Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Counting-MLNs: Learning Relational Structure for Decision Making

Aniruddh Nath
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
nath@cs.washington.edu

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

Abstract

Many first-order probabilistic models can be represented much more compactly using aggregation operations such as *counting*. While traditional statistical relational representations share factors across sets of interchangeable random variables, representations that explicitly model aggregations also exploit interchangeability of random variables *within* factors. This is especially useful in decision making settings, where an agent might need to reason about counts of the different types of objects it interacts with. Previous work on counting formulas in statistical relational representations has mostly focused on the problem of exact inference on an existing model. The problem of learning such models is largely unexplored. In this paper, we introduce *Counting Markov Logic Networks* (C-MLNs), an extension of Markov logic networks that can compactly represent complex counting formulas. We present a structure learning algorithm for C-MLNs; we apply this algorithm to the novel problem of generalizing natural language instructions, and to relational reinforcement learning in the *Crossblock* domain, in which standard MLN learning algorithms fail to find any useful structure. The C-MLN policies learned from natural language instructions are compact and intuitive, and, despite requiring no instructions on test games, win 20% more Crossblock games than a state-of-the-art algorithm for following natural language instructions.

1 Introduction

Statistical relational representations such as Markov logic networks (Richardson and Domingos 2006) and Relational Markov networks (Taskar, Abbeel, and Koller 2002) can compactly represent large graphical models by storing factor *templates* rather than individual factors; to generate the corresponding ground graphical model, these templates are instantiated over a large set of interchangeable random variables, with all instantiations sharing the same structure and parameters. However, dependencies involving a large number of variables can cause the factor templates themselves to become large and difficult to represent. There has recently been interest in exploiting interchangeability *within* factors through the use of *counting* and other aggregations. For example, consider a model of a social network, with a factor that depends on whether or not a person has more than 100 friends. Without an explicit representation of counting, the size of the factor template will depend on the number of people in the network. However, if the representation can reason about counts, the complexity of the representation is independent of the number of objects in the domain.

Milch et al. (2008) incorporated *counting formulas* into FOVE (Poole 2003; de Salvo Braz, Amir, and Roth 2005), an algorithm for exact inference on statistical relational models. The algorithm was further extended by Kisyński and Poole (2009), incorporating other kinds of aggregation. While these algorithms perform inference on counting-based models, the problem of *learning* such models has received less attention. The most relevant work we are aware of is that of Natarajan et al. (2011) on imitation learning in relational domains; their approach learns an ensemble of relational regression trees, which may include aggregations such as counting in their inner nodes. Pasula, Zettlemoyer, and Kaelbling's (2005) work on learning stochastic action models also has some similarities to this line of work.

In this work, we introduce counting formulas to Markov logic networks, a popular statistical relational representation. We describe an algorithm for learning powerful, intuitive counting formulas in Markov logic. As a motivating example, we address the problem of generalizing textual instructions, building on Branavan et al.'s (2009) work on grounded language acquisition. We show that the knowledge gained from textual instructions can be generalized through imitation learning, using the representation and learning algorithm introduced in this work. We also describe a variant of the algorithm that learns autonomously, without the aid of textual instructions.

2 Background

Graphical models compactly represent the joint distribution of a set of variables $\mathbf{X} = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ as a product of factors (Pearl 1988): $P(\mathbf{X}=\mathbf{x}) = \frac{1}{Z}\prod_k \phi_k(\mathbf{x}_k)$, where each factor ϕ_k is a non-negative function of a subset of the variables $\mathbf{x}_k$, 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 factors. Graphical models can also be represented in *log-linear form*: $P(\mathbf{X}=\mathbf{x}) =$

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

$\frac{1}{Z}\exp\left(\sum_i w_i g_i(\mathbf{x})\right)$, where the *features* $g_i(\mathbf{x})$ are arbitrary functions of the state.

Markov logic is a probabilistic extension of first-order logic. Formulas in first-order logic are constructed from logical connectives, predicates, constants, variables and functions. A *grounding* of a predicate (or *ground atom*) is a replacement of all its arguments by constants (or, more generally, ground terms). Similarly, a grounding of a formula is a replacement of all its variables by constants. A *possible world* is an assignment of truth values to all possible groundings of predicates.

A *Markov logic network (MLN)* is a set of weighted first-order formulas. Together with a set of constants, it defines a Markov network with one node per ground atom and one feature per ground formula. The weight of a feature is the weight of the first-order formula that originated it. The probability distribution over possible worlds $\mathbf{x}$ specified by the MLN and constants is thus $P(\mathbf{x}) = \frac{1}{Z}\exp(\sum_i w_i n_i(\mathbf{x}))$, where w_i is the weight of the ith formula and $n_i(\mathbf{x})$ its number of true groundings in $\mathbf{x}$.

2.1 Learning MLNs

The weights of formulas of an MLN can be learned by maximizing the likelihood of a relational database. A natural method of doing this is by gradient ascent; the gradient of the log-likelihood with respect to the weights is:

$$\begin{aligned}\frac{\partial}{\partial w_i}\log P_w(\mathbf{x}) &= n_i(\mathbf{x}) - \sum_{\mathbf{x}'} P_w(\mathbf{x}')n_i(\mathbf{x}') \\ &= n_i(\mathbf{x}) - E_w[n_i(\mathbf{x})]\end{aligned}$$

Calculating this expectation requires inference, which is #P-Complete (Roth 1996). Although there are several efficient approximate inference algorithms for MLNs, they are still too expensive to run in every iteration of gradient ascent. Richardson and Domingos (2006) avoided having to perform inference by optimizing the *pseudo-log-likelihood* (Besag 1975) instead of the true likelihood. Unfortunately, this approximation does not always perform well.

Singla and Domingos (2005) optimized the *conditional* log-likelihood, the gradient of which is:

$$\begin{aligned}\frac{\partial}{\partial w_i}\log P_w(\mathbf{y}|\mathbf{x}) &= n_i(\mathbf{x},\mathbf{y}) - \sum_{\mathbf{y}'} P_w(\mathbf{y}'|\mathbf{x})n_i(\mathbf{x},\mathbf{y}') \\ &= n_i(\mathbf{x},\mathbf{y}) - E_w[n_i(\mathbf{x},\mathbf{y})]\end{aligned}$$

(where $\mathbf{x}$ is an assignment of values to the evidence atoms, and $\mathbf{y}$ is an assignment to the query atoms.)

Since $\mathbf{x}$ is fixed, the inference problem is easier. However, computing the expectations may still be intractable in large domains. Singla and Domingos (2005) approximate the expected count with the count in the MAP state, as in Collins' (2002) *voted perceptron* algorithm. Although this is a better approximation than the pseudo-log-likelihood, it still occasionally fails on complex domains. Lowd and Domingos (2007) describe several more sophisticated weight learning algorithms.

It is also possible to learn the MLN formulas themselves directly from a relational database. MLN structure learning algorithms are similar in spirit to traditional inductive logic programming algorithms, except that they directly optimize the likelihood of the data (or related quantities). The first structure learning algorithm designed specifically for MLNs was MSL (Kok and Domingos 2005). The most widely used version of MSL performs a beam search over the space of first-order clauses (disjunctions). New clauses are generated from existing candidates by adding a new literal to the clause (sharing at least one variable with existing literals), or by flipping the sign of an existing literal. Note that evaluating a candidate MLN requires learning the weights of all its clauses. This is done approximately by optimizing a variant of pseudo-log-likelihood. To combat overfitting, its model also imposes an exponential penalty on clause length, and restricts the maximum number of variables in a single clause.

3 Generalizing natural language instructions

The field of *grounded language acquisition* deals with understanding natural language from its context in the world, rather than learning language in a supervised setting, from labeled training data (Roy 2002; Yu and Ballard 2004; Chen and Mooney 2008). The context often comes in the form of environmental feedback (utility or cost) from an agent's interactions with the world; this setting lends itself to reinforcement learning approaches (Branavan, Zettlemoyer, and Barzilay 2010; Vogel and Jurafsky 2010).

Reinforcement learning has been successfully applied to the problem of mapping natural language instructions to actions with little or no supervision (Branavan et al. 2009). However, this mapping cannot be used to solve new problem instances unless we also receive a set of instructions for the new instance. We present an algorithm for generalizing the knowledge gained while learning to map instructions to actions, allowing us to solve new problem instances with no additional input.

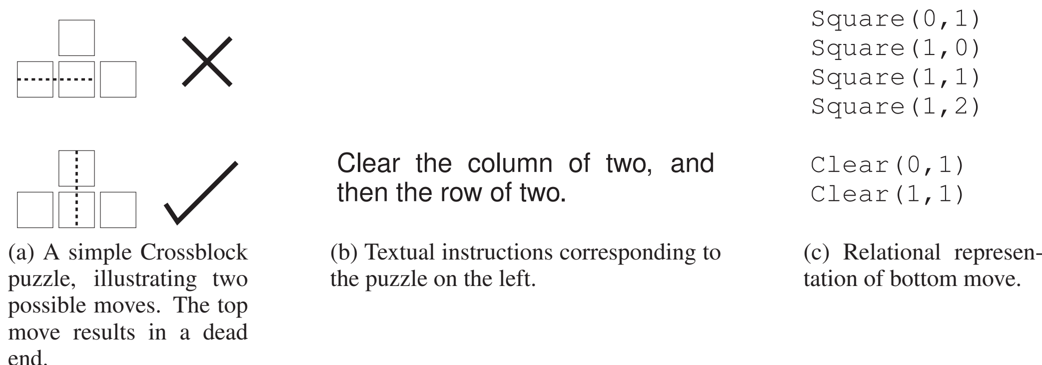
Since our work builds on the algorithm of Branavan et al. (2009), we provide a brief description of their system. Branavan et al.'s algorithm takes as input a set of example problems in a domain, each problem accompanied by natural language instructions describing an optimal solution. For instance, in a game playing domain, the example problems are game instances, and the instructions are walkthroughs, explaining how to win each of the example games. The agent also has access to the domain's reward function; it can execute an action and observe the resulting reward.

The policy is represented as a log-linear model, capturing a joint probability distribution over the actions, world state and instructions:

$$p(a|s,\mathbf{w}) = \frac{\exp(\mathbf{w}.\mathbf{g}(s,a))}{\sum_{a'}\exp(\mathbf{w}.\mathbf{g}(s,a'))}$$

(a is an action; s is the state, including the text of the instructions; $\mathbf{w}$ is a vector of parameters; $\mathbf{g}(s,a)$ is a vector of features.)

The choice of features is domain-dependent; features are chosen to capture correlations between words in the text and actions or environmental properties. The weights of the model are trained using a *policy gradient* algorithm; for



(a) A simple Crossblock puzzle, illustrating two possible moves. The top move results in a dead end.

(b) Textual instructions corresponding to the puzzle on the left.

(c) Relational representation of bottom move.

Figure 1: Example Crossblock puzzle

each of the example problems, the agent repeatedly samples and executes sequences of actions from the current policy, and updates the weights of the features based on the reward. Once the model has been trained, it can solve previously unseen game instances, if accompanied by matching instructional text.

3.1 Example problem: Crossblock

Crossblock[1] is a single-player puzzle game played on a grid. At the start of each level, some of the grid positions are filled with blocks. The player's goal is to clear the grid by drawing horizontal or vertical line segments; a line segment removes all blocks it crosses. Each line segment must cross out some fixed number of blocks, depending on the level. Crossblock was originally released as an online Flash game; the online gaming community created textual walkthroughs[2] to help new players complete the more difficult levels. Figure 1a shows an example Crossblock puzzle, with the accompanying walkthrough.

Crossblock was one of the domains on which Branavan et al. (2009) evaluated their system. The challenging nature of the game and the availability of natural language instructions made it an ideal testbed for their algorithm. We use this domain as a running example for the rest of this paper. (Note that the algorithms we describe are not restricted to Crossblock.)

3.2 Generalizing instructions

Branavan et al.'s algorithm learns a mapping from instructions to actions, allowing it to follow instructions for previously unseen game levels. However, the model cannot be applied to new problem instances unless they are accompanied by textual instructions. If the goal is to create an autonomous agent, the learned knowledge is not applicable.

Learning to play a previously unknown game from scratch is an extremely challenging problem (Genesereth, Love, and Pell 2005), especially in a reinforcement learning setting where the game mechanics may not be known in advance. In domains where textual instructions are available, this information can be exploited to make the learning problem easier. We now describe an algorithm for generalizing the knowledge gained by learning an instruction-following model, allowing the system to solve new problem instances without the aid of any textual instructions at test time.

The algorithm is a form of imitation learning. The instruction-following system acts as the teacher; traces of its actions on the training games form the training examples for the autonomous agent. The goal of the autonomous agent is to imitate the actions of the teacher, without making use of the textual data.

As a concrete example, consider the Crossblock domain. Here, Branavan et al.'s system acts as the teacher. The first step of our algorithm is to run Branavan et al.'s policy gradient algorithm until convergence (or for some fixed number of iterations). At the conclusion of learning, their system follows instructions well enough to successfully complete most of the training games. For these games, given a game state and a set of instructions, their model outputs an optimal action.

The next step of the algorithm is the generalization, which is done via imitation learning. The training data is generated from the traces of the games which Branavan et al.'s agent won. Each training example consists of a game state, along with the action that Branavan et al.'s agent chose during its winning trace. The goal of the learning problem is to predict Branavan et al.'s choice of actions given the game state. Note that the resulting model does not include the textual instructions. This is what allows it to generalize to previously unseen game instances without walkthroughs.

Given this training set, we are left with a traditional inductive learning problem; in principle, any representation and learning algorithm could be chosen. Crossblock training examples can be naturally represented in first-order logic using two predicates (figure 1c):

- `Square`(x, y): indicates the presence of a square at grid location (x, y).
- `Clear`(x, y): indicates that Branavan et al.'s agent chose to cross out grid location (x, y).

To model the policy learned by Branavan et al.'s agent, we need to learn the conditional joint probability distribution of the `Clear`(x, y) atoms (the actions) given the `Square`(x, y) atoms (the state).

[1] http://hexaditidom.deviantart.com/art/Crossblock-108669149
[2] http://jayisgames.com/archives/2009/01/crossblock.php

4 Counting-MLNs

Markov logic networks are a popular representation for probabilistic models of relational domains. Implementations of several MLN learning and inference algorithms are publicly available. The oldest and most widely used MLN software package is ALCHEMY (Kok et al. 2008); it includes an implementation of Kok and Domingos' (2005) original structure learning algorithm for MLNs. Kok and Domingos' (2010) state-of-the-art MLN structure learning algorithm is also available as an open-source package.

We applied both these algorithms to the imitation learning problem described in section 3.2. Neither algorithm succeeded in learning any useful structure in the Crossblock domain. In our preliminary experiments, they either returned blank MLNs, or learned worse-than-random policies by overfitting the training data. Considering the low-level feature representation made available to the learning algorithm, this is not surprising. Successfully applying statistical relational learning algorithms to decision making problems usually requires the design of high-level predicates that allow useful formulas to be compactly represented; this is one of the mechanisms by which background knowledge can be inserted into the learner.

While the ability to incorporate background knowledge in this manner is a useful feature of statistical relational languages, the design of useful features is a labor-intensive process. Ideally, a structure learning algorithm would automatically discover these high-level features with minimal human input. The high-level features for many standard benchmark decision making problems tend to involve counts or other aggregations. For example, Džeroski, De Raedt, and Driessens (2001) introduced high-level predicate $\texttt{NumberOfBlocksOn}(a, n)$ into their formulation of the blocks world problem, allowing them to learn a good policy using relational reinforcement learning. Similarly, Driessens and Džeroski's (2004) formulation of Tetris incorporates high-level predicates such as $\texttt{BlockWidth}(a, n)$, $\texttt{BlockHeight}(a, n)$, $\texttt{HoleDepth}(c, n)$, etc.

Standard Markov logic networks have no way to represent these high-level concepts in terms of the primitive low-level predicates. Unless the high-level predicates are provided by an expert, there is no way for them to be incorporated into the structure learning process. In this section, we describe *Counting Markov Logic Networks* (C-MLNS), an extension of Markov logic designed to capture precisely this kind of regularity.

A *counting expression* is an expression of the form $\#_x[\alpha]$ (similar to the notion of a *counting formula* in Milch et al. 2008). α is a first-order logic formula, and x is a variable. The value of a counting expression is the number of values of x for which α is satisfied. For example, $\#_x[\texttt{Smokes}(x)]$ is the number of smokers in the world. A counting expression can also contain unbound variables; for example, $\#_y[\texttt{Friends}(x, y) \wedge \texttt{Smokes}(y)]$ is the number of smokers that x is friends with. An expression can also count over multiple variables; for example, $\#_{x,y}[\texttt{Friends}(x, y)]$ counts the total number of friendship relations in the world.

A *Counting Markov Logic Network* is an MLN whose formulas include *comparison terms*: equalities or inequalities that include counting expressions (and optionally, numerical constants). For example, the following comparison term asserts that Anna has at least ten more friends than Bob and Charles put together:

$$\#_x[\texttt{Friends}(Anna, x)] \geq \#_x[\texttt{Friends}(Bob, x)] + \#_x[\texttt{Friends}(Charles, x)] + 10$$

The expressions on either side of counting terms can involve arbitrary mathematical operations, such as addition, subtraction, multiplication, etc.

The following C-MLN formula states that people with more smoker than non-smoker friends are more likely to smoke:

$$(\#_y[\texttt{Friends}(x, y) \wedge \texttt{Smokes}(y)] > \#_y[\texttt{Friends}(x, y) \wedge \neg\texttt{Smokes}(y)]) => \texttt{Smokes}(x)$$

As in a traditional MLN, this formula need not be deterministically true; it is associated with a weight w that may be learned from data.

In principle, inference can be carried out by grounding the C-MLN into a propositional Markov network, and running standard exact or approximate inference algorithms. However, factors involving counting expressions are connected to all the atoms involved in the counts. The resulting dense structure is unfavorable to many commonly used algorithms, such as loopy belief propagation (Yedidia, Freeman, and Weiss 2001).

The alternative to propositional inference is *lifted* inference, which avoids grounding out the model when possible, by grouping together sets of objects that are indistinguishable given the available evidence. Milch et al. (2008) introduced an exact lifted inference algorithm incorporating counting formulas. However, since inference in Markov networks is a #P-Complete problem (Roth 1996), exact inference algorithms do not always scale up to realistic problems. Developing scalable approximate algorithms for C-MLNs and related representations is an area for future work.

In this work, we use C-MLNs to represent a policy in a decision making problem. For problems that are fully observable, with a relatively small number of legal actions, exact inference is tractable. The most probable state of the world can be found simply by enumerating over all legal actions, selecting the one with the highest posterior. To calculate the partition function, simply add the potentials resulting from each possible action. The marginal probability of an atom is the sum of the potentials of the worlds where the atom is true, divided by the partition function.

4.1 Learning C-MLNs

Given a tractable inference algorithm (either exact or approximate), C-MLN weights and structure can be learned using variants of the standard MLN learning algorithms described in section 2.1. For weight learning, we adapted Singla and Domingos' (2005) discriminative learning algorithm to C-MLNs. We perform gradient ascent, optimizing the conditional log-likelihood of the training data.

Singla and Domingos approximated the expected number of true groundings of formulas by the counts in the MAP state; instead, we calculate the true counts, since inference is tractable in our setting due to the fully observable state and the small number of legal actions at each step.

For structure learning, we perform a beam search, based on MSL (Kok and Domingos 2005). The introduction of counting expressions expands the already vast search space of MLNs; to make structure learning feasible, we impose some restrictions on the search space:

- Each formula is a disjunction of comparison terms, using the 'greater than or equal to' ($\geq$) operator.
- The quantity on each side of the comparison is the sum of one or more counting expressions.
- The formula in each counting expression is a unit clause.

Although these restrictions lose some of the generality of C-MLNs, the language remains rich enough to capture many interesting patterns. This is analogous to the restriction of most MLN structure learners to only learn disjunctive formulas. Depending on the domain, it may be worthwhile to allow mathematical operations other than addition, to allow numerical constants in the comparison terms, or to allow non-unit clauses in the counting expressions; however, any of these generalizations would make the search problem more challenging.

The beam search itself is identical to MSL. Instead of initializing the search with unit clauses, we initialize with all atomic comparisons (i.e. comparisons with a single counting expression on each side of the comparison operator). We exclude comparisons with the same predicate on both sides. We also require that all counting expressions in a formula count over the same variable.

To generate new candidates from existing clauses, we apply two clause construction operators:

- Add a new atomic comparison to the clause.
- Add a new counting expression to one of the comparisons in the clause.

In each iteration of the beam search, we generate all legal candidates from the formulas in the current beam, subject to the restrictions above. Like MSL, we impose a limit on the total number of distinct variables in a clause, and place a penalty on clause length. We also limit the number of counting expressions in a single term.

We implemented the above C-MLN weight and structure learning algorithms in a system called CHAMELEON, available online[3].

4.2 C-MLNs for Crossblock

Like blocks world and Tetris, Crossblock is a domain where counting-based features make it much more practical to represent a useful policy. For example, a good tactic for Crossblock is to prefer moves that clear an entire row or column. These moves are less likely to leave behind unclearable blocks (figure 1a). Standard MLNs cannot capture this rule in terms of the $\texttt{Square}(x, y)$ and $\texttt{Clear}(x, y)$ predicates, but C-MLNs can capture it quite concisely:

$$\#_y[\texttt{Square}(x, y)] = \#_y[\texttt{Clear}(x, y)]$$

The rule does not always hold; in some situations, the optimal move may not clear any row or column. This is why it is beneficial to learn weights for rules in the policy, rather than enforcing them deterministically.

To summarize, our algorithm for generalizing Crossblock instructions consists of two steps:

1. Train Branavan et al.'s (2009) model using their policy gradient algorithm.
2. Treating their winning game traces as training data, use CHAMELEON to learn the probability distribution over actions given the world state.

5 Relational reinforcement learning

Our algorithm for generalizing natural language instructions uses Branavan et al.'s model to provide training data for an imitation learning system that outputs an autonomous agent capable of solving new instances of the problem. Note that the generalization ability of the instruction-following system does not affect the performance of the autonomous agent; the learned agent may generalize well even if the instruction-following system heavily overfits the training data.

The idea of learning a policy through imitation is not limited to generalizing textual instructions. Any system that can provide examples of successful behavior can be used as a teacher for an imitation learning algorithm, whether or not the original system can generalize to new problem instances. By varying the source of the example behavior, imitation learning can be used to solve a variety of decision making problems. For example, if the training data is provided directly by an expert, the algorithm becomes an instance of *programming by demonstration* (Cypher 1993).

Imitation learning can also be used for traditional reinforcement learning, without the aid of expert guidance in either natural language form or through demonstration. The simplest way to generate training data is to generate a large number of random action sequences for each of the training games, and use the ones that achieve high rewards as examples for the imitation learning system. Such an approach is appropriate for domains with a clearly defined goal state, and where random action sequences have a non-negligible probability of reaching this goal state, on at least some of the training games. Since Crossblock meets these criteria, we implemented a version of our algorithm that uses this approach to learn a Crossblock policy without the aid of textual instructions.

For domains that do not meet the above criteria, a more complex approach is needed for generating training data. One approach would be to generate action sequences from some initial policy which may be suboptimal, though better than random. If a starting policy cannot be provided by an expert, it could be learned by policy gradient on a model with a large number of randomly generated features that depend on the state. Such a model would almost certainly fail to generalize to new problem instances, but may be adequate

[3]http://cs.washington.edu/homes/nath/chameleon

Table 1: Crossblock results

Algorithm	Score	Setting
BR-I	52%	Instructions at training and test
C-MLN-I	72%	Instructions at training
C-MLN-NI	62%	No instructions
BR-NI	34%	No instructions
RAND	13%	Random moves

to capture a good policy on the training data; this would allow it to function as a teacher for the imitation learner, despite being useless as an autonomous agent in its own right. Investigating the practicality of this approach is a direction for future work.

6 Experiments

We implemented two versions of our imitation learning algorithm, with different sources of training data.

- *C-MLN-I* learns from the model output by Branavan et al.'s algorithm, which uses policy gradient to map instructions to actions on the training games. C-MLN-I's training data consists of the winning action sequences taken by Branavan et al.'s model on the training games.
- *C-MLN-NI* creates training data by generating random action sequences for each of the training games, as described in section 5. We repeatedly generated sequences for each game until we found one winning sequence, or gave up after 200 failed attempts. The winning action sequences form the training set.

For both algorithms, the autonomous agent is learned from the training data using CHAMELEON. Each clause was restricted to a single free variable (i.e. each formula described a single row or column). Each term was restricted to at most three counting expressions on each side of the comparison. We used a clause length penalty of 0.01. Weight learning was run for 40 iterations during structure learning, and for 100 iterations once structure learning terminated.

We compared both algorithms to Branavan et al.'s (2009) instruction-following system, *BR-I*. Note that BR-I has access to textual instructions at both training and test time, while C-MLN-I only uses them in training, and C-MLN-NI does not use them at any time. Branavan et al. also developed a version of their algorithm without any text-based features; we refer to their system as *BR-NI* and report their results for comparison. Finally, we also report results for a baseline system that selected a random legal action in each step (*RAND*).

We evaluated these systems on the 50 Crossblock levels in the online implementation of the game, using the same textual instructions as Branavan et al. (2009). Grids ranged in size from 3x2 to 15x15, with the number of blocks removed per turn ranging from 2 to 7. Table 1 reports the percentage of games won, averaged over five random folds (each trained on 40 and tested on 10).

Interestingly, C-MLN-I wins 20% more games than BR-I, despite not having access to textual instructions at test time, and using BR-I as its teacher in its imitation learning phase. (The difference is statistically significant, as measured by the sign test with $p = 0.05$.) This is likely due to BR-I overfitting on the training set, because of the complexity of its model, which includes several features for each word in the instruction vocabulary. C-MLN-I's model only depends on the state and the action, and can therefore be represented compactly. On average, the learned C-MLNs had fewer than 10 formulas (and therefore 10 weights), while BR-I had over 9000. The following is an example of a formula learned by CHAMELEON:

$$(\#_y[\texttt{Square}(x,y)] \geq \#_y[\texttt{Clear}(x,y)] + \#_y[\texttt{Clear}(x,y)]) \\ \vee (\#_y[\texttt{Clear}(x,y)] \geq \#_y[\texttt{Square}(x,y)])$$

Intuitively, this formula states that each horizontal line segment that the player draws clears either an entire row or less than half the row. A move that violates this rule leaves behind too few blocks in that row to be cleared by another horizontal line segment; each remaining block will need to be cleared by a separate vertical line segment. This is more likely to lead to a dead end, making the game unwinnable. This rule is understandable and empirically useful, but not obvious to a human player.

C-MLN-NI also wins more games than BR-I, despite not using the instructions in any way (though in this case the difference is not statistically significant). C-MLN-NI and BR-NI are directly comparable, since both systems use features that only depend on the game state and not the text. While BR-NI's features were designed by hand, C-MLN-NI's features were learned by CHAMELEON. C-MLN-NI wins 28% more games than BR-NI. (The difference is significant, measured by a Fisher exact test with $p = 0.05$. We did not use a sign test since individual game results for BR-NI were not available.) It should be noted that building an autonomous Crossblock player was not the focus of Branavan et al.'s (2009) work, and BR-NI's features were not described in detail in their paper. However, this result is a promising proof of concept for using C-MLN-based imitation learning as part of a reinforcement learning system.

7 Future work

Aside from the decision-making problems dealt with in this paper, counting-based features can useful in many other domains, such as the following:

- Social networks: to describe 'hub' nodes with a large number of relationships; useful for viral marketing and epidemiological models.
- Anomaly detection: to reason about the counts or frequencies of different types of events.
- Resource allocation problems: to specify quantities of available resources.

Efficient inference algorithms would greatly increase the applicability of C-MLNs to these and other problems.

Much progress has been made recently in improving MLN structure and weight learning algorithms. Some of these new algorithms can be extended to learn C-MLNs. For example, Khot et al. (2011) use functional gradient boosting

to learn MLNs, learning a relational regression tree at each step. Relational regression trees have been extended to incorporate counting (Van Assche et al. 2006); however, since Khot et al. need to convert the learned trees into standard MLN clauses, they cannot make use of these extensions. Learning C-MLNs instead of traditional MLNs would remove this restriction.

8 Conclusions

- Counting-MLNs can compactly express formulas that pose problems for traditional statistical relational representations. Standard MLN weight and structure learning algorithms can be adapted to C-MLNs; however, more work is needed on efficient inference.
- CHAMELEON successfully generalizes the instruction-following model learned by Branavan et al.'s (2009) algorithm. Due to the simplicity of the learned C-MLN, it solves new problem instances more successfully than the original model, despite having access to less information at testing time.
- CHAMELEON can also learn to play Crossblock autonomously, without the use of textual instructions. Future work in this direction includes partially observable domains, more complex utility functions, etc.

Acknowledgments

This research was conducted while the first author was at Microsoft Research.

References

Besag, J. 1975. Statistical analysis of non-lattice data. *The Statistician* 24.

Branavan, S. R. K.; Chen, H.; Zettlemoyer, L.; and Barzilay, R. 2009. Reinforcement learning for mapping instructions to actions. In *Proc. of ACL-09*.

Branavan, S. R. K.; Zettlemoyer, L.; and Barzilay, R. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proc. of ACL-10*.

Chen, D. L., and Mooney, R. J. 2008. Learning to sportscast: A test of grounded language acquisition. In *Proc. of ICML-08*.

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP-02*.

Cypher, A. 1993. *Watch What I Do: Programming by Demonstration*. MIT Press.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *Proc. of IJCAI-05*.

Driessens, K., and Džeroski, S. 2004. Integrating guidance into relational reinforcement learning. *Machine Learning* 57.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43.

Genesereth, M.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AAAI Magazine* 26(2).

Khot, T.; Natarajan, S.; Kersting, K.; and Shavlik, J. 2011. Learning markov logic networks via functional gradient boosting. In *Proc. of ICDM-11*.

Kisyński, J., and Poole, D. 2009. Lifted aggregation in directed first-order probabilistic models. In *Proc. of IJCAI-09*.

Kok, S., and Domingos, P. 2005. Learning the structure of Markov logic networks. In *Proc. of ICML-05*.

Kok, S., and Domingos, P. 2010. Learning markov logic networks using structural motifs. In *Proc. of ICML-10*.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; Wang, J.; and Domingos, P. 2008. The Alchemy system for statistical relational AI. Technical report, University of Washington. http://alchemy.cs.washington.edu.

Lowd, D., and Domingos, P. 2007. Efficient weight learning for Markov Logic Networks. In *Proc. of PKDD-07*.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaebling, L. P. 2008. Lifted probabilistic inference with counting formulas. In *Proc. of AAAI-08*.

Natarajan, S.; Joshi, S.; ; Tadepalli, P.; Kersting, K.; and Shavlik, J. 2011. Imitation learning in relational domains: A functional-gradient boosting approach. In *Proc. of IJCAI-11*.

Pasula, H.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2005. Learning symbolic models of stochastic domains. *J. Artif. Intel. Res.* 1.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Poole, D. 2003. First-order probabilistic inference. In *Proc. of IJCAI-03*.

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

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82.

Roy, D. 2002. Learning visually grounded words and syntax for a scene description task. *Computer Speech and Language* 16.

Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *Proc. of AAAI-05*.

Taskar, B.; Abbeel, P.; and Koller, D. 2002. Discriminative probabilistic models for relational data. In *Proc. of UAI-02*.

Van Assche, A.; Vens, C.; Blockeel, H.; and Džeroski, S. 2006. First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning* 64.

Vogel, A., and Jurafsky, D. 2010. Learning to follow navigational directions. In *Proc. of ACL-10*.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2001. Understanding belief propagation and its generalizations. In *Proc. of IJCAI-01*.

Yu, C., and Ballard, D. H. 2004. On the integration of grounding language and learning objects. In *Proc. of AAAI-04*.