belief propagation (sum-product), variational approximation, and others. The most widely used method is Gibbs sam-

pling, a form of MCMC, which consists simply of sampling each variable in turn given its neighbors, repeating until some termination criterion is met, and estimating probabilities from counts over the sampling run (Gilks *et al.* 1996).

Markov network weights can be learned from data using a variety of methods, including convex optimization of the likelihood or a related function, iterative scaling, and margin maximization. Network structure can also be learned from data, typically by performing a greedy search over conjunctions of variables (Pietra *et al.* 1997).

Logic and Satisfiability

A *first-order knowledge base (KB)* is a set of sentences or formulas in first-order logic (Genesereth & Nilsson 1987). First-order logic allows us to compactly represent complex relational structure.

A central (and NP-complete) problem in logic is that of determining if a KB (usually in clausal form) is *satisfiable*, i.e., if there is an assignment of truth values to ground atoms that makes the KB true. One approach to this problem is stochastic local search, exemplified by the WalkSat solver (Selman *et al.* 1996). Beginning with a random truth assignment, WalkSat repeatedly flips the truth value of either (a) an atom that maximizes the number of satisfied clauses, or (b) a random atom in an unsatisfied clause. The *weighted satisfiability* problem is a variant of satisfiability where each clause has an associated weight, and the goal is to maximize the sum of the weights of satisfied clauses. MaxWalkSat is a direct extension of WalkSat to this problem (Kautz *et al.* 1997).

Numerical Optimization

Our inference algorithms use numerical optimization as a subroutine. Numerical optimization is the problem of finding the inputs to a function that maximize (or minimize) its value. First-order methods like gradient descent are popular, but can be very slow. We use L-BFGS, a state-of-the-art second-order method (Liu & Nocedal 1989). Second-order methods use the Hessian matrix (second-order derivatives) in addition to the gradient. L-BFGS scales to large problems by computing a running approximation of the inverse Hessian instead of storing the full matrix.

Markov Logic Networks

First-order logic is brittle because formulas are hard constraints; violating a single formula has the same effect as violating all. The basic idea in MLNs is to allow soft first-order constraints: when a world violates one formula it becomes less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal. Together with a set of constants, an MLN defines a Markov network with one node per ground atom and one feature per ground clause. The weight of a feature is the weight of the first-order clause that originated it. The probability of a state x in such a network is given by $P(x) = (1/Z) \exp\left(\sum_i w_i f_i(x)\right)$, where Z is a normalization constant, w_i is the weight of the ith clause, $f_i = 1$ if the ith clause is true, and $f_i = 0$ otherwise. An MLN can be viewed as a template for constructing Markov networks. Most commonly used probabilistic models can be succinctly formulated as MLNs, including hidden Markov models, conditional random fields, logistic regression, Bayesian networks with context-specific independence, social network models, Ising and Potts models, Boltzmann machines, etc.

MPE inference in MLNs can be performed using a weighted satisfiability solver like MaxWalkSAT. In principle, conditional inference can be performed using Gibbs sampling or other standard techniques, but in practice these can be extremely slow when weights are large (representing strong dependencies), and break down when weights are infinite (deterministic dependencies). Recently, Poon and Domingos (2006) introduced MC-SAT, which handles determinism and achieves very large speedups by combining MCMC with satisfiability testing. MC-SAT is a slice sampler with one auxiliary variable per clause. Slice samplers alternately sample the state and auxiliary variables. The goal of the auxiliary variables is to decorrelate the state ones, enabling rapid mixing. In state x, MC-SAT samples the auxiliary variable u_k corresponding to clause f_k uniformly from $[0, e^{w_k f_k(x)}]$. It then samples a new state uniformly from the "slice," i.e., the states compatible with the auxiliary vector u. This is done using the SampleSAT algorithm, which achieves near-uniform sampling at near-WalkSAT speeds by mixing WalkSAT and simulated annealing steps (Wei *et al.* 2004). Poon and Domingos showed that MC-SAT satisfies ergodicity and detailed balance. In essence, MC-SAT is fast because it uses a SAT solver to jump between modes, instead of waiting exponential time for the Markov chain to drift from one to the other, as in Gibbs sampling and related methods.

MLN weights can be learned generatively using pseudo-likelihood (Richardson & Domingos 2006) or discriminatively using a variety of techniques (Lowd & Domingos 2007). MLN structure can be learned using a form of inductive logic programming (Kok & Domingos 2005).

Hybrid Markov Logic Networks

Representation

Conceptually, extending MLNs to numeric and hybrid domains is quite straightforward: it suffices to allow numeric properties of objects as nodes, in addition to Boolean ones, and numeric terms as features, in addition to logical formulas. Since the syntax of first-order logic already includes numeric terms, no new constructs are required. A numeric term is a term of numeric type, and a numeric property is a designated function of numeric type. For example, if we are interested in distances between objects as random variables, we can introduce the numeric property $\texttt{Distance}(\texttt{x}, \texttt{y})$.

Definition 1 *A hybrid Markov logic network L is a set of pairs (F_i, w_i), where F_i is a first-order formula or a numeric term, and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one node for each possible grounding with constants in C of each predicate or numeric property appearing in L. The value of a predicate node is 1 if the ground predicate is true, and 0 otherwise. The value of a numeric node is the value of the corresponding ground term.*
2. *$M_{L,C}$ contains one feature for each possible grounding with constants in C of each formula or numeric term F_i in L. The value of a formula feature is 1 if the ground formula is true, and 0 otherwise. The value of a numeric feature is the value of the corresponding ground term. The weight of the feature is the w_i associated with F_i in L.*

Thus an HMLN defines a family of log-linear models of the form $P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i s_i(x)\right)$, where s_i is the sum of the values of all groundings of F_i in x. As in Richardson and Domingos (2006), we assume that the value of every function for every tuple of arguments is known, and thus when grounding an HMLN every functional term can be replaced by its value. For convenience, we allow some extensions of first-order syntax in defining HMLNs:

Infix notation. Numeric terms may appear in infix notation (e.g., $\mathtt{F(x)} + \mathtt{G(x)}$).

Formulas inside terms. Logical formulas may appear within numeric terms as indicator functions. This simplifies defining hybrid features.

Soft equality. $\alpha = \beta$ may be used as a shorthand for $-(\alpha - \beta)^2$, where α and β are arbitrary numeric terms. This makes it possible to state numeric constraints as equations, with an implied Gaussian penalty for diverging from them. If the weight of a formula is w, the standard deviation of the Gaussian is $\sigma = 1/\sqrt{2w}$. A numeric domain can now be modeled simply by writing down the equations that describe it. In our experience, this is the most common use of numeric features in HMLNs.

Soft inequality. $\alpha > t$ may be used as a shorthand for $-\log(1 + e^{a(t-\alpha)})$, and $\alpha < t$ for $-\log(1 + e^{a(\alpha-t)})$, with α an arbitrary numeric term. In other words, the (log) sigmoid function is used to represent soft inequality, with a controlling the degree of softness.

We now develop algorithms for inference in HMLNs by extending the corresponding ones for MLNs. For simplicity, the exposition assumes that all weights are positive. Weights can be learned using the same algorithms as for MLNs, with feature counts generalized to feature sums. Other numeric parameters require a straightforward extension of these algorithms. HMLNs structure learning is a topic for future research.

Inferring the Most Probable State

Our algorithm for MPE inference in HMLNs combines MaxWalkSAT and L-BFGS. We call it hybrid MaxWalkSAT, or HMWS for short. Pseudo-code for it is shown in Algorithm 1, where $\mathbf{x}$ is the current state and $S(\mathbf{x})$ is the current sum of weighted features. Numeric variables are initialized uniformly at random, with user-determined range. At each search step, HMWS randomly chooses an unsatisfied clause or numeric feature c, and performs a random step with probability p and a greedy one otherwise. In random steps, HMWS chooses a variable at random and sets it to the value that maximizes $c(\mathbf{x})$. For a Boolean variable in an unsatisfied clause, this simply means flipping it. For a numeric variable, the maximization is done using L-BFGS (line 13), and HMWS then adds Gaussian noise to it (with user-determined variance). In greedy steps, HMWS first performs a one-dimensional optimization of $S(\mathbf{x})$ as a function of each variable, and chooses the variable change that yields the greatest improvement in $S(\mathbf{x})$. In numeric features, if all one-variable changes fail to improve S, HMWS performs a greedy or exhaustive search over assignments to the Boolean variables, for each assignment maximizing the feature sum as a function of the numeric variables using L-BFGS (line 26). The choice between greedy and exhaustive search is determined by the number of Boolean variables. The result of the numeric search for each setting of the Boolean variables is cached, to avoid redoing the search in the future. As in MaxWalkSAT, this process continues for a predetermined number of steps, and is restarted a predetermined number of times. The best assignment found is returned. When all variables and features are Boolean, HMWS reduces to MaxWalkSAT; when all are numeric, to a series of calls to L-BFGS.

Inferring Conditional Probabilities

We extend MC-SAT to handle numeric variables and features by first extending WalkSAT and SampleSAT. A *Boolean constraint* is a clause. A *numeric constraint* is an inequality of the form $f_k(x) \geq a$. Given a set of constraints over Boolean and numeric variables, hybrid WalkSAT (HWS) attempts to find the assignment of values to the variables that maximizes the number of satisfied constraints. HWS is similar to HMWS with the functions $f_k(x)$ as numeric features, except that in each step maximization of $f_k(x)$ is halted as soon as $f_k(x) \geq a$, and the global objective function is the number of satisfied constraints, not the weighted sum of features.

Hybrid SampleSAT (HSS) generates a uniform sample from the states that satisfy a set of constraints M. It mixes HWS and simulated annealing steps. The energy (negative log probability) of a state for simulated annealing is the number of satisfied constraints. A new candidate state is generated by choosing a variable at random, flipping it if it is Boolean, and adding Gaussian noise with user-determined variance to it if it is numeric.

Hybrid MC-SAT (HMCS) is a slice sampler with one auxiliary variable per feature. In state x, the auxiliary variable u_k corresponding to feature f_k is sampled uniformly from $[0, e^{w_k f_k(x)}]$. For Boolean features, the construction of the constraint set M is the same as in MC-SAT. For numeric feature f_k, the constraint $f_k \geq \log(u_k)/w_k$ is added to M. The constraints in M define the slice, and a new state is sampled from it using HSS. Pseudo-code for HMCS is shown in Algorithm 2, where the first step calls WalkSAT to satisfy all hard (infinite-weight) clauses, and $\mathcal{U}_S$ is the uniform distribution over set S. The proof that HMCS satisfies ergod-

Algorithm 1 Hybrid MaxWalkSAT (*clauses, numeric_terms, weights, max_tries, max_flips*)

$\mathbf{x}^* =$ null
$S(\mathbf{x}^*) = -\infty$
for $i \leftarrow 1$ to *max_tries* **do**
  $\mathbf{x} \leftarrow$ random assignment
  for $j \leftarrow 1$ to *max_flips* **do**
    if $S(\mathbf{x}) > S(\mathbf{x}^*)$ **then**
      $\mathbf{x}^* \leftarrow \mathbf{x}$
      $S(\mathbf{x}^*) \leftarrow S(\mathbf{x})$
    end if
    $c \leftarrow$ a random unsatisfied clause or numeric term
    if uniform(0,1) $< p$ **then**
      select a random variable x appearing in c
      $x \leftarrow \text{argmax}_x c(\mathbf{x})$
      if x is numeric **then**
        $x \leftarrow x +$ Gaussian noise
      end if
    else
      for each variable x_i in c **do**
        $x'_i \leftarrow \text{argmax}_{x_i} S(\mathbf{x})$
        $S(\mathbf{x}'_i) \leftarrow S(\mathbf{x})$ with $x_i \leftarrow x'_i$
      end for
      $x'_f \leftarrow x'_i$ with highest $S(\mathbf{x}'_i)$
      if $S(\mathbf{x}'_f) > S(\mathbf{x})$ or c is a clause **then**
        $x_f \leftarrow x'_f$
      else
        $\mathbf{x}_c \leftarrow \text{argmax}_{\mathbf{x}_c} S(\mathbf{x})$,
          where $\mathbf{x}_c$ is the subset of $\mathbf{x}$ appearing in c
      end if
    end if
  end for
end for
return $\mathbf{x}^*$

Algorithm 2 Hybrid MC-SAT(*clauses, numeric_terms, weights, num_samples*)

$x^{(0)} \leftarrow$ Satisfy(hard *clauses*)
for $i \leftarrow 1$ to *num_samples* **do**
  $M \leftarrow \emptyset$
  for all $c_k \in$ *clauses* satisfied by $x^{(i-1)}$ **do**
    With probability $1 - e^{-w_k}$ add c_k to M
  end for
  for all $c_k \in$ *numeric_terms* **do**
    $u_k \sim \mathcal{U}_{[0, \exp(w_k f_k(x^{(i-1)}))]}$
    add $f_k(x^{(i)}) \geq \log(u_k)/w_k$ to M
  end for
  Sample $x^{(i)} \sim \mathcal{U}_{SAT(M)}$
end for

icity and detailed balance is analogous to that for MC-SAT. In purely discrete domains, HMCS reduces to MC-SAT. In purely continuous ones, it is a new type of slice sampler, using a combination of simulated annealing and numerical optimization to very efficiently sample from the slice.

Experiments

Problem and Data

We tested HMLNs and their algorithms on the problem of mobile robotic map building (Limketkai *et al.* 2005). The goal is to infer the map of an indoor environment from laser range data, obtained by the robot as it moves about the environment. The evidence is a set of range finder segments, defined by the (x, y) coordinates of their endpoints. The output is: (a) a labeling of each segment as Door, Wall, or Other (classification); (b) an assignment of wall segments to the walls they are part of (clustering); and (c) the position of each wall, defined by the (x, y) coordinates of its endpoints (regression). We used the mobile robot sensor data from the Radish robotics data set repository (radish.sourceforge.net). In average, each map consists of 99 segments. For evaluation purposes, the "ground truth" labeling is the one provided in the dataset, the assignment was done manually, and a wall's location was computed by fitting a least-squares regression line to (the endpoints of) the wall's true segments, and projecting the endpoints of the wall's first and last segments onto this line. The datasets are also available in the online appendix to this paper.

HMLN for Mobile Robot Map Building

We now describe the HMLN we developed for this problem. Every segment belongs to exactly one type. Every segment type has a prior probability, represented by a unit clause (e.g., $\texttt{SegType}(\texttt{s}, \texttt{Door})$). Doors and walls also have a typical length and depth, represented by numeric terms (e.g., $\texttt{SegType}(\texttt{s}, \texttt{Door}) \cdot (\texttt{Length}(\texttt{s}) = \texttt{DoorLength})$, which has value 0 if the segment is not a door and $-(\texttt{Length}(\texttt{s}) - \texttt{DoorLength})^2$ otherwise). A segment's depth is defined as the signed perpendicular distance of its midpoint to the nearest wall line. During inference, lines are reestimated from the segments assigned to them, using the formulas below. A number of rules involving depth and angle between a segment and the nearest line identify segments of type Other, whose distribution is more irregular than that of doors and walls.

The type of a segment is predictive of the types of consecutive segments along a wall line:

$$\texttt{SegType}(\texttt{s}, \texttt{t}) \wedge \texttt{Consecutive}(\texttt{s}, \texttt{s}') \Rightarrow \texttt{SegType}(\texttt{s}', \texttt{t}')$$

Two segments are consecutive if they are the closest segments to each other along either direction of the nearest line. In addition, segments that are aligned tend to be of the same type:

$$\texttt{SegType}(\texttt{s}, \texttt{t}) \wedge \texttt{Consecutive}(\texttt{s}, \texttt{s}') \wedge \texttt{Aligned}(\texttt{s}, \texttt{s}') \Rightarrow \texttt{SegType}(\texttt{s}', \texttt{t})$$

Segments are aligned if one is roughly a continuation of the other (i.e., their angle is below some threshold, and so is their perpendicular distance). The rules above perform collective classification of segments into types.

Aligned wall segments are usually part of the same wall line:

$$\texttt{SegType}(\texttt{s}, \texttt{Wall}) \wedge \texttt{SegType}(\texttt{s}', \texttt{Wall}) \wedge \texttt{Aligned}(\texttt{s}, \texttt{s}') \wedge \texttt{PartOf}(\texttt{s}, \texttt{l}) \Rightarrow \texttt{PartOf}(\texttt{s}', \texttt{l})$$

This rule clusters wall segments into wall lines. Notice that it captures long-range dependencies between segments along the same corridor, not just between neighboring segments. A segment is the start of a line if and only if it has no previous aligned segment:[1]

$$\texttt{PartOf}(\texttt{s}, \texttt{l}) \Rightarrow (\neg \texttt{PreviousAligned}(\texttt{s}) \Leftrightarrow \texttt{StartLine}(\texttt{s}, \texttt{l}))$$

If a segment is the start of a line, their initial points are (about) the same:

$$\texttt{StartLine}(\texttt{s}, \texttt{l}) \cdot (\texttt{x}_\texttt{i}(\texttt{s}) = \texttt{x}_\texttt{i}(\texttt{l}))$$

where $\texttt{x}_\texttt{i}(\texttt{s})$ is the x coordinate of s's initial point, etc. If a segment belongs to a line, its slope should be the same as the slope of a line connecting their initial points. Multiplying by the Δx's to avoid singularities, we obtain:

$$\texttt{PartOf}(\texttt{s}, \texttt{l}) \cdot [(\texttt{y}_\texttt{f}(\texttt{s}) - \texttt{y}_\texttt{i}(\texttt{s}))(\texttt{x}_\texttt{i}(\texttt{s}) - \texttt{x}_\texttt{i}(\texttt{l})) = (\texttt{y}_\texttt{i}(\texttt{s}) - \texttt{y}_\texttt{i}(\texttt{l}))(\texttt{x}_\texttt{f}(\texttt{s}) - \texttt{x}_\texttt{i}(\texttt{s}))]$$

where the subscript f denotes final points. Line initial points are handled similarly. These rules infer the locations of the wall lines from the segments assigned to them. They also influence the clustering and labeling of segments (e.g., if a better line fit is obtained by relabeling a segment from Wall to Door and thus excluding it, this will be inferred). Classification, clustering and regression are fully joint. In the next section we empirically test the utility of this approach. The full HMLN, containing some additional formulas, is available in the online appendix (file robotmap.mln).

Experimental Methodology

We implemented HMLNs and their algorithms as extensions of the Alchemy system (alchemy.cs.washington.edu). We learned HMLN weights using Alchemy's voted perceptron with HMWS for inference, 100 steps of gradient descent, and a learning rate of 1.0. To combat ill-conditioning, we divided each formula/term's learning rate by the absolute sum of its values in the data (Lowd & Domingos 2007). Other parameters were learned by fitting Gaussian distributions to the relevant variables. We used leave-one-map-out cross-validation throughout. In all inferences, the correct number of walls was given. We evaluated inference results for discrete variables (SegType and PartOf) using F1 score (harmonic mean of precision and recall over all groundings). We used mean square error (MSE) for continuous variables ($\texttt{x}_\texttt{i}(\texttt{l})$, $\texttt{y}_\texttt{i}(\texttt{l})$, $\texttt{x}_\texttt{f}(\texttt{l})$ and $\texttt{y}_\texttt{f}(\texttt{l})$). We also computed the negative log likelihoods of the test values. To obtain density functions for continuous variables from the output of MCMC, we placed a Gaussian kernel at each sample, with a standard deviation of $3r/n$, where r is the sampled range of the variable and n is the number of samples. Details of the experimental procedure, parameter settings and results are included in the online appendix.

[1] By convention, a point precedes another if it has lower x coordinate and, if they are the same, lower y. Segments and lines are ordered by their start points, which precede their final points.

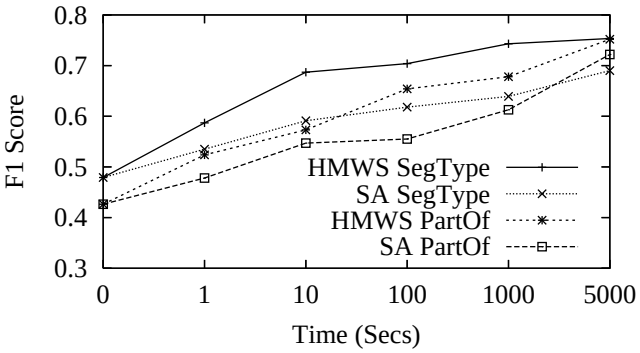

Figure 1: F1 scores for MPE inference.

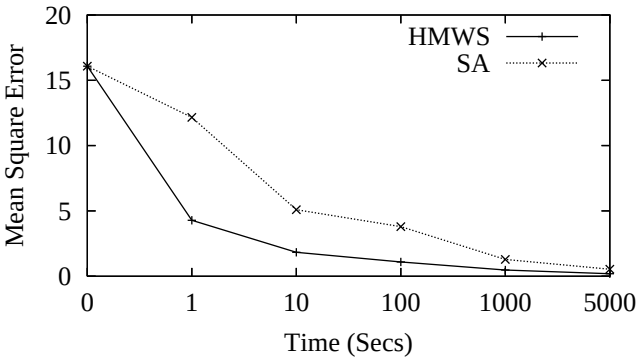

Figure 2: Mean square error for MPE inference.

Inferring the Most Probable State

We compared HMWS with simulated annealing, starting from the same initial random state. We tried a range of parameter values for each algorithm, and report the best results (0.06 random move probability for HMWS, initial temperature of 10 and reduction rate of 0.96 for every 100 steps of simulated annealing). The results are shown in Figures 1 and 2. HMWS is roughly two orders of magnitude faster than simulated annealing.

We also compared HMWS, which infers the discrete and continuous variables jointly, with a more traditional pipeline approach, where the segment types and assignments are first inferred using MWS, and the lines are then estimated by least-squares linear regression over the endpoints of the segments assigned to them by MWS. HMWS and MWS were started from the same initial random state and run for 1,000,000 flips; they typically converged within 100,000 flips. The results are shown in the top two rows of Table 1. Joint inference outperforms pipelined inference on both discrete and and numeric variables.

Inferring Conditional Probabilities

We compared HMCS with Gibbs sampling with ten chains, and with simulated tempering (Marinari & Parisi 1992) with three runs of ten swapping chains. We used the best settings for simulated tempering found by Poon and Domingos (2006). In HSS we used a probability of simulated annealing of 0.4, a temperature of 0.5, 100 steps after reaching a solution, and a standard deviation of one tenth of the variable range for the annealing proposal distribution. The results

Table 1: Joint vs. pipeline approaches.

Measure	SegType F1	PartOf F1	MSE
HMWS	**0.746**	**0.753**	**0.112**
MWS+LR	0.742	0.717	0.198
HMCS	**0.922**	**0.931**	**0.002**
MCS+LR	0.904	0.919	0.037
RMNs	0.899	N/A	N/A

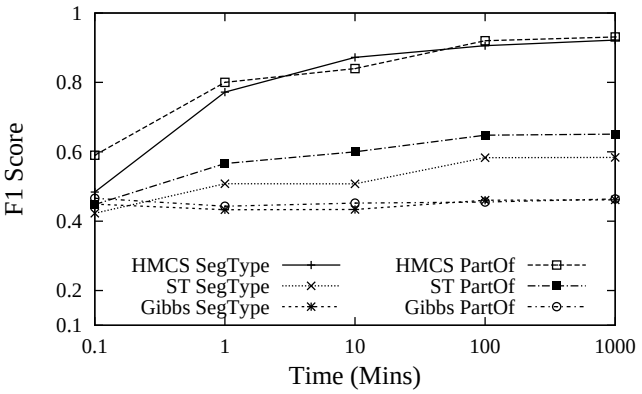

Figure 3: F1 scores for conditional inference.

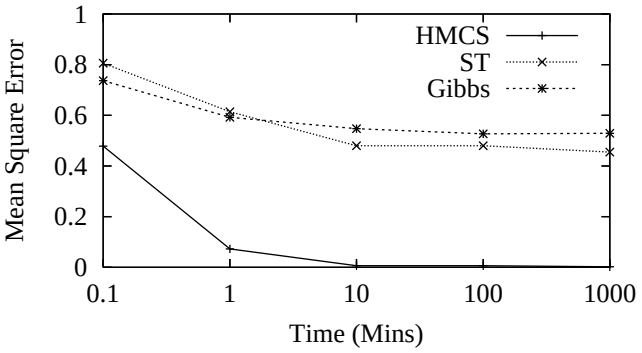

Figure 4: Mean square error for conditional inference.

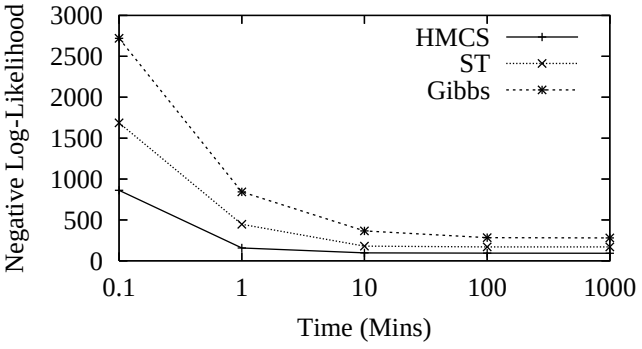

Figure 5: Negative log likelihood of query variables for conditional inference.

are shown in Figures 3, 4 and 5. HMCS greatly outperforms Gibbs sampling and simulated tempering.

We also compared HMCS with a pipeline approach, where the discrete variables are first inferred using MC-SAT, and the lines are then estimated by linear regression over the endpoints of the segments that have probability greater than 0.3 of belonging to them. (There are typically four to six lines in a map). HMCS and MC-SAT were both run for 30,000 steps, with all other settings as above. The results are shown in Table 1 (the bottom three lines). Joint inference performs best, illustrating its benefits. Table 1 also shows the results on classifying SegType of Limketkai et al's result on IJCAI 2005(Limketkai *et al.* 2005), which is based on relational Markov networks. HMLNs perform best, illustrating the benefits of using a more flexible modeling language.

Conclusion

Hybrid Markov logic networks are an elegant language for defining complex non-i.i.d. models in domains with both discrete and numeric features. In this paper we introduced the language and inference algorithms for it, and illustrated their effectiveness on a robot mapping task.

Acknowledgements We are grateful to Fei Wu, Henry Kautz and Dan Weld for helpful discussions. This research was funded by DARPA contracts NBCH-D030010/02-000225, FA8750-07-D-0185, and HR0011-07-C-0060, DARPA grant FA8750-05-2-0283, NSF grant IIS-0534881, and ONR grant N-00014-05-1-0313. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, NSF, ONR, or the United States Government.

References

G. H. Bakir, T. Hofmann, B. Scholkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan (eds.). *Predicting Structured Data*. MIT Press, 2007.

M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

L. Getoor and B. Taskar (eds.). *Introduction to Statistical Relational Learning*. MIT Press, 2007.

W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In *The Satisfiability Problem*. AMS, 1997.

S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proc. ICML-05*.

U. Lerner, R. Parr. Inference in Hybrid Networks: Theoretical Limits and Practical Algorithms. In *Proc. UAI-01*.

B. Limketkai, L. Liao, and D. Fox. Relational object maps for mobile robots. In *Proc. IJCAI-05*.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

D. Lowd and P. Domingos. Efficient weight learning for Markov logic networks. In *Proc. PKDD-07*.

E. Marinari and G. Parisi. Simulated tempering: a new Monte Carlo scheme. *Europhysics Letters*, 19, 451-458, 1992.

K. Murphy. Inference and Learning in Hybrid Bayesian Networks. *University of Berkeley Technical Report*, UCB/CSD-98-990.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

S. D. Pietra, V. D. Pietra, and J. Lafferty. Inducing features of random fields. *Trans. PAMI*, 19, 380-393, 1997.

H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proc. AAAI-06*.

M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62, 107-136, 2006.

B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability*. AMS, 1996.

W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *Proc. AAAI-04*.