Improving Learning of Markov Logic Networks using Transfer and Bottom-Up Induction

Lilyana S. Mihalkova
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
lilyanam@cs.utexas.edu

Doctoral Dissertation Proposal

Supervising Professor: Raymond J. Mooney

Abstract

Statistical relational learning (SRL) algorithms combine ideas from rich knowledge representations, such as first-order logic, with those from probabilistic graphical models, such as Markov networks, to address the problem of learning from multi-relational data. One challenge posed by such data is that individual instances are frequently very large and include complex relationships among the entities. Moreover, because separate instances do not follow the same structure and contain varying numbers of entities, they cannot be effectively represented as a feature-vector. SRL models and algorithms have been successfully applied to a wide variety of domains such as social network analysis, biological data analysis, and planning, among others. Markov logic networks (MLNs) are a recently-developed SRL model that consists of weighted first-order clauses. MLNs can be viewed as templates that define Markov networks when provided with the set of constants present in a domain. MLNs are therefore very powerful because they inherit the expressivity of first-order logic. At the same time, MLNs can flexibly deal with noisy or uncertain data to produce probabilistic predictions for a set of propositions. MLNs have also been shown to subsume several other popular SRL models.

The expressive power of MLNs comes at a cost: structure learning, or learning the first-order clauses of the model, is a very computationally intensive process that needs to sift through a large hypothesis space with many local maxima and plateaus. It is therefore an important research problem to develop learning algorithms that improve the speed and accuracy of this process. The main contribution of this proposal are two algorithms for learning the structure of MLNs that proceed in a more data-driven fashion, in contrast to most existing SRL algorithms. The first algorithm we present, RTAMAR, improves learning by transferring the structure of an MLN learned in a domain related to the current one. It first diagnoses the transferred structure and then focuses its efforts only on the regions it determines to be incorrect. Our second algorithm, BUSL improves structure learning from scratch by approaching the problem in a more bottom-up fashion and first constructing a variablized Markov network template that significantly constrains the space of viable clause candidates. We demonstrate the effectiveness of our methods in three social domains.

Our proposed future work directions include testing BUSL in additional domains and extending it so that it can be used not only to learn from scratch, but also to revise a provided MLN structure. Our most ambitious long-term goal is to develop a system that transfers knowledge from multiple potential sources. An important prerequisite to such a system is a method for measuring the similarity between domains. We would also like to extend BUSL to learn other SRL models and to handle functions.

Contents

1 Introduction

Statistical relational learning (SRL) algorithms combine ideas from rich knowledge representations, such as first-order logic, with those from probabilistic graphical models, such as Bayesian networks, to address the problem of learning from multi-relational data. Such data involves multiple entities that participate in complex relationships. Frequently, individual training instances in the data do not follow the same structure. Consider, for example, a dataset that describes an academic department and includes information about all people, courses, and publications, as well as their relationships—who published a particular paper, who advises a particular student, and who is the instructor of a given course. Because different departments have varying numbers of students and professors, who publish different numbers of papers, and offer varying numbers of courses, a traditional feature vector representation would be unnatural and difficult to learn from. An additional challenge is that individual training instances can be very large because the complex interconnections among the entities they contain make it impossible to break them into smaller pieces. These arguments demonstrate that traditional machine learning classification techniques developed for feature vector representations are inadequate for multi-relational data, and new models and algorithms for learning and inference are needed when dealing with such data.

Apart from posing an interesting intellectual challenge, SRL is also very important in practice. SRL algorithms have been applied to a variety of problems including information extraction from natural language (e.g., Bunescu & Mooney, 2007), document mining (e.g., Popescul et al., 2003), social network analysis (e.g., Richardson & Domingos, 2006), entity resolution (e.g., Singla & Domingos, 2006), planning (e.g., Guestrin et al., 2003), biological data analysis (e.g., Perlich & Merugu, 2005), medical diagnosis (e.g., Davis et al., 2007), and others (Getoor & Diehl, 2005).

Markov logic networks (MLNs) (Richardson & Domingos, 2006) are a powerful and conceptually clean SRL model that can be roughly described as consisting of a set of weighted first-order clauses. Given the constants in a particular domain, MLNs can be grounded into Markov networks that can then be used to infer probabilities for a set of query literals given the truth values of a set of evidence literals. MLNs are capable of representing all possible probability distributions over a finite number of objects (Richardson & Domingos, 2006). Moreover, as demonstrated by Richardson (2004), MLNs subsume all SRL representations that can be formed as special cases of first-order logic or probabilistic graphical models. These include several widely used models, such as probabilistic relational models (Getoor et al., 2001) and relational Markov networks (Taskar, Abbeel, & Koller, 2002). There is also a publicly available codebase for MLN learning and inference (Kok et al., 2005). For the reasons discussed above, we have chosen MLNs as the model on which the research presented in this proposal is based.

The representational power of MLNs comes at a cost: structure learning, or learning the first-order clauses of the model, is a very computationally intensive process that needs to sift through a large hypothesis space with many local maxima and plateaus. It is therefore an important research problem to develop learning algorithms that improve the speed and accuracy of this process. The main contribution of this proposal are two algorithms for learning the structure of MLNs. A distinguishing characteristic of our methods is that, unlike most SRL algorithms, they use the data to narrow down the search space for possible structures, thus proceeding in a more bottom-up way. We demonstrate that, when carefully designed, more data-driven algorithms can learn more accurate structures significantly faster. Our proposed techniques are therefore an important step towards the ability to take advantage of the full expressive power of MLNs.

The first algorithm we present uses knowledge transfer to improve structure learning in a target domain. Rather than starting learning from scratch, transfer learning algorithms aim to improve their accuracy on a target task by utilizing knowledge acquired while learning in previous source domains (Thrun & Pratt, 1998). The specific approach to transfer we take is through mapping and revision. In the first stage, the MLN structure learned in the source domain is mapped to the predicates of the target domain. For example, if transferring from a movie domain to an academic domain, during this stage the algorithm might find that directors in the movie business are like professors in academia, that actors are like students, and that movies are like publications. In the second stage of the process, the mapped source structure is revised so that it better fits the training data in the target domain. If the source and target tasks are related, we expect that this process will save training time and will improve the accuracy of the learned model, particularly when data in the target domain is scarce. In this proposal, we focus on the second, revision, stage of the transfer process and present an algorithm that revises a mapped source structure. Our algorithm starts by autonomously diagnosing the mapped source structure in order to determine which of its portions can be reused unchanged in the target domain and which need to be updated. This is done by performing inference over the transfered model in the target domain and observing the accuracy of the individual clauses. The search for revisions is then limited to the incorrect portions of the source structure. Our experiments in several relational domains demonstrate that the diagnostic information allows our algorithm to significantly speed up learning over the current state-of-the-art MLN structure learner (Kok & Domingos, 2005), while maintaining an accuracy that is at least as high. These improvements are observed both when the existing algorithm starts from scratch and when it is provided with the source MLN.

The second algorithm we present is a novel approach to learning MLN structure from scratch called BUSL for Bottom-Up Structure Learning. Our approach breaks away from the top-down paradigm common in probabilistic graphical model learning where a greedy search through the hypothesis space is conducted by systematically generating a large number of candidates at each iteration, scoring them according to a probabilistic measure, and keeping the most promising ones from which new candidates are generated at the next iteration (e.g., Heckerman, 1995). Instead, BUSL proceeds in a more bottom-up fashion by first constructing a *Markov network template*, a variablized Markov network, whose nodes consist of chains of one or more literals and serve as clause building blocks. The Markov network template is used to restrict the search space for clauses by requiring that all literals in a clause be part of a clique in the template. This restriction is motivated by the observation that the clauses in an MLN define functions over the cliques of the Markov network obtained by grounding the MLN for a particular domain. Our experiments in three real relational domains demonstrate that this approach dramatically reduces the search space for clauses and attains a significantly higher accuracy than the current best MLN structure learning algorithm (Kok & Domingos, 2005), which follows a top-down approach.

Our short-term future work plans include testing BUSL in additional domains and extending it so that it can be used to revise a provided MLN. One possible way of performing revision with BUSL is by following the strategy used in our existing transfer learner and first diagnosing the source MLN. For example, by comparing the Markov network templates from the source and target domains, the algorithm can determine where there are newly-emerging or newly-disappearing dependencies and focus learning on those parts of the space.

Our longer-term research directions include:

- *Structure learning with functions*

  The capability to learn models that include functions is absent in both BUSL and previous work on structure learning for MLNs (Kok & Domingos, 2005). This ability would allow BUSL to learn models that take full advantage of the expressiveness of first-order logic and could be achieved by using ideas from least general generalizations, which includes a principled way of handling functions (Lavrač & Džeroski, 1994).

- *Transfer learning from multiple potential sources*

  Our present work on transfer learning has focused on the scenario where the learner is presented with the target task after learning in a single source domain. It assumes that the source and target tasks are related enough so that transfer is beneficial. A more challenging and interesting scenario is one in which the learner has compiled a library of previous training experiences. When faced with a new problem, it first needs to determine which of its previous domains are most closely related to the current one and transfer the knowledge only from them. This is related to the TC algorithm of Thrun and O'Sullivan (1996).

- *Extensions to* BUSL *for learning with other models*

  Our encouraging results with BUSL suggest that a similar learning technique can be applied to other SRL models, most notably Bayesian logic programs, which are first-order analogs to Bayesian networks (Kersting & De Raedt, 2001).

2 Background and Related Work

The research described in this document draws on several diverse bodies of work within artificial intelligence. In this section we give a brief overview of each of these areas.

2.1 First-Order Logic

First-order logic provides an expressive language for describing the features and relations that hold in an environment. It distinguishes among four types of symbols—constants, variables, predicates, and functions (Russell & Norvig, 2003). Constants describe the objects in a domain and can have types. For example, a domain may contain the constants *jack* and *jill* of type person and *male* and *female* of type gender. Variables act as placeholders to allow for quantification. Predicates represent relations in the domain, such as *WorkedFor*. Function symbols represent functions over tuples of objects. The arity of a predicate or a function is defined as the number of arguments it takes. These arguments can also be typed, thus restricting the type of constant that can be used. We will denote constants by strings starting with lower-case letters (i.e. *jill*), variables by single upper-case letters (i.e A, B), and predicates by strings starting with upper-case letters (i.e. *WorkedFor*). Sets of variables will be denoted with bold upper-case letters (i.e. $\mathbf{A}, \mathbf{B}$).

Example: *As a running example, we will use the following simplified version of one of our test domains. The domain contains facts about individuals in the movie business, describing their profession (Actor(A) or Director(A)), their relationships, and the movies on which they have worked. The WorkedFor(A, B) predicate specifies that person A worked on a movie under the supervision of person B, whereas the Movie(T, A) predicate specifies that individual A appeared in the credits of movie T. Here A, B, and T are variables. Actor and Director each have one argument of type person; WorkedFor has two arguments of type person; and Movie has two arguments where the first one is of type movieTitle and the second one is of type person. Our example domain has the constants brando and coppola of type person, and godFather of type movieTitle.*

A term is a constant, a variable, or a function that is applied to terms. Ground terms contain no variables. An atom is a predicate applied to terms. A positive literal is an atom, and a negative literal is a negated atom. We will use the term *gliteral* to refer to a ground literal, i.e. one containing only constants, and *vliteral* to refer to a literal that contains only variables. A clause is a disjunction of positive and negative literals.

Ground clauses contain only gliterals. The length of a clause is the number of literals in the disjunction. A definite clause is a clause with exactly one positive literal, called the head, whereas the negative literals compose the body. A Horn clause is a clause with at most one positive literal. A world is an assignment of truth values to all possible gliterals in a domain. If the closed-world assumption is made, only the true gliterals need to be listed; under this assumption all unlisted gliterals are assumed to be false. For the remainder of this document, we will make the closed-world assumption.

Example: *For example, WorkedFor(A, B) is a vliteral, while WorkedFor(brando,coppola) is a gliteral. The following clause is definite because it contains exactly one positive literal:*

$$\textit{Movie(T,B)} \vee \neg\ \textit{Movie(T,A)} \vee \neg\ \textit{WorkedFor(A, B)}.$$

Using the fact that $q \vee \neg p$ is logically equivalent to $p \Rightarrow q$, we can rewrite this clause in a more human-readable way, without modifying its meaning, as follows:

$$\textit{Movie(T, A)} \wedge \textit{WorkedFor(A, B)} \Rightarrow \textit{Movie(T,B)}.$$

Note that every definite clause of length at least 2 can be rewritten as a conjunction of positive literals that serve as the premises (the body) and a conclusion consisting of a single positive literal (the head).
One possible grounding of the above clause is:

$$\textit{Movie(godFather, brando)} \wedge \textit{WorkedFor(brando, coppola)} \Rightarrow \textit{Movie(godFather,coppola)}.$$

In fact, we can rewrite any clause as an implication. Consider, for example, the following clause, which is neither Horn, nor definite because it contains more than one positive literal:

$$\textit{Actor(A)} \vee \neg\ \textit{Movie(T, A)} \vee \textit{Director(A)}$$

This clause can be rewritten as an implication in several ways, depending on what literal we would like to serve as the conclusion:

$$\neg \textit{Actor(A)} \wedge \textit{Movie(T, A)} \Rightarrow \textit{Director(A)}$$

$$\textit{Movie(T, A)} \wedge \neg\ \textit{Director(A)} \Rightarrow \textit{Actor(A)}$$

$$\neg\ \textit{Actor(A)} \wedge \neg\ \textit{Director(A)} \Rightarrow \neg\ \textit{Movie(T, A)}$$

We will call the literals to the left of the implication premises or antecedents. The literal on the right of the implication will be called the conclusion. These implication rewrites will be helpful in Section 3.

2.2 Inductive Logic Programming

Inductive logic programming (ILP) is an area within machine learning that studies algorithms for learning sets of first-order clauses (Lavrač & Džeroski, 1994). Usually, the task is to learn rules for a particular target predicate, such as *WorkedFor*, given background knowledge. This background knowledge may consist either of general clauses, or, more commonly, of a list of the true gliterals of all predicates in the domain except the target predicate. The negative and positive examples are provided by the true and false gliterals of the target predicate (i.e. in our case *WorkedFor*). The form learned rules can take is frequently restricted by demanding that they be definite clauses (e.g., Richards & Mooney, 1995) or by allowing the user to impose some other declarative bias (e.g., De Raedt & Dehaspe, 1997). By performing techniques such as resolution on the learned clauses, new examples can be classified as positive or negative.

2.2.1 Top-Down ILP

Top-down ILP algorithms (e.g., Quinlan, 1990; De Raedt & Dehaspe, 1997) search the hypothesis space by considering, at each iteration, all valid refinements to a current set of candidate hypotheses. These candidates are then evaluated based on how well they cover positive examples and exclude negatives, a set of well-performing ones is greedily selected, and the process continues with the next iteration. In addition to classification accuracy, several other heuristics for scoring, or evaluating, candidates have been used. For example, FOIL uses an information theoretic measure of the information gained by adding a literal to a candidate clause (Quinlan, 1990); whereas CLAUDIEN uses a measure that takes into account the length of the clause (De Raedt & Dehaspe, 1997). In summary, top-down ILP techniques use the data only to evaluate candidate hypotheses but not to suggest ways for forming new candidates.

2.2.2 Bottom-Up ILP

Bottom-up ILP algorithms start with the most specific hypothesis and proceed to generalize it until no further generalizations are possible without covering some negative examples (Lavrač & Džeroski, 1994). For example, the initial hypothesis may be a set of rules where each rule's premises are simply a conjunction of the true gliterals in the background knowledge, and the conclusion is one of the positive examples. One way of generalizing this initial set of clauses is via the technique of *least general generalization* (LGG) (Plotkin, 1970), which can be intuitively understood as the most cautious, or conservative, generalization.

The LGG of two clauses c_1 and c_2, $\text{LGG}(c_1, c_2)$ is a clause formed by matching every possible pair of literals, one from c_1 and one from c_2, and applying an LGG to them if they are compatible. Two literals are compatible if they are of the same predicate and are both either negative or positive. The LGG of two compatible literals is a literal in which each argument is formed as an LGG of the corresponding arguments from the original literals. For example, $\text{LGG}(a, a) = a$, and $\text{LGG}(a, b) = A$ where a and b are constants and A is a variable.

The technique of LGG is appealing because it also provides a principled way of dealing with functions. The LGG of two functions f_1 and f_2 is defined as follows:

$$\text{LGG}(f_1(x_1, \ldots, x_n), f_2(y_1, \ldots, y_m)) = \begin{cases} V & \text{if } f_1 \neq f_2 \\ f\,(\text{LGG}(x_1, y_1), \ldots, \text{LGG}(x_n, y_n)) & \text{if } f = f_1 = f_2 \end{cases}$$

In words, the LGG of two different functions is a variable, whereas the LGG of two functions that are the same is the function of the LGG of each pair of corresponding arguments.

Example: *As an example, consider the LGG of the following two clauses, where directorOf(m) is a function that returns the director of movie m:*

$$\textit{Actor(brando)} \wedge \textit{Movie(godFather, brando)} \Rightarrow \textit{WorkedFor(brando, directorOf(godFather))}$$
$$\textit{Actor(brando)} \wedge \textit{Movie(streetcar, brando)} \Rightarrow \textit{WorkedFor(brando, directorOf(streetcar))}$$

The LGG of these two clauses is

$$\textit{Actor(brando)} \wedge \textit{Movie(A, brando)} \Rightarrow \textit{WorkedFor(brando, directorOf(A))}$$

Note that LGG is cautious in the sense that it generalizes constants to variables only if it observes at least two different constants in corresponding gliterals. This is why the constant brando was kept in the resulting clause.

One popular ILP system that uses LGG is GOLEM (Muggleton & Feng, 1992). LGG has also been used by Thomas (2003) to develop an algorithm that extracts information from hypertext documents.

An alternative method for bottom-up ILP is inverse resolution (Lavrač & Džeroski, 1994), in which the basic idea is to start from a positive example in the data and attempt to construct rules from which the example can be derived using resolution.

In summary, bottom-up ILP algorithms take stronger guidance from the data, which is also used to *propose* clause candidates. This is in contrast with top-down algorithms, which use the data only to evaluate candidate clauses.

2.2.3 Hybrid Approaches

Hybrid approaches (e.g., Zelle, Mooney, & Konvisser, 1994; Muggleton, 1995) aim to exploit the strengths of top-down and bottom-up techniques while avoiding their weaknesses. Because bottom-up techniques generalize from single examples, they are very sensitive to outliers and noise in the training data; however, because many bottom-up techniques employ LGGs, they are better-suited for handling functions. Similarly, top-down techniques can better make use of general background knowledge to evaluate their hypotheses, but the greedy search through the hypothesis space can lead to long training times.

For example, Zelle et al. (1994) present an approach, CHILLIN, that successfully improves accuracy over both a purely top-down and a purely bottom-up learner by combining ideas from these two paradigms. CHILLIN uses LGGs to form initial clauses and refines them further by searching for additional antecedents in a top-down way, as well as inventing new predicates that are necessary in order to express the target concept concisely.

Relational pathfinding (RPF), developed by Richards and Mooney (1992), is another hybrid approach to clausal discovery. RPF views the relational domain as a graph G in which the constants are the vertices and two constants are connected by an edge if they appear together in a true gliteral. Intuitively, RPF forms definite clauses in which the head is a particular true gliteral, and the body consists of gliterals that define a path in the relational graph G. These clauses are then variablized. More specifically, RPF searches G for an alternate path of length at least 2 between any two constants, $c1$ and $c2$, connected by an edge. If such a path is found, it is transformed into a clause as follows. First, a *negative* literal is created for each predicate that labels an edge in the path and is grounded with the constants connected by this edge. In addition, a *positive* literal is constructed in this way for the edge connecting $c1$ and $c2$. The resulting clause is a disjunction of these literals with constants replaced by variables. This is the bottom-up part of the process. Hill-climbing search, which proceeds in a top-down fashion, is used to further improve the clauses by possibly adding unary predicates.

Example: *Suppose Figure 1 lists all true facts in the domain. Figure 2 shows the relational graph for this domain. The highlighted edges form an alternative path between* $brando$ *and* $coppola$*, from which we construct the clause:*

$$\textit{WorkedFor(brando,coppola)} \vee \neg \textit{Movie(godFather,brando)} \vee \neg \textit{Movie(godFather,coppola)}.$$

After variablizing, this clause becomes:

$$\textit{WorkedFor(A,B)} \vee \neg \textit{Movie(T,A)} \vee \neg \textit{Movie(T,B)}.$$

This can be rewritten as

$$\textit{Movie(T,A)} \wedge \textit{Movie(T,B)} \Rightarrow \textit{WorkedFor(A,B)}.$$

Hill-climbing search might lead to the addition of $\neg \textit{Actor(A)}$ *and* $\neg \textit{Director(B)}$ *to the disjunction.*

Director(coppola) Actor(brando)
Movie(godFather, brando) Movie(godFather, coppola)
Movie(rainMaker, coppola) WorkedFor(brando, coppola)

Figure 1: Example relational database

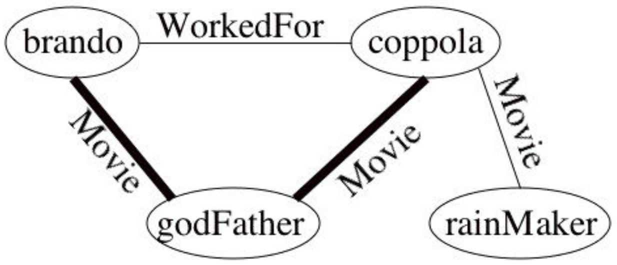


Figure 2: Example of a relational graph

2.2.4 Revision of Logic Programs

The ILP algorithms discussed so far all learn from scratch. Sometimes, however, an initial, somewhat incorrect, first-order logic theory is provided, along with training data, and the task is to revise the theory so that it fits the training data by modifying it as little as possible. This is the problem addressed by Richards and Mooney (1995). The resulting system, FORTE, can be viewed as a hybrid revision algorithm. FORTE is a top-down learner in that it uses hill-climbing search to improve the provided theory. However, rather than attempting all possible refinements to the provided clauses, FORTE starts in a bottom-up fashion and focuses its search by first diagnosing the possible sources of errors in the provided theory. It does this by attempting to prove positive examples and observing where the clauses fail. These points of failure are marked as revision points and are the only places in the original theory where attempts for improvements are made.

In a more recent vein of work, Goldsmith and Sloan (2005) present revision algorithms for restricted classes of Horn clauses. They give an algorithm for the case of depth-one acyclic Horn clauses in which variables that occur as a head in a clause do not appear in the body of any other clause. A second algorithm deals with the restricted case of Horn clauses with unique heads. The introduction of these subclasses of Horn clauses, allows the authors to give theoretical guarantees of the efficiency of their algorithms.

As we will discuss later, revision algorithms can be used in transfer learning settings when the initial first-order logic theory was learned in a previous domain, rather than being provided by a human.

All the approaches discussed in Section 2.1 result in the construction of first-order theories. Even though this representation is highly expressive, it is not well-suited to modeling uncertain domains and cannot provide estimates of the probability that a certain fact is true. We next turn to an overview of probabilistic graphical models, which provide an important step towards modeling uncertainty.

2.3 Probabilistic Graphical Models

Probabilistic graphical models provide a compact way of representing a joint probability distribution over sets of variables. Assuming that each variable can take on at most v values, any joint probability distribution can be expressed by listing the probability for every possible combination of assignments of values to the variables. If the total number of variables is n, this would require one to specify v^n parameters. Probabilistic graphical models take advantage of the observation that frequently a given variable is directly dependent on only a small subset of the variables, and this subset renders it conditionally independent of the rest. Thus, in a complete listing of probabilities for all possible value combinations, many of the parameters will have the same value. Probabilistic graphical models avoid this redundancy by explicitly modeling the conditional independencies in the domain. The variables are represented as nodes in a graph and the edges indicate dependencies among the variables. Probabilities are computed via a set of functions defined over the graph. For example, Bayesian networks (Pearl, 1988) are a popular model represented as a directed acyclic graph, in which the joint probability is computed using a set of conditional probability functions, one for each node in the graph, that specify the probability of that node taking a particular value given the values of its parents in the graph. Another popular probabilistic graphical model are Markov networks (Pearl, 1988), which, in contrast to Bayesian networks, are represented by undirected graphs and are therefore easier to learn because one does not need to ensure that the resulting graph is acyclic. We next describe in detail Markov networks because they will be important for understanding the work presented later in this document.

2.3.1 Markov Networks

A Markov network (Pearl, 1988), also known as a Markov random field (Della Pietra, Della Pietra, & Lafferty, 1997), is represented as an undirected graph G in which there is a vertex for each variable in the domain. The semantics of G is that each variable X is conditionally independent of all other variables, given its immediate neighbors. Because of its importance, the set of immediate neighbors of X is called a *Markov Blanket* of X and we will denote it with MB_X.

The probability distribution defined by a Markov network is described by a set of nonnegative functions $g_i(\mathbf{C_i})$ where $\mathbf{C_i}$ consists of the variables in the i-th maximal clique of G. The probability of assigning particular values $\mathbf{x}$ to the set of variables $\mathbf{X}$ in G (with the cliques having values $\mathbf{c_i}$) is:

$$P(\mathbf{X}=\mathbf{x}) = \frac{\prod_i g_i(\mathbf{c_i})}{\sum_{\mathbf{y}} \prod_i g_i(\mathbf{c_i})} \tag{1}$$

The function in the denominator, known as the *partition function*, simply sums over the values of the numerator for all possible value assignments to the variables and serves as a normalizing term. Intuitively, it is possible to represent a probability distribution that preserves the conditional independencies captured by G as a product of functions over only the cliques of G because a variable influences directly only its neighboring variables. This intuition has been formalized as the Hammersley Clifford Theorem (Hammersley & Clifford, 1971), which states that if P is a strictly positive probability distribution (i.e. every event has some chance of happening), then it can be expressed as a product of functions over the cliques of a graph G_P if and only if every conditional independence implied by the structure of G_P exists in P.

Markov networks are most commonly represented as log-linear models where the functions $g_i(\mathbf{C_i})$ take the form $\exp(\lambda_i f_i(\mathbf{C_i}))$. The λ_i-s are called weights, and the f_i-s are called features. With this formulation, equation 1 can be rewritten as follows:

$$P(\mathbf{X}=\mathbf{x}) = \frac{\exp\left(\sum_i \lambda_i f_i(\mathbf{C_i})\right)}{\sum_{\mathbf{y}} \exp\left(\sum_i \lambda_i f_i(\mathbf{C_i})\right)} \tag{2}$$

Apart from their convenience, log-linear models are desirable also because it can be shown that if such a model is used, optimizing the weights in order to maximize the data likelihood, leads to the model with the highest entropy (Berger, 1996; Della Pietra et al., 1997).

2.3.2 Learning of Markov Networks

If the features are given, one effective way of learning the weights is by using gradient descent because, for fixed features, optimization of the weights is over a convex space (Della Pietra et al., 1997). One common approach to learning the features of Markov networks, also known as *structure learning*, is by proceeding in iterations where in each iteration the feature that gives the best improvement in data fit is greedily added. For example, Della Pietra et al. (1997) choose the feature that gives the largest decrease in Kullback-Leibler divergence between the empirical distribution of the data and the distribution represented by the current model. It is also common to add a term that penalizes complex models (Lee, Ganapathi, & Koller, 2006). These types of approaches are *feature-centric* in that they focus on selecting the features that give the best immediate advantage, without considering the underlying graph structure and the implied conditional independencies among the variables.

An alternative approach to learning Markov networks is to proceed in a *graph-centric* way by first focusing on establishing a graph structure that asserts the existing conditional independencies among the variables. One such algorithm, which we will use in one of our methods in Section 4, is the Grow-Shrink Markov Network (GSMN) algorithm by Bromberg, Margaritis, and Honavar (2006). For each variable X, GSMN goes through two stages—grow and shrink. In the grow phase, the algorithm incrementally constructs the Markov blanket, MB_X, of each variable X. Initially MB_X is empty. The algorithm goes through all other nodes and at each iteration, uses the χ^2 test to determine whether X and Y are conditionally independent given MB_X, where Y is the current potential addition to MB_X. If the two variables are not conditionally independent, Y is added to MB_X. In the shrink phase, GSMN goes through each node $Y \in \mathrm{MB}_X$ and attempts to remove it by testing whether X and Y are conditionally independent given $\mathrm{MB}_X \setminus Y$. After going through the grow and shrink stages for each node, GSMN enters a collaboration phase in which the algorithm ensures that for all pairs of nodes X and Y, if $Y \in \mathrm{MB}_X$, then $X \in \mathrm{MB}_Y$.

Graph-centric algorithms for learning of other probabilistic graphical models include SGS and PC (Spirtes, Glymour, & Scheines, 2001) and Margaritis and Thrun's (2000) work that learn Bayesian networks based on independence tests among the variables, as well as the work of Abbeel, Koller, and Ng (2006) that constructs Markov blankets using conditional entropy.

Probabilistic graphical models can effectively represent probability distributions over a set of variables. However, they can capture dependencies only over a fixed set of (propositional) variables and cannot concisely model generally valid relationships that hold over large groups of objects. We will next turn to a short description of statistical relational learning which aims at overcoming this problem by incorporating ideas from first-order logic, while still maintaining the advantages of probabilistic graphical models.

2.4 Statistical Relational Learning

Statistical relational learning (SRL) (Getoor & Taskar, 2007) combines ideas from first-order logic and probabilistic graphical models to develop learning models and algorithms capable of representing complex relationships among entities in uncertain domains. As opposed to traditional classification where it is assumed that each testing instance is independent of the rest, SRL is best suited to situations in which the entities to be classified are interrelated and the label of one affects the classification of the remaining ones in some non-trivial way. Moreover, SRL addresses the case where learning occurs from multi-relational data, and thus training instances have varying numbers of entities and relations.

Some popular SRL models include probabilistic relational models (PRMs) (Getoor et al., 2001) and Bayesian logic programs (BLPs) (Kersting & De Raedt, 2001), which are both relational analogs to Bayesian networks; and relational Markov networks (RMNs) (Taskar et al., 2002) and Markov logic networks (Richardson & Domingos, 2006), which are relational analogs to Markov networks.

In the remainder of this subsection, we will describe in detail Markov logic networks, which are the SRL model on which the work in this proposal is focused. As discussed in the Introduction, the choice of this model is motivated by the fact that it is highly expressive and subsumes all SRL models that can be formed as special cases of first-order logic and probabilistic graphical models.

2.4.1 Markov Logic Networks

Markov logic networks (MLNs), introduced by Richardson and Domingos (2006), consist of a set of first-order clauses, each of which has an associated weight. MLNs can be viewed as relational analogs to Markov networks whose features are expressed in first-order logic. In this way MLNs combine the advantages of first-order logic with those of probabilistic graphical models while avoiding the drawbacks of the two representations. In particular, the expressive power of first-order logic enables MLNs to represent complex general relationships and to reason about variable numbers of entities using the same model. On the other hand, because the first-order logic features are embedded in the framework of probabilistic graphical models, MLNs avoid the brittleness of pure first-order logic by making worlds that violate some of the clauses less likely but not altogether impossible.

We next provide a formal description of MLNs. Let **X** be the set of all propositions describing a world (i.e. these are all possible gliterals that can be formed by grounding the predicates with the constants in the domain), $\mathscr{F}$ be the set of all first-order clauses in the MLN, and w_i be the weight associated with clause $f_i \in \mathscr{F}$. Let $\mathscr{G}_{f_i}$ be the set of all possible groundings of clause f_i with the constants in the domain. Then, the probability of a particular truth assignment **x** to **X** is given by the formula (Richardson & Domingos, 2006):

$$P(\mathbf{X}=\mathbf{x}) = \frac{\exp\left(\sum_{f_i\in\mathscr{F}} w_i \sum_{g\in\mathscr{G}_{f_i}} g(\mathbf{x})\right)}{\sum_{\mathbf{y}} \exp\left(\sum_{f_i\in\mathscr{F}} w_i \sum_{g\in\mathscr{G}_{f_i}} g(\mathbf{y})\right)} \tag{3}$$

The value of $g(\mathbf{x})$ is either 1 or 0, depending on whether g is satisfied. Thus the quantity $\sum_{g\in\mathscr{G}_{f_i}} g(\mathbf{x})$ simply counts the number of groundings of f_i that are true given the current truth assignment to **X**. The denominator is the normalizing partition function. Intuitively w_i determines how much less likely is a world in which a grounding of f_i is not satisfied than one in which it is satisfied. The first-order clauses are commonly referred to as *structure*. Figure 3 shows a simple MLN that provides an example for our simplified movie domain. Note that the first-order formulas do not have to have any particular form, i.e. they are not restricted to being definite.

0.7	$Actor(A) \Rightarrow \neg Director(A)$
1.2	$Director(A) \Rightarrow \neg WorkedFor(A,B)$
1.4	$Movie(T,A) \wedge WorkedFor(A,B) \Rightarrow Movie(T,B)$

Figure 3: Simple MLN for the sample domain

To perform inference over a given MLN, one needs to ground it into its corresponding Markov network. As described by Richardson and Domingos (2006), this is done as follows. First, all possible gliterals in the domain are formed, and they serve as the nodes in the Markov network. The edges are determined by the groundings of the first-order clauses: gliterals that participate together in a grounding of a clause, are connected by an edge. Thus, nodes that appear together in a ground clause form cliques. For example, Figure 4 shows the ground Markov network corresponding to the MLN in Figure 3 using the constants *coppola* and *brando* of type person and *godFather* of type movieTitle. It is also useful to note the similarity between equation 3 and equation 2. MLNs can be considered as a concise and general way of specifying Markov networks in which there is a feature for each grounding of each clause, and features that correspond to the same unground clause have the same weight.

One technique that can be used to perform inference over the ground Markov network is Gibbs sampling (Richardson & Domingos, 2006). The goal of sampling is to compute the probability that each of a set of query gliterals is true, given the values of the remaining gliterals as evidence. Gibbs sampling starts by assigning a truth value to each query gliteral. This can be done either randomly or by using a weighted satisfiability solver such as MaxWalksat (Kautz, Selman, & Jiang, 1997) that initializes the truth values to maximize the sum of the weights. It then proceeds in rounds to re-sample a value for gliteral X, given the truth values of its Markov blanket MB_X (i.e. the variables with which it participates in ground clauses), using the following formula to calculate the probability that X takes on a particular value x.

$$P(X = x|\mathrm{MB}_X = \mathbf{m}) = \frac{e^{S_X(x,\mathbf{m})}}{e^{S_X(0,\mathbf{m})} + e^{S_X(1,\mathbf{m})}}. \tag{4}$$

Here, $S_X(x,\mathbf{m}) = \sum_{g_i \in \mathscr{G}_X} w_i g_i(X = x, \mathrm{MB}_X = \mathbf{m})$, where $\mathscr{G}_X$ is the set of ground clauses in which X appears and $\mathbf{m}$ is the current truth assignment to MB_X. Efficiency can be improved by including only the query gliterals and those in the Markov blanket of a gliteral with an unknown value, rather than fully grounding the MLN (Richardson & Domingos, 2006).

An alternative inference approach is MC-SAT that has been shown to outperform Gibbs sampling in both speed and the accuracy of the returned probability estimates (Poon & Domingos, 2006).

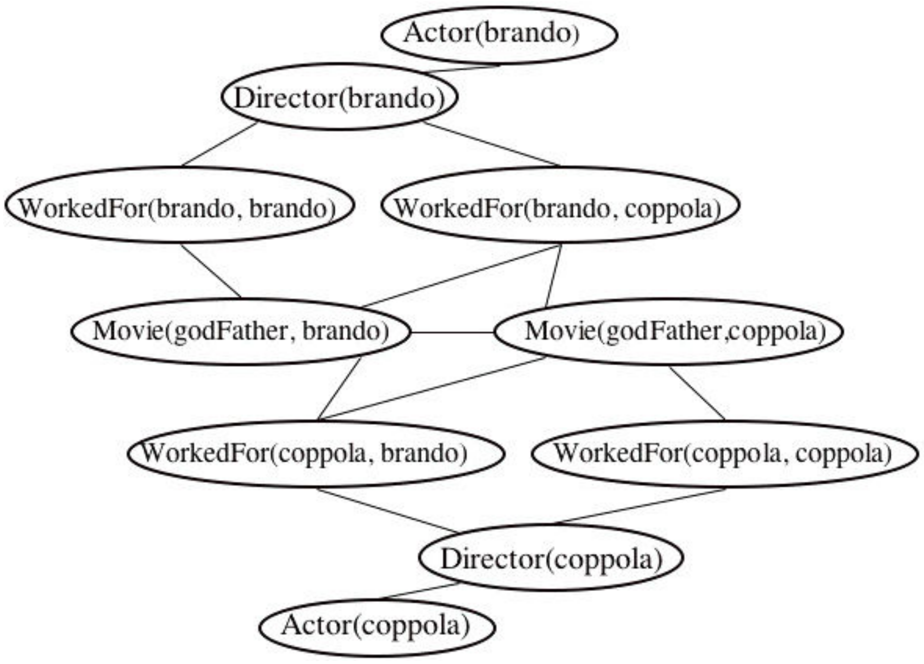


Figure 4: Result of grounding the sample MLN

2.4.2 Learning of Markov Logic Networks

As with Markov networks, there are two parts to learning an MLN: the weights and the structure. Richardson and Domingos (2006) propose performing weight learning for a fixed set of clauses using L-BFGS (Liu & Nocedal, 1989), a second-order optimization procedure. This process is highly efficient.

Structure learning, on the other hand, is very computationally intensive. The current state-of-the-art MLN structure learning algorithm, due to Kok and Domingos (2005), proceeds in a top-down fashion, employing either beam search or shortest-first search. We will discuss and compare to the beam search version, which we will call KD after its authors. The shortest-first search constructs candidates in the same way but conducts a more complete search, which, however, requires longer training times. KD performs several iterations of beam search, and after each iteration adds to the MLN the best clause found. Clauses are evaluated using a weighted pseudo log-likelihood measure (WPLL), introduced in (Kok & Domingos, 2005), that sums over the log-likelihood of each node given its Markov blanket, weighting it appropriately to ensure that predicates with many gliterals do not dominate the result. The beam search in each iteration starts from all single-vliteral clauses. It generates candidates by adding a vliteral in each possible way to the initial clauses, keeps the best *beamSize* clauses, from which it generates new candidates by performing all possible vliteral additions, keeps the best *beamSize* and continues in this way until candidates stop improving the WPLL. At this point, the best candidate found is added to the MLN, and a new beam search iteration begins. Weights need to be learned for a given structure before its WPLL can be computed. KD has been empirically shown to outperform an impressive number of competitive baselines (Kok & Domingos, 2005). In particular, it performed better than several popular inductive logic programming algorithms and also outperformed purely probabilistic methods.

2.5 Learning through Revision

Rather than learning from scratch, revision algorithms refine a partially incorrect model that is provided to them at the start of learning. Revision algorithms have been developed for a variety of learning models. One such algorithm, FORTE, was described in section 2.2.4. Paes, Revoredo, Zaverucha, and Costa (2005) extended FORTE to allow it to handle Bayesian logic programs (Kersting & De Raedt, 2001). Both of these algorithms first diagnose the provided model and then focus the search for revisions on the potentially faulty regions. An analogous approach is used by Ramachandran and Mooney (1998) for revision of Bayesian networks where the source networks are instrumented with leak nodes that are then used as indicators for errors.

If the provided model was learned in a related domain, revision algorithms can be used for transfer learning, the area of machine learning on which we focus next.

2.6 Transfer Learning

Transfer learning, also known as learning to learn (Thrun & Pratt, 1998), addresses the problem of how to leverage knowledge from related domains in order to improve the efficiency and accuracy of learning in a new domain. Transfer learning algorithms have been demonstrated to improve the performance of a variety of learning algorithms, as we describe next. In this section, we provide a glimpse of the numerous transfer algorithms that have been developed.

2.6.1 Multi-task Transfer Learning

Transfer learning has been studied in two main settings. In the multi-task setting, the algorithm is presented with all domains simultaneously during training and thus can build common structure of the learned models. For example, in (Caruana, 1997) neural networks with a shared hidden layer are trained on two or more tasks simultaneously. A related approach is used in (Niculescu-Mizil & Caruana, 2005, 2007) for simultaneous training of Bayesian networks. Exploiting a similar idea, Ando and Zhang (2005) perform optimization over a set of tasks simultaneously to find an optimal parameterization of the hypothesis space, and then optimize a linear predictor from this hypothesis space for the target task.

2.6.2 Single-task Transfer Learning

In an alternative transfer setting, tasks are presented to the learner one by one and the goal is to improve learning on the current, target, task by utilizing some knowledge acquired in previous learning domains. For example, Taylor, Stone, and Liu (2005) use the value function learned in a source task to initialize reinforcement learning in the target task. Guestrin et al. (2003) transfer value functions learned separately for different objects in a planning domain. Value function transfer is also used by Banerjee and Stone (2007) to transfer knowledge across game-playing domains by using state features extracted from look-ahead game trees. Torrey et al. (2005) propose extracting advice from the value function learned in the source task, which is then provided to a reinforcement learner in the target task. Taylor, Whiteson, and Stone (2007) propose using policies learned in the source task to direct reinforcement learning in the target task in a more promising direction.

Transfer learning in the single-task setting has been studied also for other types of machine learning. One of the earliest approaches, the TC Algorithm by Thrun and O'Sullivan (1996), improves target task performance of a nearest-neighbor algorithm by transferring the distance metrics learned on related problems over the same feature space. An interesting aspect of the TC algorithm is that rather than assuming that the previously-encountered tasks are similar, it autonomously determines task relatedness by using a validation set to estimate how likely it is that a distance metric optimized for a previous task improves performance on the target task. Bonilla et al. (2006) propose a method for transfer learning for estimation of distribution algorithms (EDA) in which a solution to an optimization problem is found by progressively developing a distribution over solutions that estimates the likelihood that a particular solution is optimal. In this work, transfer is achieved by initializing the EDA algorithm with the solution distribution of previously-solved problems. This is done by either combining the predictive distributions from all previous problems or from the k most similar ones, which are found using a k-nearest-neighbor algorithm. Raina, Ng, and Koller (2006) use several related source tasks to construct the covariance matrix for a Gaussian prior in a text classification task.

We next present the first contribution of this proposal: an algorithm for transfer learning of MLNs in the single-task setting.

3 Revising Markov Logic Network Structure for Transfer Learning

The problem of transferring the structure of an MLN from a source to a target domain can be viewed as consisting of two parts. First, in order to translate the source structure to the target domain, a correspondence between the predicates of the source domain and those of the target domain needs to be established. Second, once the source structure has been translated, it needs to be revised in order to adapt it to the target domain.

Source clause:

Publication(T, A) $\wedge$ Publication(T, B) $\wedge$ Professor(A) $\wedge$ Student(B) $\wedge\neg$SamePerson(A, B) $\Rightarrow$ AdvisedBy(B, A)

Best mapping:

Publication(title,person)	$\rightarrow$ Movie(movie,person)
Professor(person)	$\rightarrow$ Director(person)
Student(person)	$\rightarrow$ Actor(person)
SamePerson(person,person)	$\rightarrow$ SamePerson(person,person)
AdvisedBy(person,person)	$\rightarrow$ WorkedFor(person,person)

Figure 5: An output of the predicate mapping algorithm

This section focuses on solving the second problem and describes our algorithm for revising the structure of the source MLN. The algorithm assumes that the predicates in the source structure have been mapped to the target domain. This is a safe assumption because this mapping capability was developed by Tuyen Huynh as part of TAMAR, a complete transfer system (Mihalkova, Huynh, & Mooney, 2007). We will call the mapping portion of TAMAR, MTAMAR. MTAMAR maps individually every source clause to the target domain. This is done by considering all possible ways of translating the source predicates appearing in the clause to the predicates in the target domain, while ensuring that the resulting mapping does not cause an argument type from the source domain to be mapped to more than one type in the target domain. Each possible valid clause mapping is evaluated using the WPLL score computed on the target data, and the mapping that achieves the highest score is output. For example, Figure 5 shows a sample output of the mapping algorithm (Mihalkova et al., 2007).

3.1 Revision of MLN Structure for Transfer

We will call the revision portion of TAMAR, RTAMAR. The skeleton of RTAMAR has three steps and is similar to that of FORTE (Richards & Mooney, 1995), which revises first-order theories.

1. **Self-Diagnosis:** The purpose of this step is to focus the search for revisions only on the inaccurate parts of the MLN. The algorithm inspects the source MLN and determines for each clause whether it should be shortened, lengthened, or left as is. For each clause *C*, this is done by considering every possible implication rewrite of *C* in which one of the literals is placed on the right-hand side of the implication and is treated as the conclusion and the remaining literals serve as the antecedents. The conclusion of a clause is drawn only if the antecedents are satisfied and the clause "fires." Thus, if a clause makes the wrong conclusion, it is considered for lengthening because the addition of more literals, or conditions, to the antecedents will make them harder to satisfy, thus preventing the clause from firing. On the other hand, there may be clauses that fail to draw the correct conclusion because there are too many conditions in the antecedents that prevent them from firing. In this case, we consider shortening the clause.

2. **Structure Update:** Clauses marked as too long are shortened, while those marked as too short are lengthened.

3. **New Clause Discovery:** Using relational pathfinding (RPF) (Richards & Mooney, 1992), new clauses are found in the target domain.

We next describe each step in more detail.

3.1.1 Self-Diagnosis

A natural approach to self-diagnosis is to use the transferred MLN to make inferences in the target domain and observe where its clauses fail. This suggests that the structure can be diagnosed by performing Gibbs sampling over it. Specifically, this is done as follows. Each predicate in the target domain is examined in turn. The current predicate under examination is denoted as P^*. Self-diagnosis performs Gibbs sampling with P^* serving as a query predicate with the values of its gliterals set to unknown, while the gliterals of all other predicates provide evidence. In each round of sampling, in addition to re-sampling a value for gliteral X, the algorithm considers the set of all ground clauses $\mathcal{G}_X$ in which X participates.

Each ground clause $C \in \mathcal{G}_X$ can be placed in one of four bins with respect to X and the current truth assignments to the rest of the gliterals. These bins consider all possible cases of the premises being satisfied and the conclusion being correct. We label a clause as Relevant if the premises are satisfied and Irrelevant otherwise. Similarly, we mark a clause as Good if its conclusion is correct and Bad if the conclusion is incorrect. The four bins are defined by all possible ways of marking a clause as Relevant/Irrelevant and Good/Bad.

Let v be the actual value of X. This value is known from the data, even though for the purposes of sampling we have set it to unknown. As an illustration, we will use some groundings of the clauses in Figure 6 with respect to the data in Figure 1 (page 10) listing the current truth assignments to the gliterals (the ones present are true; the rest are false). Figure 6 also lists rewrites of the clauses in implication form where the implication is with respect to the target predicate. This will be helpful in the exposition of the algorithm. Let $X =$ Actor(brando) with $v = true$.

Weight	Clausal form	Implication form wrt target predicate *Actor*
1.5	Director(A)∨Actor(A)	¬Director(A) ⇒ Actor(A)
0.1	Movie(M, A)∨¬Actor(A)	¬Movie(M, A) ⇒ ¬Actor(A)
1.3	¬WorkedFor(B, A)∨¬Actor(A)	WorkedFor(B, A) ⇒ ¬Actor(A)
0.5	Actor(A)∨¬Movie(M,A)∨¬WorkedFor(A, B)	Movie(M, A) ∧ WorkedFor(A, B) ⇒ Actor(A)

Figure 6: Example MLN for diagnosis

- **[Relevant; Good]** This bin contains clauses in which the premises are satisfied and the conclusion drawn is correct. For example, let C be ¬Director(brando)⇒Actor(brando). Considering the clausal form of this formula, Director(brando) ∨ Actor(brando), we can alternatively describe clauses in this bin as ones which hold true only if X has value v, the value it has in the data.

- **[Relevant; Bad]** The clauses in this bin are those whose premises are satisfied but the conclusion drawn is incorrect. For example, let C be ¬Movie(rainMaker, brando) ⇒ ¬ Actor(brando). Considering the clausal form, Movie(rainMaker, brando) ∨¬ Actor(brando), we see that this bin contains clauses that are only satisfied if X has value $\neg v$, the negation of its correct value in the data.

- **[Irrelevant; Good]** This bin contains clauses whose premises are not satisfied, and therefore the clauses do not "fire," but if they were to fire, the conclusion drawn would be incorrect. For example, let C be WorkedFor(coppola, brando) ⇒ ¬Actor(brando). In clausal form this formula is ¬WorkedFor(coppola, brando) ∨¬Actor(brando). Thus a more mechanical way of describing the clauses in this bin is that they are satisfied regardless of the value of X in the data; however, the literal corresponding to X in C is true only if X has value $\neg v$.

- **[Irrelevant; Bad]** The clauses in this bin are those whose premises are not satisfied, but if the clauses were to fire, the conclusion would be correct. For example, let C be Movie(rainMaker, brando) $\wedge$ WorkedFor(brando, coppola) $\Rightarrow$ Actor(brando). If we consider the clausal form, Actor(brando)$\vee\neg$Movie(rainMaker,brando)$\vee\neg$WorkedFor(brando, coppola), we can alternatively describe the clauses in this bin as ones that are satisfied regardless of the value of X and in which the literal corresponding to X in C is true only if X has value v.

This taxonomy is motivated by equation 4. The probability of $\mathbf{X} = \mathbf{x}$ is increased only by clauses in the **[Relevant; Good]** bin and is decreased by clauses in the **[Relevant; Bad]** bin. Clauses in the other two bins do not have an effect on this equation because their contribution to the numerator and denominator cancels out. To see how this happens, consider a clause $g_{irr} \in \mathcal{G}_X$ from the set of clauses in which X participates, such that g_{irr} is satisfied regardless of the truth value of X. The quantity $S_X(x, \mathbf{m})$ from equation 4 (page 14) can be rewritten as follows:

$$
\begin{aligned}
S_X(x,\mathbf{m}) &= \sum_{g_i \in \mathcal{G}_X} w_i g_i(X = x, \mathrm{MB}_X = \mathbf{m}) &(5)\\
&= \sum_{g_i \in \mathcal{G}_X, i \neq irr} w_i g_i(X = x, \mathrm{MB}_X = \mathbf{m}) + w_{irr} g_{irr}(X = x, \mathrm{MB}_X = \mathbf{m}) &(6)\\
&= \sum_{g_i \in \mathcal{G}_X, i \neq irr} w_i g_i(X = x, \mathrm{MB}_X = \mathbf{m}) + w_{irr} &(7)\\
&= S^*_X(x,\mathbf{m}) + w_{irr} &(8)
\end{aligned}
$$

The next-to-last line follows because g_{irr} was picked such that the value of $g_{irr}(X = x, \mathrm{MB}_X = \mathbf{m})$ is 1 regardless of x. Using this derivation, we can rewrite equation 4 as follows:

$$
\begin{aligned}
P(X = x|\mathrm{MB}_X = \mathbf{m}) &= \frac{e^{S_X(x,\mathbf{m})}}{e^{S_X(0,\mathbf{m})} + e^{S_X(1,\mathbf{m})}} &(9)\\
&= \frac{e^{S^*_X(x,\mathbf{m})+w_{irr}}}{e^{S^*_X(0,\mathbf{m})+w_{irr}} + e^{S^*_X(1,\mathbf{m})+w_{irr}}} &(10)\\
&= \frac{e^{w_{irr}} e^{S^*_X(x,\mathbf{m})}}{e^{w_{irr}} \left(e^{S^*_X(0,\mathbf{m})} + e^{S^*_X(1,\mathbf{m})}\right)} &(11)
\end{aligned}
$$

As can be seen in line 11, the contribution of g_{irr}, $e^{w_{irr}}$, can be canceled from the numerator and denominator.

If some of the literals other than X in an **[Irrelevant; Bad]** clause, are deleted so that X's value becomes crucial, it will be moved to the **[Relevant; Good]** bin. Similarly, if we add some literals to a **[Relevant; Bad]** clause so that it starts to hold regardless of the value of X, it will enter the **[Irrelevant; Good]** bin and will no longer decrease the probability of X having its correct value.

As the value of a gliteral is re-sampled in each iteration of Gibbs sampling, for each clause in which the gliteral participates, we count the number of times it falls into each of the four bins. Finally, if a clause was placed in the **[Relevant; Bad]** bin more than p percent of the time, it is marked for lengthening and if it fell in the **[Irrelevant; Bad]** bin more than p percent of the time, it is marked for shortening. We anticipated that in the highly sparse relational domains in which we tested, clauses would fall mostly in the **[Irrelevant; Good]** bin. To prevent this bin from swamping the other ones, we set p to the low value of 10%. This value was set during earlier experiments on artificial data (Mihalkova & Mooney, 2006) and was not tuned to the data used for the experiments presented here.

The process described above is repeated for each predicate, P^*, in the target domain.

3.1.2 Structure Updates

Once the set of clauses to revise is determined, the actual updates are performed using beam search. Unlike Kok and Domingos (2005), however, we do not consider all possible additions and deletions of a literal to each clause. Rather, we only try removing literals from the clauses marked for shortening and we try literal additions only to the clauses marked for lengthening. The candidates are scored using WPLL. Thus, the search space is constrained first by limiting the number of clauses considered for updates, and second, by restricting the kind of update performed on each clause.

3.1.3 New Clause Discovery

The revision procedure can update clauses transferred from the source domain but cannot discover new clauses that capture relationships specific only to the target domain. To address this problem, we used RPF (Section 2.2.3) to search for new clauses in the target domain. The clauses found by RPF were evaluated using WPLL, and the ones that improved the overall score were added to the MLN. RPF and the previous structure updates step operate independently of each other; in particular, the clauses discovered by RPF are not diagnosed nor revised. However, we found that better results are obtained if the clauses discovered by RPF are added to the MLN before carrying out the revisions. This can be explained as follows. The revision step fills the resulting structure with clauses that together achieve a very good WPLL on the training data. If we perform RPF after this, even though it finds clauses that are very reasonable and would perform quite well, the MLN already has other clauses that interfere. In this way, the good clauses discovered by RPF sometimes end up not being added. On the other hand, if we first add the RPF clauses to the MLN, they give an initial boost in WPLL and also constrain the beam search, causing it to finish faster because it has less to improve.

3.2 Experiments

In this section we present an experimental evaluation of TAMAR. First, we describe our methodology, which will also be used for the experiments in Section 4. We then discuss the results specific to TAMAR.

3.2.1 Experimental Methodology

We used three real-world relational domains—IMDB, UW-CSE, and WebKB. Each dataset is broken into *mega-examples*, where each mega-example contains a connected group of facts. Individual mega-examples are independent of each other.

The IMDB database is organized as five mega-examples, each of which contains information about four movies, their directors, and the first-billed actors who appear in them. Each director is ascribed genres based on the genres of the movies he or she directed. The Gender predicate is only used to state the genders of actors. The complete list of predicates in this domain is given in Figure 7 a). This data set is dramatically smaller than the data available from the International Movie Database (www.imdb.com). The reason for this is that originally the dataset was intended to be used as a target domain in which data is limited.

The UW-CSE database was first used by Richardson and Domingos (2006).[1] It lists facts about people in an academic department (i.e. Student, Professor) and their relationships (i.e. AdvisedBy). The complete list of predicates is given in Figure 7 b). The database is divided into mega-examples based on five areas of computer science.

[1]Available at http://www.cs.washington.edu/ai/mln/database.html.

Director(person)
Actor(person)
Movie(title, person)
Gender(person, gend)
WorkedUnder(person, person)
Genre(person, genr)
SamePerson(person, person)
SameMovie(title, title)
SameGenre(genr, genr)
SameGender(gend, gend)

a)

TaughtBy(course, person, semester)
CourseLevel(course, level)
Position(person, pos)
AdvisedBy(person, person)
ProjectMember(project, person)
Phase(person, phas)
TempAdvisedBy(person, person)
YearsInProgram(person, year)
TA(course, person, semester)
Student(person)
Professor(person)
SamePerson(person, person)
SameCourse(course, course)
SameProject(project, project)
Publication(title, person)

b)

Student(person)
SamePerson(person, person)
Faculty(person)
Project(projname, person)
CourseTA(coursename, person)
CourseProf(coursename, person)

c)

Figure 7: Predicates in each of our domains. The argument types for each predicate are listed in the parentheses

The WebKB database contains information about human relationships from the "University Computer Science Department" data set, compiled by Craven et al. (1998). The original dataset contains web pages from four universities labeled according to the entity they describe (e.g. student, course), as well as the words that occur in these pages. Our version of WebKB contains the predicates listed in Figure 7 c). The textual information is ignored. This data contains four mega-examples, each of which describes one university. To extract the truth values for these predicates, we used the files from the original data set that list the student, faculty, instructors-of, and members-of-project relationships. We treated each web address in these files as an entity in the domain and used the label of the corresponding page to determine the gliteral truth values.

Table 1 provides additional statistics about the domains.

Data Set	Number of Consts	Number of Types	Number of Preds	Number of True Gliterals	Total Number of Gliterals
IMDB	316	4	10	1,540	32,615
UW-CSE	1,323	9	15	2,673	678,899
WebKB	1,700	3	6	2,065	688,193

Table 1: Statistics about the domains.

To evaluate a given MLN, one needs to perform inference over it, providing some of the gliterals in the test mega-example as evidence and testing the predictions of the remaining ones. We followed the testing scheme employed by Kok and Domingos (2005) and tested for the gliterals of each of the predicates of the domain in turn, providing the rest as evidence, and averaging over the results. However, for inference we used the MC-SAT algorithm that has been demonstrated to give more accurate results (Poon & Domingos, 2006). The inference procedure outputs the probability that each of the query gliterals is true. To summarize these results, we used the two evaluation metrics employed by Kok and Domingos (2005), the area under the precision-recall curve (AUC) and the conditional log-likelihood (CLL). To compute the AUC, first a precision-recall curve is generated. This is done by varying a probability threshold whose value determines which propositions are labeled positive and which negative; i.e. the ones whose probability of being true is greater than the threshold are positive and the rest are negative. The precision and recall are computed as follows:

$$\text{Precision} = \frac{\text{Number of propositions correctly labeled as positive}}{\text{Number of all propositions labeled as positive}}$$

$$\text{Recall} = \frac{\text{Number of propositions correctly labeled as positive}}{\text{Total number of positive propositions in the data}}$$

A curve is produced by plotting a point for the precision and recall obtained at a set of threshold values. The AUC is the area under this curve. The AUC is useful because it demonstrates how well the algorithm predicts the few positives in the data and is not affected by the large number of true negatives typically present in relational data sets (compare the number of true gliterals to the total number of gliterals in Table 1).

The CLL is computed by taking the log of the probability predicted by the model that a gliteral has its correct truth value in the data, and averaging over the query gliterals. The CLL complements the AUC because it determines the quality of the probability predictions output by the algorithm.

Learning curves for each performance measure were generated using a leave-1-mega-example-out approach, averaging over *k* different runs, where *k* is the number of mega-examples in the domain. In each run, we reserved a different mega-example for testing and trained on the remaining $k-1$, which were provided one by one. All systems observed the same sequence of mega-examples. The error bars on the curves are formed by averaging the standard error over the predictions for the groundings of each predicate and over the learning runs. Error bars are drawn on all curves but in some cases they are tiny.

We also present results on the training times needed by the learners, and the number of clauses they considered in their search. Timing runs within the same transfer experiment were conducted on the same dedicated machine.

3.2.2 Systems Compared

We compared the performance of the following systems. KD run from scratch (ScrKD) in the target domain; KD used to revise a source structure translated into the target domain using MTAMAR (TrKD); and the complete transfer system using MTAMAR and RTAMAR (TAMAR).

We used the implementation of KD provided as part of the Alchemy software package (Kok et al., 2005) and implemented our new algorithms as part of the same package. We kept the default parameter settings of Alchemy except that we set the parameter penalizing long clauses to 0.01, the one specifying the maximum number of variables per clause to 5, and the minWeight parameter to 0.1 in IMDB and WebKB and to 1 in UW-CSE, the value used in (Kok & Domingos, 2005). All three learners used the same parameter settings.

We considered the following transfer scenarios: WebKB → IMDB, UW-CSE → IMDB, WebKB → UW-CSE, IMDB → UW-CSE. We did not consider transfer to WebKB because the small number of predicates and large number of constants in this domain made it too easy to learn from scratch and therefore a good source domain but uninteresting as a target domain. Source MLNs were learned by ScrKD . We also consider the scenario where the hand-built knowledge base provided with the UW-CSE data is used as a source MLN (UW-KB → IMDB). In this interesting twist on traditional theory refinement, the provided theory needs to be mapped to the target domain, as well as revised.

	TR		PI	
Experiment	TrKD	TAMAR	TrKD	TAMAR
WebKB → IMDB	1.51	1.55	50.54	53.90
UW-CSE → IMDB	1.42	1.66	32.78	52.87
UW-KB → IMDB	1.61	1.52	40.06	45.74
WebKB → UW-CSE	1.84	1.78	47.04	37.43
IMDB → UW-CSE	0.96	1.01	-1.70	-2.40

Table 2: Transfer ratio (TR) and percent improvement from 1 mega-example (PI) on **AUC** over ScrKD .

	TR		PI	
Experiment	TrKD	TAMAR	TrKD	TAMAR
WebKB → IMDB	1.41	1.46	51.97	67.19
UW-CSE → IMDB	1.33	1.56	49.55	69.28
UW-KB → IMDB	1.21	1.44	30.66	58.62
WebKB → UW-CSE	1.17	1.36	19.48	32.69
IMDB → UW-CSE	1.62	1.67	34.69	54.02

Table 3: Transfer ratio (TR) and percent improvement from 1 mega-example (PI) on **CLL** over ScrKD.

3.2.3 Results

Rather than presenting the full learning curves, we summarize them using two statistics: the transfer ratio (TR) (Cohen, Chang, & Morrison, 2007), and the percent improvement from 1 mega-example (PI). TR is the ratio between the area under the learning curve of the transfer learner (TAMAR or TrKD) and the area under the learning curve of the learner from scratch (ScrKD). Thus, TR gives an overall idea of the improvement achieved by transfer over learning from scratch. TR > 1 signifies improvement over learning from scratch in the target domain. PI gives the percent by which transfer improves accuracy over learning from scratch after observing a single mega-example in the target domain. It is useful because, in transfer learning settings, data for the target domain is frequently limited.

In terms of AUC (Table 2), both transfer systems improve over ScrKD in all but one experiment. Neither transfer learner consistently outperforms the other on this metric. Table 3 shows that transfer learning always improves over learning from scratch in terms of CLL, and TAMAR's performance is better than TrKD's in all cases. In the last experiment where we transferred from IMDB → UW-CSE we observe that transfer improves over learning from scratch in terms of CLL but is slightly worse in terms of AUC. This demonstrates that the two performance metrics complement each other. We believe this slightly worse performance of the transfer systems is probably due to random noise.

Moreover, as can be seen from Table 4, TAMAR trains faster than TrKD, and both transfer systems are faster than ScrKD. TAMAR also considers fewer candidate clauses during its beam search, as can be seen in Table 5. According to a t-test performed for each point on each of the learning curves, at the 95% level with sample size 5 per point, these differences were significant in 15 out of 20 cases for speed and 18 out of 20 for number of candidates. In some cases TrKD considers more candidates than ScrKD but takes less time to train. This can happen if TrKD considers more candidates earlier in the learning curves when each candidate is evaluated faster on less data.

Experiment	ScrKD	TrKD	TAMAR	Speed-Up factor of TAMAR over TrKD
WebKB→IMDB	62.23	32.20	11.98	2.69
UW-CSE→IMDB	62.23	38.09	15.21	2.50
UW-KB→IMDB	62.23	40.67	6.57	6.19
WebKB→UW-CSE	1127.48	720.02	13.70	52.56
IMDB→UW-CSE	1127.48	440.21	34.57	12.73

Table 4: Average (over all learning curve points) total training time in minutes.

Experiment	ScrKD	TrKD	TAMAR
WebKB→IMDB	7558	10673	1946
UW-CSE→IMDB	7558	14163	1976
UW-KB→IMDB	7558	15118	1613
WebKB→UW-CSE	32096	32815	827
IMDB→UW-CSE	32096	7924	978

Table 5: Average (over all learning curve points) number of candidate clauses considered.

The complete learning curves are given in Appendix 1. Here we present the most interesting among them. Figure 8 shows the learning curve in the UW-CSE → IMDB experiment. Here we additionally tested the performance of systems that do not use MTAMAR but are provided with an intuitive hand-constructed mapping that maps Student → Actor, Professor → Director, AdvisedBy/TempAdvisedBy → WorkedFor, Publication → MovieMember, Phase → Gender, and Position → Genre. The last two mappings are motivated by the observation that Phase in UW-CSE applies only to Student and Gender in IMDB applies only to Actor, and similarly Position and Genre apply only to Professor and Director respectively.

3.3 Summary

In this section we presented our algorithm, RTAMAR, for revising the structure of an MLN learned in a source domain and mapped to the predicates of a target domain. Our experimental results demonstrate that our proposed algorithm is competitive with the current state-of-the-art revision algorithm but requires significantly less time and considers a much smaller number of candidate structures during training. Moreover, because TAMAR outperforms learning from scratch in almost all cases, our results demonstrate that our specific approach to transfer through mapping and revision is effective. In the following section, we present an alternative approach to improving MLN structure learning.

4 Bottom-up Learning of Markov Logic Network Structure

In Section 3 we described how to improve learning in the target domain by revising the structure of an MLN learned in a source domain. In this section we present a novel algorithm that aims at improving MLN structure learning from scratch by approaching the problem in a more bottom-up way. We call our algorithm BUSL for Bottom-Up Structure Learning (Mihalkova & Mooney, 2007).

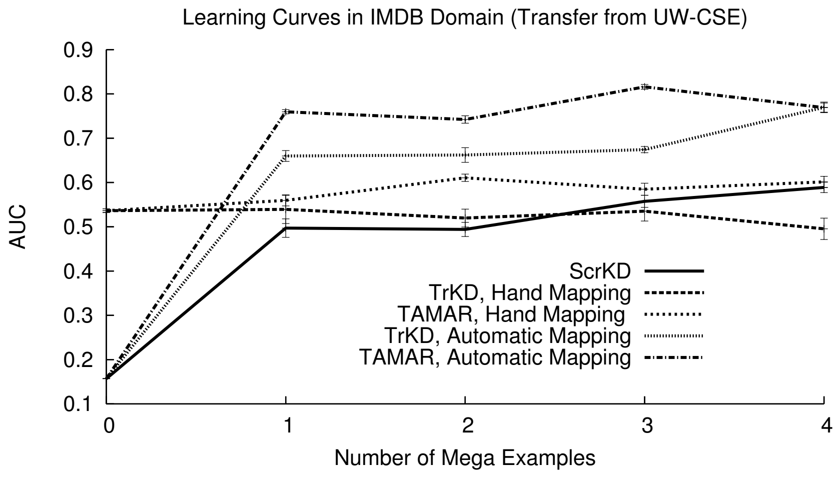


Figure 8: Learning curves in UW-CSE → IMDB for a) AUC and b) CLL. The zeroth points are obtained by testing the MLN provided to the learner at the start.

4.1 BUSL Overview

As pointed out by Richardson and Domingos (2006), MLNs serve as templates for constructing Markov networks when different sets of constants are provided. Because the cliques of the ground Markov network are defined by the groundings of the same set of first-order clauses, the graph exhibits a high degree of redundancy where the same pattern is repeated several times, corresponding to each grounding of a particular clause.

Example: *Considering Figure 4 (page 14) again, we observe that the pattern of nodes and edges appearing above the two Movie gliterals is repeated below them with different constants. In fact, this Markov network can be viewed as an instantiation of the template shown in Figure 9.*

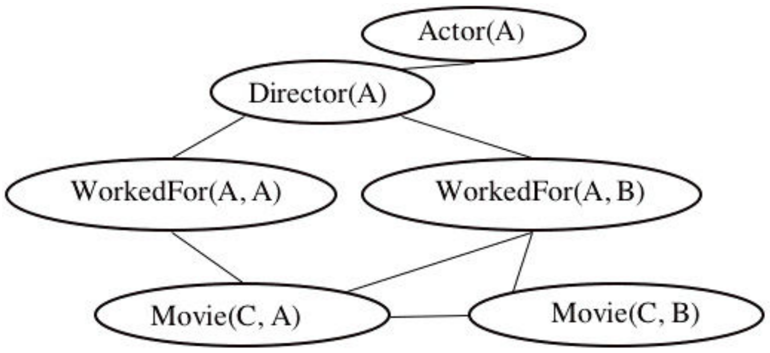


Figure 9: Example Markov Network Template

The basic idea behind BUSL is to learn MLN structure by first creating a Markov network template similar to the one shown in Figure 9 from the provided data. The nodes in this template are used as components

Algorithm 1 Skeleton of BUSL

```
for each $P \in \mathscr{P}$ do
    Construct TNodes for predicate $P$ (Section 4.2.1)
    Connect the TNodes to form a Markov network template (Section 4.2.2)
    Create candidate clauses, using this template to constrain the search (Section 4.2.3)
end for
Remove duplicate candidates
Evaluate candidates using WPLL and add best ones to final MLN
```

from which clauses are constructed, and can contain one or more vliterals that are connected by a shared variable. We will call these nodes *TNodes* for template nodes. As in ordinary Markov networks, a TNode is independent of all other TNodes given its immediate neighbors. Recall from Section 2.3.1, that the Hammersley Clifford Theorem guarantees that we can specify any probability distribution compliant with the conditional independencies implied by a particular graph by using functions defined only over the cliques of the graph. In the case of MLNs where the functions are expressed as first-order logic rules, this implies that to learn the structure, the algorithm only needs to consider clause candidates that comply with the Markov network template. In other words, BUSL uses the Markov network template to restrict the search space for clauses only to those candidates whose literals correspond to TNodes that form a clique in the template.

The approach taken by BUSL follows the same philosophy as the graph-centric learners discussed in Section 2.3.2 where the algorithm first focuses on learning the conditional independencies among the variables before specifying the features that define the probability distribution. This is in stark contrast to KD, which takes a feature-centric approach and proceeds by directly learning the clauses of the MLN.

Algorithm 1 gives the complete skeleton of BUSL. Letting $\mathscr{P}$ be the set of all predicates in the domain, the algorithm considers each predicate $P \in \mathscr{P}$ in turn. A Markov network template is *automatically* constructed from the perspective of the current target predicate P. Template construction involves creating variablized TNodes, or components for clause construction, and determining the edges between them. Even though the template aids the search for clauses, it does not carry all the information about the MLN. Namely, it does not specify whether the vliterals participating in a clause are positive or negative, or precisely what clauses correspond to a given clique. For example, a three-node clique could correspond to one three-literal clause or to three two-literal clauses, etc. Information about the weights is also excluded. To search for actual clauses, we generate clause candidates by focusing on each maximal clique in turn and producing all possible clauses consistent with it. More specifically, these are all possible clauses of length 1 to *cliqueSize* containing only members of the clique. We can then evaluate each candidate using the WPLL score.

In the following section we give the details of each step.

4.2 BUSL details

A Markov network template is created for each predicate in the domain in order to ensure that the relationships of all predicates are properly modeled. Below, we describe the process for the current target predicate P.

4.2.1 TNode construction

TNodes contain conjunctions of one or more vliterals and serve as building blocks for creating clauses. Intuitively, TNodes are constructed by looking for groups of constant-sharing gliterals that are true in the data

Algorithm 2 Construct TNode Set

```
Input:
 1: P: Predicate currently under consideration
 2: m: Maximum number of vliterals in a TNode
Output:
 3: TNodeVector: Vector of constructed TNodes
 4: M_P: Matrix of Boolean values
Procedure:
 5: Make head TNode, headTN, and place it in position 0 of TNodeVector
 6: for each (true or false) gliteral, G_P, of P do
 7:   Add a row of 0-s to M_P
 8:   currRowIndex = numRows(M_P) − 1
 9:   if G_P is true then
10:     Set M_P[currRowIndex][0] = 1
11:   end if
12:   Let $\mathscr{C}_{G_P}$ be the set of true gliterals connected to G_P
13:   for each c ∈ $\mathscr{C}_{G_P}$ do
14:     for each possible TNode of length 1 to m based on c do
15:       newTNode = CreateTNode(c, G_P, headTN, m) (Algorithm 3)
16:       position = TNodeVector.find(newTNode)
17:       if position is not valid then
18:         append newTNode to end of TNodeVector
19:         append a column of 0-s to M_P
20:         position = numColumns(M_P) − 1
21:       end if
22:       Set M_P[currRowIndex][position] = 1
23:     end for
24:   end for
25: end for
```

and variablizing them. Thus, TNodes could also be viewed as portions of clauses that have true groundings in the data. The process of TNode construction is inspired by relational pathfinding (Richards & Mooney, 1992), which we described in Section 2.2.3. The result of running TNode construction for P is the set of TNodes and a matrix M_P containing a column for each of the created TNodes and a row for each gliteral of P. Each entry $M_P[r][c]$ is a Boolean value that indicates whether the data contains a true grounding of the TNode corresponding to column c with at least one of the constants of the gliteral corresponding to row r. This matrix is used later to find the edges between the TNodes.

Algorithm 2 describes how the set of TNodes and the matrix M_P are constructed. The algorithm uses the following definitions:

Definition 1 *Two gliterals are* connected *if there exists a constant that is an argument of both of them. Similarly, two vliterals are connected if there exists a variable that is an argument of both of them.*

Definition 2 *A* chain *of literals of length l is a list of l literals such that for $1 < k \leq l$ the kth literal is connected to the $(k-1)$th via a previously unshared variable.*

Algorithm 3 CreateTNode

```
Input:
1: G_P: Current gliteral of P under consideration
2: c: Gliteral connected to G_P on which this TNode is based
3: headTN: Head TNode
4: m: Maximum number of of vliterals allowed in a TNode
Output:
5: newTNode: The constructed TNode
Procedure:
6: size = pick the number of vliterals in this TNode (between 1 and m)
7: v = variablize c such that the constants shared with G_P are replaced with their corresponding variables
   from headTN and all others are replaced with unique variables
8: Create newTNode containing v
9: previousGliteral = c
10: lastVliteralInChain = v
11: while length(newTNode) < size do
12:     c_1 = pick true gliteral connected to previousGliteral via a previously unshared constant
13:     v_1 = variablize c_1 such that constants shared with G_P or previousGliteral are replaced with their
        corresponding variables from headTN or lastVliteralInChain and all others are replaced with
        unique variables
14:     Add v_1 to newTNode
15:     previousGliteral = c_1
16:     lastVliteralInChain = v_1
17: end while
```

First, in line 5 the algorithm creates a *head* TNode that consists of a vliteral of P in which each argument is assigned a unique variable. This TNode is analogous to the head in a definite clause; however, note that our algorithm is not limited to constructing only definite clauses. Next, in lines 6 to 25 the algorithm considers each gliteral G_P of P in turn. This includes both the true and the false gliterals of P, where the true gliterals are those stated to hold in the data, while the rest are assumed to be false. A row of zeros is added to M_P for G_P, and the value corresponding to the head TNode is set to 1 if G_P is true and to 0 otherwise (lines 9-11). The algorithm then proceeds to consider the set $\mathscr{C}_{G_P}$ of all true gliterals in the data that are connected to G_P. For each $c \in \mathscr{C}_{G_P}$, it constructs each possible TNode based on c containing 1 to m vliterals. If a particular TNode was previously created, its value in the row corresponding to G_P is set to 1. Otherwise, a new column of zeros is added to M_P and the entry in the G_P row is set to 1 (lines 16-22). Thus, each entry in M_P indicates whether the TNode corresponding to its column could be formed when considering the gliteral corresponding to its row.

Algorithm 3 lists the `CreateTNode` procedure. The algorithm determines the number of vliterals in the new TNode in line 6. It then variablizes the current gliteral c connected to G_P by replacing the constants c shares with G_P with their corresponding variables from the head TNode. If the chosen TNode size is greater than 1, the algorithm enters the while loop in lines 11-17. In each iteration of this loop we extend the TNode with an additional vliteral that is constructed by variablizing a gliteral connected to the gliteral considered in the previous iteration so that any constants shared with the head TNode or with the previous gliteral are replaced with their corresponding variables.

Example: *Suppose we are given the following database for our example domain where the listed gliterals*

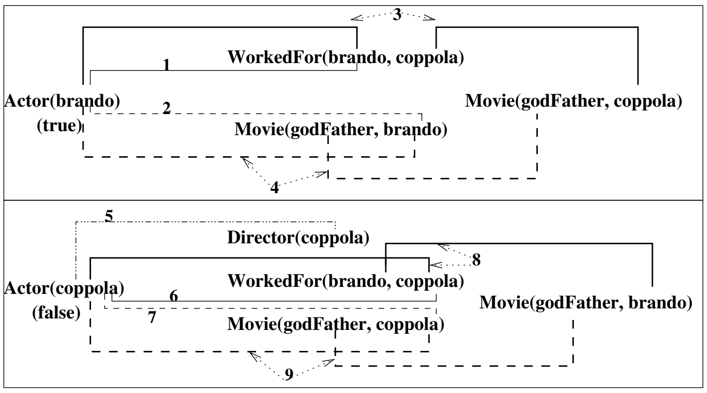


Figure 10: Illustration of TNode construction (See example below). The thin lines show the connections defining single-vliteral TNodes, and the thick lines the connections defining two-vliteral TNodes. The lines link the constants shared between the gliterals.

are $true$ and the omitted ones are $false$:

$Actor(brando)$ $Director(coppola)$
$WorkedFor(brando, coppola)$
$Movie(godFather, coppola)$ $Movie(godFather, brando)$

Let $P = Actor$ and $m = 2$ (i.e. at most 2 vliterals per TNode). The head TNode is Actor(A). Figure 10 shows the gliteral chains considered in the main loop (lines 6-25) of Algorithm 2 for each gliteral of P. Let us first focus on the case when G_P is Actor(brando) (top part). Connections 1 and 2 lead to the TNodes WorkedFor(A, B) and Movie(C, A) respectively. Connection 3 gives rise to the 2-vliteral TNode [WorkedFor(A, D), Movie(E, D)], and connection 4, to the TNode [Movie(F, A), Movie(F, G)]. The following table lists the values in M_P at this point.

Actor(A)	*WorkedFor(A, B)*	*Movie(C, A)*	*WorkedFor(A, D) Movie(E, D)*	*Movie(F, A) Movie(F, G)*
1	*1*	*1*	*1*	*1*

Note that when constructing the TNodes, we replaced shared constants with the same variables, and constants shared with G_P with the corresponding variable from the head TNode.
We did not consider the chain [Movie(godFather, brando), WorkedFor(brando, coppola)]. This chain is invalid because the shared constant, brando, has been shared previously. We can use this example of an invalid chain to provide some intuition for the requirement that a chain can be extended only by sharing a previously unshared constant. Suppose that this restriction did not exist. Then we would form the TNode

[Movie(X1,A), WorkedFor(A, X2)]

However, we notice that the vliterals composing this new TNode are present, modulo variable renaming, in two separate TNodes found earlier (the second and third TNodes in the table above). Therefore, constructing this TNode has the effect of re-discovering previous single-vliteral TNodes and forcing their vliterals to appear together in clauses.

We next consider the bottom part of Figure 10 that deals with the second iteration in which G_P is Actor(coppola). From connection 5, we construct the TNode Director(A) and from connection 6 the TNode WorkedFor(H, A), which differs from the WorkedFor TNode found earlier by the position of the variable A shared with the head TNode. An appropriate TNode for connection 7 (Movie(C,A)) already exists. Connection 8 gives rise to the two-vliteral TNode [WorkedFor(I, A), Movie(J, I)]. A TNode for connection 9, [Movie(F, A), Movie(F, G)] was constructed in the previous iteration. Table 6 lists the final set of TNodes.

Actor(A)	WkdFor(A, B)	Movie(C, A)	Director(A)	WkdFor(D, A)	WkdFor(A, E), Movie(F, E)	Movie(G, A), Movie(G, H)	WkdFor(I, A), Movie(J, I)
1	1	1	0	0	1	1	0
0	0	1	1	1	0	1	1

Table 6: Final set of TNodes and their corresponding M_P matrix

Larger values of m mean longer TNodes that could help build more informative clauses. However, a larger m also leads to the construction of more TNodes, thus increasing the search space for clauses. We used a conservative setting of $m = 2$. Note that this does not limit the final clause length to 2. To further reduce the search space, we require that TNodes with more than one vliteral contain at most one free variable (i.e. a variable that does not appear in more than one of the vliterals in the TNode or in the head TNode). We did not experiment with more liberal settings of these parameters but, as our experiments demonstrate, these values worked well in our domains.

TNode construction is very much in the spirit of bottom-up learning. Rather than producing all possible vliterals that share variables with one another in all possible ways, the algorithm focuses only on vliterals for which there is a true gliteral in the data. Thus, the data already guides and constrains the algorithm. This is related to bottom-up ILP techniques such as least-general generalizations (LGG) and inverse resolution (Lavrač & Džeroski, 1994). However, as opposed to LGG, our TNode construction algorithm always uses the generalization that leads to completely variablized TNodes and unlike inverse resolution, the process does not lead to the creation of complete clauses and does not use any logical inference algorithms like resolution.

4.2.2 Adding the Edges

Once TNodes are constructed, we can search through the space of all possible clauses composed from them. This search space is already smaller than the one considered by KD because the algorithm uses only combinations of vliterals that contain at least one true grounding in the data. Nevertheless, the number of possible clauses may still be prohibitively large. Moreover, as discussed in Section 4.1, an exhaustive search is not necessary. Thus we proceed to complete the template construction, by finding which TNodes are connected by edges. For this purpose, it is useful to recall that the templates represent variablized analogs of Markov networks. Finding the edges can therefore be cast as a Markov network structure learning problem where the TNodes are the nodes in the Markov network and the matrix M_P provides training data. At this point, any Markov network learning algorithm can be employed. We chose the recent Grow-Shrink Markov Network (GSMN) algorithm by Bromberg et al. (2006), which we described in Section 2.3.2, because its graph-centric approach to the problem follows the general philosophy of BUSL.

4.2.3 Search for clauses

Because the clauses in an MLN define functions over the cliques in the ground MLN, we should only construct clauses from TNodes that form cliques in the Markov network template. In other words, any two TNodes participating together in a clause must be connected by an edge in the template. The head TNode is required to participate in every candidate. Each clause can contain at most one multiple-literal TNode and at most one TNode that contains a single non-unary literal. These further restrictions on the clause candidates are designed to decrease the number of free variables in a clause, thus decreasing the size of the ground MLN during inference, and further reducing the search space. Complying with the above restrictions, we consider each clique in which the head TNode participates and construct all possible clauses of length 1 to the size of the clique by forming disjunctions from the literals of the participating TNodes with all possible negation/non-negation combinations.

After template creation and clause candidate generation are carried out for each predicate in the domain, duplicates are removed and the candidates are evaluated using the WPLL score described in Section 2.4.2. Recall that in order to compute this score, one needs to assign a weight to each clause. Weight learning is performed using L-BFGS, the approach used by Richardson and Domingos (2006) and also used in KD. After all candidates are scored, they are considered for addition to the MLN in order of decreasing score. To reduce overfitting and speed up inference, only candidates with weight greater than *minWeight* are considered. Candidates that do not increase the overall WPLL of the currently learned MLN are discarded.

4.3 Experimental Setup

We compared the performance of BUSL to that of KD in the same three relational domains—IMDB, UW-CSE, and WebKB—that we described in Section 3.2.1. It is important to note that our results on the UW-CSE dataset are not comparable to those presented by Kok and Domingos (2005) because due to privacy issues we only had access to the published version of this data, which differs from the original (Personal communication by Stanley Kok).

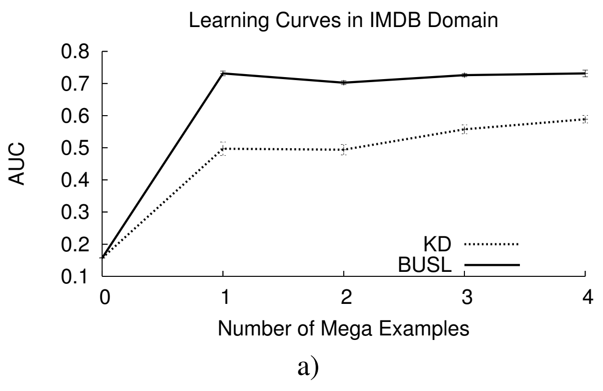
As in Section 3.2.1, we measured the performance in terms of the AUC and CLL and generated learning curves using a leave-1-mega-example-out approach. The parameter settings for running KD from scratch were identical. As before, all timing runs within the same experiment were carried out on the same dedicated machine. We implemented BUSL as part of the Alchemy package (Kok et al., 2005). We set BUSL's *minWeight* $= 0.5$ for all experiments and observed that the operation of the algorithm is not very sensitive to other settings of this parameter. Even though both BUSL and KD have a parameter called *minWeight*, they use it in different ways and the same value is therefore not necessarily optimal for both systems.

4.4 Experimental Results

Figures 11-13 show learning curves in the three domains. BUSL improves over the performance of KD in all cases except for one in terms of AUC and in all cases in terms of CLL.

Figure 12 additionally plots the AUC and CLL for a system that performs weight learning over the knowledge base provided as part of the UW-CSE dataset (Hand-KB). In terms of AUC, this system's performance is significantly worse than that of BUSL, and in terms of CLL, it performs as well as BUSL.

In Figure 13, we observe that even though KD is improving its performance in terms of AUC, its CLL score decreases. This is most probably due to the extremely small relative number of true gliterals in the domain in which the CLL can be increased by simply predicting $false$ for each query.



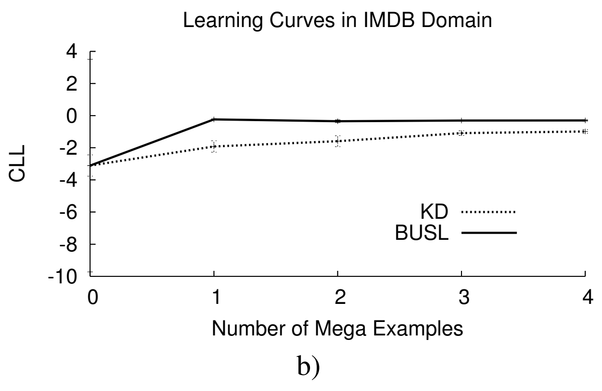
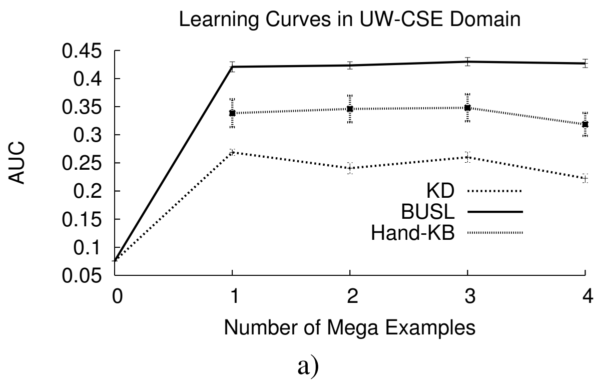


Figure 11: Accuracy in IMDB domain. a) AUC b) CLL



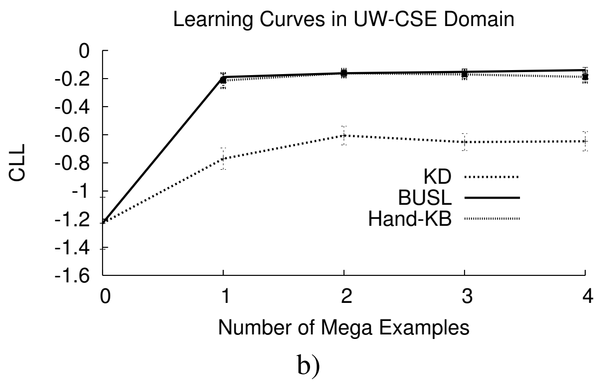
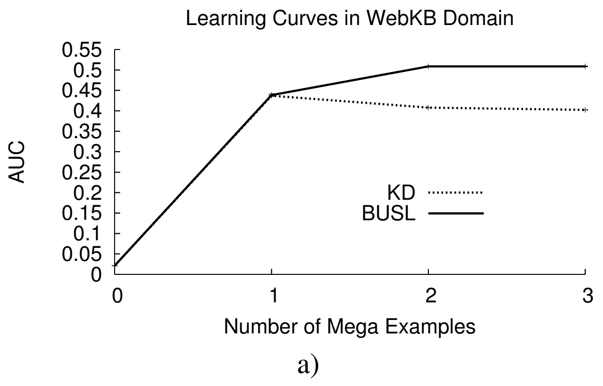


Figure 12: Accuracy in UW-CSE domain. a) AUC b) CLL



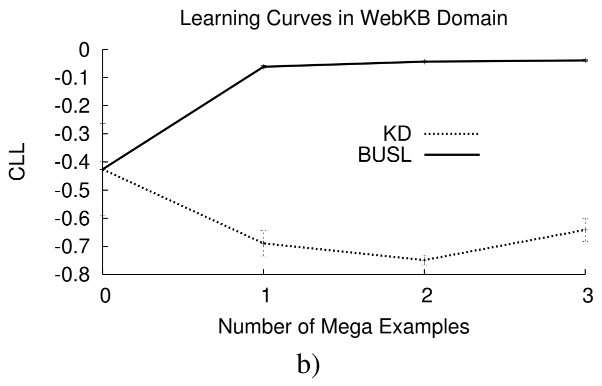


Figure 13: Accuracy in WebKB domain. a) AUC b) CLL

Another observation that requires explanation is that the learners improve by only tiny amounts, if at all, after the first point on the learning curve. This occurs because in our experience, for both learners, additional data improves only the WPLL estimate (and thus the evaluation of new clause candidates) but does not have a great effect on the clauses that are proposed. In particular, in BUSL candidates are based on the dependencies among the TNodes, and new data introduces few such dependencies. This, however, may not be the case in other domains.

Figures 11-13 give an idea of how the learners perform over all the predicates of the domain. It is also interesting, however, to see the performance of the systems for each predicate in the domains individually. Tables 7-9 show these results for AUC and CLL for the last point on the learning curves. Note that the performance for some of the predicates, such as TaughtBy in UW-CSE is extremely low. This is due to the fact that, given the information provided during testing, it is impossible to reliably predict the value of these predicates.

Table 10 shows the average training time over all learning runs for each system, and the average number of candidate clauses each learner constructed and evaluated over all runs. As can be seen, BUSL constructed fewer candidates and trained faster than KD. BUSL spends the main portion of its training time on computing the WPLL score of the generated candidates. This process takes longer in domains like WebKB that contain a great number of constants. On the other hand, we expect BUSL's savings in terms of number of generated candidates to be greater in domains, such as UW-CSE, that contain many predicates because the large number of predicates increases the number of candidate clauses generated by KD. These considerations explain why the smallest improvement in speed is achieved in WebKB that contains the least number of predicates and the greatest number of constants. The greatest speed-up is in IMDB where BUSL created the smallest number of candidates, and each candidate could be evaluated quickly because of the small number of constants in this domain.

Based on the much smaller number of candidate clauses considered by BUSL, one might expect a larger speed-up. Such a speed-up is not observed because of optimizations within Alchemy that allow fast scoring of clauses for a fixed structure of the MLN. Because KD evaluates a large number of candidates with a fixed structure, it can take advantage of these optimizations. On the other hand, after initially scoring all candidates, BUSL attempts to add them in decreasing order of score to the MLN, thus changing the MLN at almost each step, which slows down the scoring of the structure.

Finally, we checked the importance of adding the edges in Section 4.2.2. This step can, in principle, be avoided by simply producing a fully connected Markov network template. Recall that the goal of this step is to decrease the number of vliterals that could participate together in a clause. In Table 11 we show statistics on the number of TNodes constructed by the algorithm in each of the domains, as well as the proportion of TNodes that end up in the Markov blanket of the head TNode. As can be seen, the number of neighbors of the head TNode in the Markov network template is dramatically smaller than the total number of TNodes discovered. This naturally leads to a smaller number of candidate clauses that need to be considered.

5 Proposed Research

5.1 BUSL as a Transfer Learner

One limitation of BUSL is that it can learn only from scratch and cannot be used to revise the structure of MLNs transferred from a source domain. Therefore an important avenue for future work is to extend BUSL so that it is able to revise an existing MLN structure.

While experimenting with BUSL, we have noticed that most of the training time is spent on evaluating

Predicates	CLL BUSL	CLL KD	AUC BUSL	AUC KD
director	-0.24±0.12	-1.44±0.12	0.91±0.03	0.51±0.01
actor	-0.01±0.00	-0.59±0.08	1.00±0.00	0.88±0.01
movie	-1.66±0.17	-2.42±0.25	0.27±0.00	0.19±0.00
gender	-0.69±0.05	-3.33±0.33	0.48±0.01	0.36±0.00
workedUnder	-0.07±0.00	-0.24±0.02	0.26±0.00	0.10±0.00
genre	-0.18±0.05	-1.10±0.04	0.60±0.05	0.34±0.02
samePerson	-0.03±0.00	-0.03±0.01	1.00±0.00	0.89±0.01
sameMovie	-0.04±0.00	-0.11±0.03	1.00±0.00	0.99±0.00
sameGenre	-0.05±0.00	-0.44±0.23	0.80±0.00	0.63±0.04
sameGender	-0.04±0.00	-0.14±0.07	1.00±0.00	0.99±0.01

Table 7: Per-predicate results from last point on learning curve in IMDB

Predicates	CLL BUSL	CLL KD	AUC BUSL	AUC KD
taughtBy	-0.02±0.00	-0.03±0.00	0.01±0.00	0.00±0.00
courseLevel	-0.82±0.08	-2.95±0.37	0.48±0.03	0.28±0.01
position	-0.16±0.03	-1.33±0.08	0.33±0.03	0.09±0.02
advisedBy	-0.04±0.01	-0.12±0.01	0.02±0.00	0.00±0.00
projectMember	-0.02±0.00	-0.01±0.01	0.00±0.00	0.00±0.00
phase	-0.35±0.03	-0.75±0.13	0.32±0.01	0.26±0.01
tempAdvisedBy	-0.02±0.00	-0.09±0.01	0.01±0.00	0.00±0.00
yearsInProgram	-0.22±0.04	-0.37±0.04	0.16±0.02	0.10±0.01
tA	-0.03±0.00	-0.02±0.00	0.00±0.00	0.00±0.00
student	-0.06±0.02	-1.58±0.10	1.00±0.00	0.59±0.03
professor	-0.07±0.05	-1.51±0.08	0.98±0.01	0.16±0.03
samePerson	-0.03±0.00	-0.06±0.01	1.00±0.00	0.79±0.00
sameCourse	-0.04±0.00	-0.29±0.06	1.00±0.00	0.41±0.00
sameProject	-0.04±0.00	-0.38±0.11	1.00±0.00	0.60±0.00
publication	-0.18±0.02	-0.20±0.02	0.10±0.01	0.05±0.00

Table 8: Per-predicate results from last point on learning curve in UW-CSE

Predicate	CLL BUSL	CLL KD	AUC BUSL	AUC KD
student	-0.01±0.00	-0.81±0.09	1.00±0.00	0.93±0.00
samePerson	-0.02±0.00	-0.01±0.00	0.99±0.00	0.88±0.01
faculty	-0.02±0.00	-2.78±0.13	1.00±0.00	0.56±0.00
project	-0.13±0.01	-0.17±0.02	0.03±0.00	0.02±0.00
courseTA	-0.03±0.00	-0.03±0.00	0.01±0.00	0.01±0.00
courseProf	-0.03±0.00	-0.04±0.01	0.02±0.00	0.01±0.00

Table 9: Per-predicate results from last point on learning curve in WebKB

	Training time			# candidates	
Dataset	BUSL	KD	Speed-up	BUSL	KD
IMDB	4.59	62.23	13.56	162	7558
UW-CSE	280.31	1127.48	4.02	340	32096
WebKB	272.16	772.09	2.84	341	4643

Table 10: Average training time in minutes, average speed-up factor, and average number of candidates considered by each learner.

Data set	IMDB	UW-CSE	WebKB
Average number of TNodes constructed	31.44	70.70	18.83
Average proportion of TNodes in MB of head TNode	0.12	0.14	0.22
Maximum number of TNodes constructed	56	144	28
Maximum size of MB of head TNode	17	41	15

Table 11: Statistics on the average number of TNodes constructed, the average proportion of TNodes that appear in the Markov blanket of the head TNode, the maximum number of TNodes constructed, and the maximum Markov blanket size, over the predicates in all learning runs in each domain.

candidate structures. The TNode and Markov network template construction steps take significantly less time. Thus, one promising way of speeding up BUSL via transfer is by following the same philosophy as with TAMAR discussed in Section 3 where the algorithm starts by diagnosing the transferred structure. After determining the portions of this structure that are still valid in the target domain, and the ones that need to be modified, the number of candidate clauses to evaluate can be decreased by focusing only on the newly-emerging dependencies or independencies among the variables. We can additionally expect to improve the performance in the target domain, especially when data is limited, by biasing learning with the structure learned in the source domain. Below we describe our proposed algorithm in more detail. We will first focus on the case when training in the source domain is carried out using BUSL and will then discuss a way of removing this assumption.

If we use BUSL to learn the structure of an MLN in a source domain, we can output not only the final MLN but also any intermediate structures developed by the algorithm. In particular, we can also record the Markov network templates constructed for each predicate (as described in Section 4.2.2). Let S_P be the Markov network template constructed for predicate P in the source domain. As before, we assume that there is an oracle or a procedure that maps the predicates from the source domain to those of the target domain. Thus, we can translate every S_P to $S_{\hat{P}}$ where $\hat{P}$ is the predicate in the target domain that corresponds to P in the source domain and all other predicates in the template are likewise translated. The proposed revision version of BUSL proceeds as listed in Algorithm 4 where $\hat{\mathcal{P}}$ is the set of predicates in the target domain.

As before, the algorithm considers each predicate $\hat{P}$ in the domain in turn (line 1). For each $\hat{P}$, it inputs $S_{\hat{P}}$, the translated Markov network template constructed for the predicate that corresponds to $\hat{P}$ in the source domain and creates a Markov network template $T_{\hat{P}}$ from the data in the target domain. The differences between these two templates are used to diagnose the clauses transfered from the source domain. In line 3, the algorithm considers each clause c that was constructed in the source domain from a clique in $S_{\hat{P}}$. Recall from Section 4.2.3 that BUSL considers only clauses whose vliterals correspond to TNodes that form a clique in the Markov network template. Therefore, to determine if c could still be valid in the target

Algorithm 4 BUSL for revision

```
Output: $\mathscr{A}$ Set of All constructed candidate clauses
 1: for each $\hat{P} \in \hat{\mathscr{P}}$ do
Input: $S_{\hat{P}}, \mathscr{C}_{\hat{P}}$: Markov network template constructed for the predicate corresponding to $\hat{P}$ in the source
       domain and the set of clauses created from this template, mapped to the predicates in the target
       domain.
 2:   Construct $T_{\hat{P}}$, the Markov network template for predicate $\hat{P}$ in the target domain.
 3:   for each $c \in \mathscr{C}_{\hat{P}}$ do
 4:     $\mathscr{N}$ = empty set of New candidates constructed from c
 5:     if The clique of TNodes in $S_{\hat{P}}$ from which c was constructed is present in $T_{\hat{P}}$ then
 6:       Add c to $\mathscr{N}$
 7:     else if The clique of TNodes in $S_{\hat{P}}$ from which c was constructed is missing some edges in $T_{\hat{P}}$ then
 8:       Consider all possible ways of removing literals from c such that the remaining literals have
          corresponding TNodes that participate together in a clique in $T_{\hat{P}}$
 9:       Add the resulting clauses to $\mathscr{N}$
10:     end if
11:     for each Clause $n \in \mathscr{N}$ do
12:       Collect a list of TNodes in $T_{\hat{P}}$ that are connected with an edge to all of the TNodes that corre-
          spond to the literals in n
13:       Form new clauses from n by adding in all possible ways the literals from the TNodes collected
          above.
14:       Add these new clauses to $\mathscr{A}$
15:     end for
16:     Add all clauses in $\mathscr{N}$ to $\mathscr{A}$.
17:   end for
18:   for each Previously unexamined clique in which the head TNode participates do
19:     Construct clauses as in Section 4.2.3 and add to $\mathscr{A}$.
20:   end for
21: end for
22: Remove duplicate candidates from $\mathscr{A}$.
23: Evaluate candidates in $\mathscr{A}$ and add best ones to final MLN
```

domain, in lines 5-10 the algorithm checks whether the TNodes corresponding to the vliterals of c also form a clique in $T_{\hat{P}}$. If this is the case, c is copied unchanged. Otherwise, the algorithm considers all possible ways of deleting literals from c such that the TNodes corresponding to the remaining literals form a clique in $T_{\hat{P}}$. In lines 11-15 the algorithm checks whether any of the clauses created so far for $\hat{P}$ can be extended by including additional literals. The potential additions are again constrained by the requirement that all literals in a clause correspond to TNodes that form a clique in the template. Finally, in lines 18-20 the algorithm resorts to the complete search for clauses from Section 4.2.3 over any cliques that do not correspond to any previously created clauses. In particular, if the current predicate being considered does not have a corresponding predicate in the source domain, the algorithm skips through the loop in lines 3-17 and proceeds identically to BUSL.

Example: *As an example, suppose the task is to transfer the MLN learned in our movie domain to an academic domain similar to UW-CSE. Table 12 gives a concise list of the predicates in each domain and* *how they map to each other.*

Example Movie Domain		Example Academic Domain
Actor	⇒	Student
Director	⇒	Professor
Movie	⇒	Publication
WorkedFor	⇒	AdvisedBy
no match	⇒	Admin
no match	⇒	TaughtBy

Table 12: Predicates and their mapping in the two toy example domains

Let the current predicate $\hat{P}$ be Student, which is mapped to Actor. We will use the TNodes from Table 6 (page 30) but for simplicity will ignore all TNodes containing more than one vliteral. Suppose Figure 14 gives $S_{\hat{P}}$ and Table 15 gives $\mathscr{C}_{\hat{P}}$, the source clauses mapped to the target domain. Also suppose that Figure 16 gives $T_{\hat{P}}$.

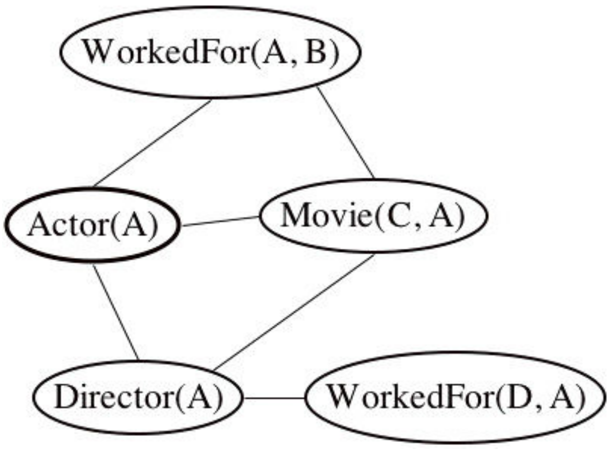


Figure 14: Markov network template for Actor predicate in source domain. The head TNode is outlined in bold.

The TNodes corresponding to the first clause still form a clique in $T_{\hat{P}}$ and thus this clause is copied unchanged to the set of candidates. Considering the second clause, we notice that there is no longer an edge between AdvisedBy(A, B) and Publication(C, A). Therefore, from this clause, we construct the candidates AdvisedBy(A, B) $\Rightarrow$ Student(A) and Publication(C, A) $\Rightarrow$ Student(A). The TNodes corresponding to the literals in the third clause are connected in the target Markov network template, and thus this clause is also copied unchanged. At this point, the algorithm has reached line 11 in the pseudocode and finds that the third clause can be extended by adding the literal Admin(A). In this way, the candidates Student(A) $\vee$ Professor(A) $\vee$ Admin(A) and Student(A) $\vee$ Professor(A) $\vee \neg$ Admin(A) are constructed. Finally, reaching line 18, the algorithm constructs all possible candidates from the new clique formed by TaughtBy(E, A) and Student(A).

The assumption on which Algorithm 4 is based is that if the dependencies among a set of TNodes are

Source Clauses in Example Movie Domain	Mapped Clauses to Example Academic Domain
Movie(C, A) ⇒ Actor(A) ∨ Director(A)	Publication(C, A) ⇒ Student(A) ∨ Professor(A)
WorkedFor(A, B) ∧ Movie(C, A) ⇒ Actor(A)	AdvisedBy(A, B) ∧ Publication(C, A) ⇒ Student(A)
Actor(A) ∨ Director(A)	Student(A) ∨ Professor(A)

Figure 15: Clauses built from the Markov network template in Figure 14 and their corresponding mappings in the target domain.

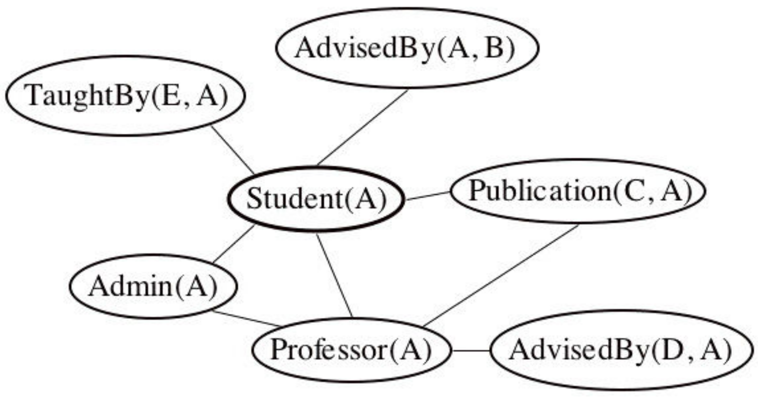


Figure 16: Markov network template for target domain. Head TNode is outlined in bold.

preserved when transferring to the target domain, then the vliterals corresponding to these TNodes will participate in the same specific relationships and therefore the same clauses as in the source domain will give the best performance. Thus, the algorithm copies over clauses that are still backed by a clique in the Markov network template constructed from the target domain and performs only a limited search for new clauses that involve newly-appearing TNodes or cliques.

5.1.1 Markov Network Template Construction

In line 2 we did not specify how the Markov network template in the target domain, $T_{\hat{P}}$, is constructed. One possibility is to follow the same approach as BUSL and ignore any information from the source domain. We expect that this approach will perform well in our current three test domains because, as discussed in Section 4.4, additional data does not introduce a significant number of dependencies that cannot be discovered from a single mega-example. This, however, may not be the case in other tasks, particularly in a game-playing domain, in which a single game may not allow the player to observe all interactions among the relevant entities. In such cases it may be beneficial to use $S_{\hat{P}}$ to bias the construction of $T_{\hat{P}}$. This can be done as follows. Suppose that A and B are two TNodes in the source domain and $\hat{A}$ and $\hat{B}$ are their corresponding TNodes in the target domain. If A and B are not connected by an edge, then we demand stronger evidence of dependence in order to connect $\hat{A}$ and $\hat{B}$. Similarly, if A and B are connected, we require stronger evidence of independence to leave $\hat{A}$ and $\hat{B}$ unconnected.

5.1.2 Constructing a Markov Network Template from an Existing MLN

Next we describe one possible way of removing the assumption that the source MLN was learned using BUSL by developing an algorithm that constructs an approximation of $S_{\hat{P}}$ from the source MLN. This can be done as follows. First the algorithm creates a head TNode as in Section 4.2.1. It then considers all clauses in the source domain that contain literals of $\hat{P}$. Each such clause is grounded with arbitrary constants, ensuring that shared variables are replaced with the same constant. The resulting gliterals are treated as the only true gliterals in a relational database. The same approach as in Section 4.2.1 can now be used to create a set of TNodes. Template construction is completed by connecting with edges TNodes that were constructed from the literals in the same clause.

5.2 Experiments on Additional Domains

We plan to test BUSL's performance in additional domains. Some possibilities include traditional ILP benchmark datasets such as Alzheimer and Mutagenesis used by Landwehr, Kersting, and De Raedt (2005, and others), where the goal is to classify various chemical compounds as mutagens or according to how effective they are against the Alzheimer's disease. These datasets are challenging because they usually contain information about a large number of entities (i.e. molecules), which are described in terms of the relations among their atoms. Another interesting domain is the yeast genome dataset whose properties and challenges are described in detail by Perlich and Merugu (2005). One interesting challenge posed by this dataset is that some relations are weighted. In other words, rather than specifying that two entities are or are not in a relationship, the dataset lists the degree to which they are related.

An alternative possibility is to test BUSL on learning for game control in games such as the ones from the General Game Playing framework (Genesereth & Love, 2005), particularly two-person games that have an opponent and therefore involve uncertain state. This can be cast as a relational problem by introducing predicates for each of the available actions, for all features describing the state, and for all features that could be used to describe the hidden state of the opponent. The task is then to learn an MLN that would effectively predict the probability of success of each action, given the current state. Since the beliefs of the opponent are unobserved, they also need to be inferred. To avoid the intricacies of training MLNs from partially unobservable data, during training, we can simulate different opponents that reveal their hidden state to the player.

5.3 Transfer from Multiple Potential Sources

In all of our transfer experiments, we have assumed that there are only two tasks, a source and a target, and that the two are related *a priori*. A much more interesting and challenging case is one where the learner has compiled a library of previous domains. When faced with a new target task, it first determines which previous tasks are most related to the current one in order to transfer only the knowledge learned from them. This is related to the TC algorithm by (Thrun & O'Sullivan, 1996) in which distance metrics utilized by a nearest neighbor algorithm are transfered only from domains that are deemed similar to the target domain. However, the TC algorithm applies only to the nearest neighbor algorithm and requires that the tasks share the same feature space. Several other researchers have studied related problems. For example, Raina et al. (2006) and Marx et al. (2005) use a set of previous tasks to construct a prior for a logistic regression classifier. Even though this work incorporates knowledge from several source tasks, it merely combines all the previous knowledge without attempting to select the more related source domains and assumes that the tasks share a feature space. Crammer, Kearns, and Wortman (2006) examine the case where a learner has available several source tasks of varying similarity, and give bounds that formalize the tradeoff between acquiring more data from source tasks and misleading the learning algorithm through the inclusion of data from increasingly unrelated sources. However, the authors assume that a disparity matrix that specifies to what extent any two tasks are related is provided to the learner.

In contrast to this previous work, we would like to address the case where various relational domains, described in terms of different predicates, are presented to the learner one by one. In this setting, the learner has to be able to autonomously map source domains to the target domain, determine the relatedness of tasks, and combine knowledge from several sources.

In the simplest multiple-source scenario, the algorithm has learned MLNs in a small set of source domains and wants to transfer the *entire* learned structures from the sources that are most similar to the target domain. This problem can be addressed using existing techniques as follows. First, we use MTAMAR briefly described in Section 3 to find the best way of mapping each of the source structures independently to the target domain. In addition to outputting the mapped structure, this algorithm also provides the WPLL score (described in Section 2.4.2) of this mapped structure in the target domain. We then transfer the mapped structures that achieve the highest WPLL score in the target domain and use RTAMAR or BUSL for revision to further improve the performance of the learned MLN in the target domain.

This simple scenario, however, is not entirely realistic because it is not common to find domains that have significant overlap over all, or most, of their predicates. However, we can reasonably expect to find pairs of domains in which small subsets of the predicates are similar. For example, suppose we would like to use UW-CSE as a source domain and transfer to WebKB. As we mentioned in Section 3.2.2, transferring a complete structure learned in UW-CSE is not beneficial because UW-CSE contains a large number of predicates, many of which have no appropriate counterparts in WebKB. There are, however, several predicates in the two domains that are a fairly good match for each other, as also signified by their names. These include Student, Professor/Faculty, TA/CourseTA, TaughtBy/CourseProf. This example suggests that, for each predicate in the target domain, a more effective transfer learner looks for similar previously encountered predicates and transfers knowledge on a predicate-by-predicate basis. In this way, the learner can seamlessly combine appropriate knowledge from a large number of domains. An important part of a system that transfers predicate-specific knowledge is the ability to judge the similarity between predicates. We next propose one approach for doing this.

5.3.1 Measuring the Similarity Between Predicates

We propose to use the Markov network templates constructed by BUSL in order to measure the similarity between predicates. Recall from Section 4 that BUSL constructs a Markov network template for each predicate P in the domain, and that the goal of this template is to capture the dependencies among vliterals that could potentially be used to build clauses. We plan to use these templates to determine which predicates have a direct influence on the current predicate P: if a vliteral of a predicate appears in the TNodes constituting the Markov blanket of the head TNode, we conclude that this predicate has a direct influence on P.

The main idea of our algorithm for measuring the similarity between predicates is that two predicates are similar if the predicates that have a direct influence on each of them are similar. We envision implementing this idea using an iterative algorithm reminiscent of Gibbs sampling as follows. At each step we recalculate the similarity between a pair of predicates $P1$ and $P2$ based on the currently-estimated similarity between the sets of predicates that have a direct influence over $P1$ and $P2$. One tractable way of estimating the similarity between two sets of predicates $\mathbf{X}$ and $\mathbf{Y}$ is by finding the most similar element of $\mathbf{Y}$ for each element in $\mathbf{X}$ and vice versa and averaging over these $|\mathbf{X}|+|\mathbf{Y}|$ similarities. This approach has complexity $2 \cdot |\mathbf{X}| \cdot |\mathbf{Y}|$. However, we would like to find a formulation that guarantees convergence of the algorithm. The algorithm can be initialized with previously computed similarities among the source predicates. To initialize similarities between source and target predicates, we can use MTAMAR to map a few of the source domains to the target domain and use the choices of this algorithm as initial evidence of similarity.

5.3.2 Selective Knowledge Transfer

Once the similarities between the predicates are computed, for each of the target predicates, the algorithm can transfer the clauses learned for the source predicate(s) estimated to be most similar. These clauses can be mapped to the target domain by replacing each source predicate with its most similar target counterpart. If source domain learning was carried out with BUSL or BUSL for revision, the Markov network templates can be similarly transferred to the target domain.

This approach allows the learner to transfer only those parts of previously learned MLNs that are relevant to the new domain. Moreover, at all times the algorithm maintains an estimate of the similarity between all encountered predicates, thus coming closer to the goal of life-long learning.

5.3.3 Further Extensions

The similarity estimates among the predicates can be used to organize the predicates into clusters. Some work in the direction of predicate clustering has already been done (e.g. Kemp et al., 2006), but this work does not explicitly compute the similarity between predicates which is crucial in the transfer learning setting we are considering. Predicate clustering is particularly useful when there are a very large number of previously-studied predicates. In this case, instead of using MTAMAR to map the target domain to all previous tasks, or to a randomly chosen set of previous tasks, we can select the source tasks so that at least one predicate from each cluster will be initialized in this way. The remaining cluster members can be initialized using the transitivity of the similarity relationship.

5.4 Using BUSL to learn models with functions

One deficiency of BUSL is that, like KD, it cannot currently handle functions. Although functions do not increase the power of first-order logic (Russell & Norvig, 2003), they can be helpful in expressing logical sentences in a more concise way. For example, suppose that we would like to encode the statement that if an agent takes the "Up" action, its *y* coordinate increases by one. Using the *increment* function, this can be easily stated as follows:

$$\text{Action(up)} \wedge \text{CurrentCoords(x, y)} \Rightarrow \text{CurrentCoords(x, increment(y))}$$

Alternatively, we can avoid using functions, thus obtaining a much more awkward statement:

$$\text{Action(up)} \wedge \text{CurrentCoords(x, y)} \wedge \text{ConsecutiveNumbers(y, z)} \Rightarrow \text{CurrentCoords(x, z)}$$

We consider constants to be functions of 0 arity and assume that clauses may contain constants, variables, and functions. Thus this extension to BUSL would be useful even in domains in which there are no functions because it would allow the algorithm to learn clauses that contain constants. For instance, one such helpful clause, encoded in the UW-CSE knowledge base is that visiting professors do not advise any students:

$$\text{Position(P, faculty\_visiting)} \Rightarrow \neg\ \text{AdvisedBy(S, P)}$$

To allow BUSL to learn models with functions, we propose to modify the TNode construction procedure described in Section 4.2.1 so that the resulting TNodes can contain functions. One way of achieving this is by using the technique of least general generalization (LGG), which we discussed in Section 2.2.2.

Suppose that the dataset lists gliterals whose terms may be functions of arity greater than 0. Unlike Algorithm 2 (page 27), which constructs the TNodes and the M_P matrix simultaneously, the approach we propose here first constructs the set of potential TNodes and then fills in the matrix for each P. Let $\mathscr{D}$ be the set of true gliterals in the data. Then, the TNodes are constructed by considering the $\text{LGG}(\mathscr{D},\mathscr{D})$. In other words, for every two gliterals $X,Y \in \mathscr{D}$, if $\text{LGG}(X,Y)$ is defined, we add it to the set of TNodes, $\mathscr{T}$. If $\mathscr{D}$ is too large, we can randomly sample a subset of pairs from it, as is done in GOLEM (Muggleton & Feng, 1992). Because LGG performs the most conservative generalization, not all constants will be replaced by variables, and in this way a more diverse set of TNodes will be created. We expect that an effective Markov network learning algorithm will be even more important in this extension of BUSL because a larger number of TNodes will be created.

Next we describe how the M_P matrix is created for predicate P. For each P, we find a vliteral in $\mathscr{T}$ that could serve as a head TNode (i.e. it is completely variablized with unique variables). If there is no such vliteral, we simply create it and add it to $\mathscr{T}$. For each grounding G_P of P, we go through $\mathscr{T}$ and check whether there is at least one gliteral connected to G_P that could be transformed into the current member t of $\mathscr{T}$ so that any variablizations are consistent (i.e. if two constants are the same, they are replaced with the same variables). If such a gliteral exists, we set $M_P[G_P][t]$ to 1. Otherwise, it remains set to 0.

5.5 Using BUSL to learn other SRL models

Our promising results with BUSL suggest that similar bottom-up techniques can be used to learn other SRL models. For example, BUSL can be adapted to learn Bayesian logic programs (BLPs), which are first-order analogs of Bayesian networks. In a BLP, the structure consists of Horn clauses which describe the dependencies among the literals. The premises of each clause become the parents of the head in the Bayesian network defined by the BLP. The present BLP learning algorithm by Kersting and De Raedt (2001) follows the top-down paradigm of generating and scoring candidate structures. Instead, we can proceed as in BUSL and first create a Markov network template that specifies which literals are conditionally independent of each other and therefore could not possibly be parents of each other. This can be done in exactly the same way as in BUSL. Note that even though BLPs are first-order analogs to Bayesian networks, which are directed graphical models, as opposed to Markov networks, which are undirected, it is still useful to first construct a Markov network template because the lack of arrows makes this model easier to learn. Directionality is added by the constraint that candidates can only be Horn clauses, i.e. there will be directed edges from each of the premises in a Horn clause to the conclusion. Alternatively, a Bayesian network template can be directly constructed over the set of TNodes by replacing GSMN with an algorithm such as that of Margaritis and Thrun (2000) that learns Bayesian network structure.

6 Conclusions

Markov logic networks are a recently developed formalism that combines the expressivity of first-order logic with the flexibility of probabilistic reasoning and subsumes many popular SRL models. In order to take advantage of the power of MLNs, however, we need methods for learning their structure, or the clauses that compose them. In this proposal we have presented two approaches for improving MLN structure learning. The first one, RTAMAR autonomously diagnoses and revises an MLN learned in a source domain in order to improve performance in a target domain. Our second algorithm, BUSL approaches the problem of structure learning in a novel, bottom-up way and significantly improves learning from scratch over the current state-of-the-art learner. In our short-term future work we propose to extend BUSL to perform revision so that it can be used like RTAMAR for transfer learning and to test it in additional domains. Our longer-term research goals include developing a much more general transfer learning framework in which the learner must independently decide which from a set of previous learning experiences are most similar to the target task and transfer knowledge only from them. We would also like to extend BUSL to handle functions and to learn the structure of other SRL models.

7 Acknowledgement

We would like to thank Tuyen Huynh for developing MTAMAR and for his help with running some of the experiments in Section 3. We are also grateful to the Alchemy team at the University of Washington for their great help with using the Alchemy package and for making this package publicly available. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and managed by the Air Force Research Laboratory (AFRL) under contract FA8750-05-2-0283.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, AFRL, or the United States Government. Most of the experiments were run on the Mastodon Cluster, provided by NSF Grant EIA-0303609.

References

Abbeel, P., Koller, D., & Ng, A. Y. (2006). Learning factor graphs in polynomial time and sample complexity. *Journal of Machine Learning Research*, *7*, 1743–1788.

Ando, R. K., & Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, *6*, 1817–1853.

Banerjee, B., & Stone, P. (2007). General game learning using knowledge transfer. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-2007)*, Hyderabad, India.

Berger, A. (1996). A brief maxent tutorial. Available at `http://www.cs.cmu.edu/afs/cs/user/aberger/www/html/tutorial/tutorial.ht%ml`.

Bonilla, E. V., Williams, C. K. I., Agakov, F. V., Cavazos, J., Thomson, J., & O'Boyle, M. F. P. (2006). Predictive search distributions. In *Proceedings of 23rd International Conference on Machine Learning (ICML-2006)*, Pittsburgh, PA.

Bromberg, F., Margaritis, D., & Honavar, V. (2006). Efficient Markov network structure discovery using independence tests. In *Proceedings of the Sixth SIAM International Conference on Data Mining (SDM-06)*, Bethesda, MD.

Bunescu, R., & Mooney, R. J. (2007). Statistical relational learning for natural language information extraction. In Getoor, L., & Taskar, B. (Eds.), *Statistical Relational Learning*. MIT Press, Cambridge, MA. To appear.

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*, 41–75.

Cohen, P. R., Chang, Y., & Morrison, C. T. (2007). Learning and transferring action schemas. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-2007)*, Hyderabad, India.

Crammer, K., Kearns, M., & Wortman, J. (2006). Learning from multiple sources. In *Advances in Neural Information Processing Systems 18: Proceedings of the 2006 Conference*.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (1998). Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 509–516, Madison, WI.

Davis, J., Ong, I., Struyf, J., Costa, V. S., Burnside, E., & Page, D. (2007). Change of representation for statistical relational learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-2007)*, Hyderabad, India.

De Raedt, L., & Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, *26*, 99–146.

Della Pietra, S., Della Pietra, V. J., & Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 380–393.

Genesereth, M., & Love, N. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, *26(2)*.

Getoor, L., & Taskar, B. (Eds.). (2007). *Statistical Relational Learning*. MIT Press, Cambridge, MA. To appear.

Getoor, L., & Diehl, C. P. (2005). Link mining: A survey. *SIGKDD Special Issue on Link Mining*, *7(2)*.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. In Džeroski, S., & Lavrač, N. (Eds.), *Relational Data Mining*.

Goldsmith, J., & Sloan, R. H. (2005). New Horn revision algorithms. *Journal of Machine Learning Research*, *6*, 1919–1938.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003).*

Hammersley, J., & Clifford, P. (1971). Markov fields on graphs and lattices. Unpublished manuscript.

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Tech. rep. MSR-TR-95-06, Microsoft Research, Redmond, WA.

Kautz, H., Selman, B., & Jiang, Y. (1997). *A General Stochastic Approach to Solving Problems with Hard and Soft Constraints*, Vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 573–586. American Mathematical Society.

Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., & Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, Boston, MA.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with Bayesian networks. In *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, pp. 118–131, Strasbourg, France.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. In *Proceedings of 22nd International Conference on Machine Learning (ICML-2005)*, Bonn,Germany.

Kok, S., Singla, P., Richardson, M., & Domingos, P. (2005). The Alchemy system for statistical relational AI. Tech. rep., Department of Computer Science and Engineering, University of Washington. http://www.cs.washington.edu/ai/alchemy.

Landwehr, N., Kersting, K., & De Raedt, L. (2005). nFOIL: Integrating naive Bayes and FOIL. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-2005)*, Pittsburgh, PA.

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Lee, S., Ganapathi, V., & Koller, D. (2006). Efficient structure learning of Markov networks using L_1-regularization. In *Advances in Neural Information Processing Systems 18: Proceedings of the 2006 Conference*.

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematic Programming*, *45*(3), 503–528.

Margaritis, D., & Thrun, S. (2000). Bayesian network induction vial local neighborhoods. In *Advances in Neural Information Processing Systems 12*.

Marx, Z., Rosenstein, M. T., Kaelbling, L. P., & Dietterich, T. G. (2005). Transfer learning with an ensemble of background tasks. In *Proceedings of NIPS-2005 Workshop on Inductive Transfer: 10 Years Later*.

Mihalkova, L., Huynh, T., & Mooney, R. J. (2007). Mapping and revising Markov logic networks for transfer learning. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-2007)*, Vancouver, BC. To appear.

Mihalkova, L., & Mooney, R. J. (2006). Transfer learning with Markov logic networks. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, PA.

Mihalkova, L., & Mooney, R. J. (2007). Bottom-up learning of Markov logic network structure. In *Proceedings of 24th International Conference on Machine Learning (ICML-2007)*, Corvallis, OR. To appear.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing Journal*, *13*, 245–286.

Muggleton, S., & Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S. (Ed.), *Inductive Logic Programming*, pp. 281–297. Academic Press, New York.

Niculescu-Mizil, A., & Caruana, R. (2005). Learning the structure of related tasks. In *Proceedings of NIPS-2005 Workshop on Inductive Transfer: 10 Years Later*.

Niculescu-Mizil, A., & Caruana, R. (2007). Inductive transfer for bayesian network structure learning. In *Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS-07)*.

Paes, A., Revoredo, K., Zaverucha, G., & Costa, V. S. (2005). Probabilistic first-order theory revision from examples. In *Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)*, Bonn, Germany.

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

Perlich, C., & Merugu, S. (2005). Multi-relational learning for genetic data: Issues and challenges. In Džeroski, S., & Blockeel, H. (Eds.), *Fourth International Workshop on Multi-Relational Data Mining (MRDM-2005)*.

Plotkin, G. D. (1970). A note on inductive generalisation. *Machine Intelligence*, *5*, 153–163.

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

Popescul, A., Ungar, L. H., Lawrence, S., & Pennock, D. M. (2003). Statistical relational learning for document mining. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM-03)*.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Raina, R., Ng, A. Y., & Koller, D. (2006). Constructing informative priors using transfer learning. In *Proceedings of 23rd International Conference on Machine Learning (ICML-2006)*, Pittsburg, PA.

Ramachandran, S., & Mooney, R. J. (1998). Theory refinement for Bayesian networks with hidden variables. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pp. 454–462, Madison, WI. Morgan Kaufmann.

Richards, B. L., & Mooney, R. J. (1995). Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, *19*(2), 95–131.

Richards, B. L., & Mooney, R. J. (1992). Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 50–55, San Jose, CA.

Richardson, M. (2004). *Learning and Inference in Collective Knowledge Bases*. Ph.D. thesis, University of Washington.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2 edition). Prentice Hall, Upper Saddle River, NJ.

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

Spirtes, P., Glymour, C., & Scheines, R. (2001). *Causation, Prediction, and Search*. MIT Press.

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proceedings of 18th Conference on Uncertainty in Artificial Intelligence (UAI-2002)*, pp. 485–492, Edmonton, Canada.

Taylor, M. E., Stone, P., & Liu, Y. (2005). Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-2005)*.

Taylor, M. E., Whiteson, S., & Stone, P. (2007). Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-07)*.

Thomas, B. (2003). Bottom-up learning of logic programs for information extraction from hypertext documents. In *Proceedings of 7th European Conference of Principles and Practice of Knowledge Discovery in Databases (PKDD-03)*, pp. 435–446. Springer Verlag.

Thrun, S., & O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. In Saitta, L. (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML-96)*.

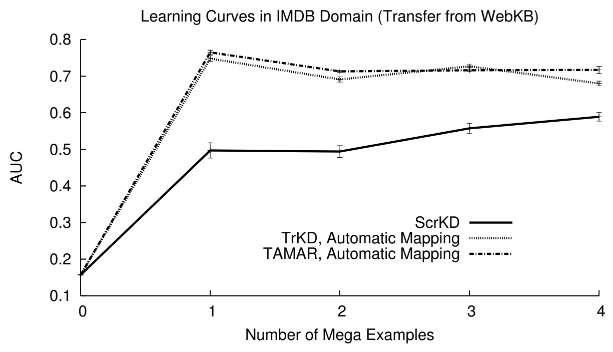
Thrun, S., & Pratt, L. (Eds.). (1998). *Learning to Learn*. Kluwer Academic Publishers, Boston.

Torrey, L., Walker, T., Shavlik, J., & Maclin, R. (2005). Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the 16th European Conference on Machine Learning (ECML-05)*, Porto, Portugal.

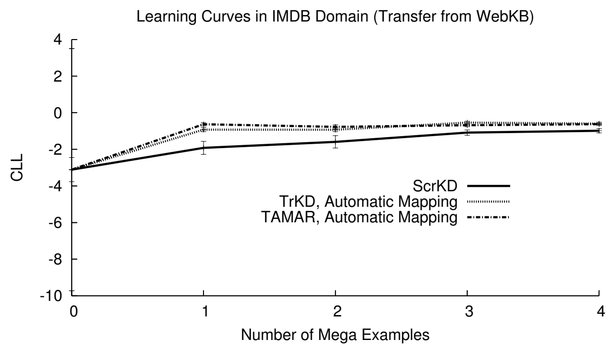
Zelle, J. M., Mooney, R. J., & Konvisser, J. B. (1994). Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML-94)*, pp. 343–351, New Brunswick, NJ.

8 Appendix 1

Figures 17 to 21 present complete learning curves for the results presented in Section 3.2.3. The zeroth points are obtained by testing the performance of the MLN provided to the learner at the start.

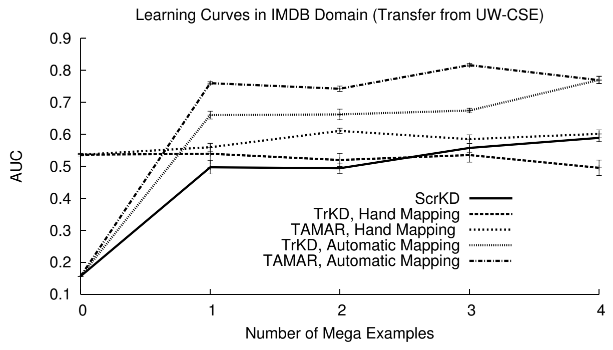


a)

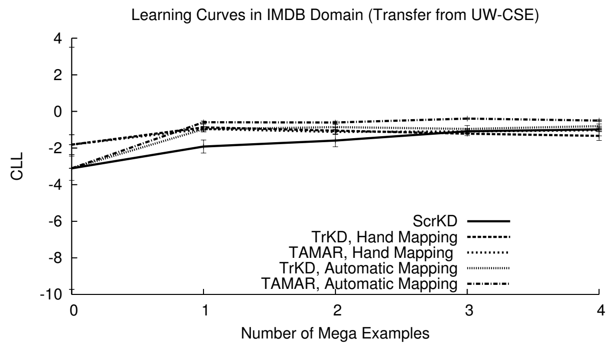


b)

Figure 17: Learning curves in WebKB → IMDB for a) AUC and b) CLL.

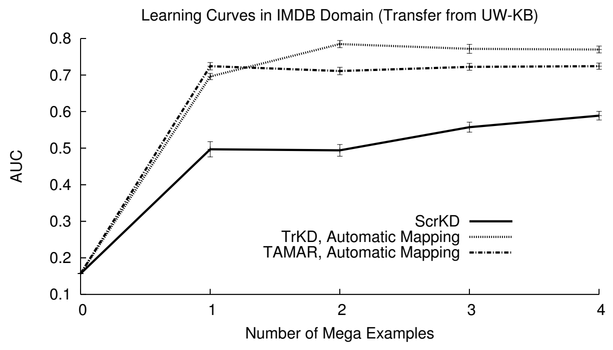


a)

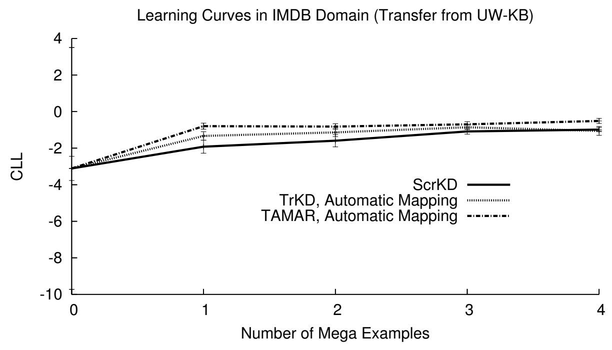


b)

Figure 18: Learning curves in UW-CSE → IMDB for a) AUC and b) CLL. Here we additionally tested the performance of systems that do not use the automatic mapping but are provided with an intuitive hand-constructed mapping that maps Student → Actor, Professor → Director, AdvisedBy/TempAdvisedBy → WorkedFor, Publication → MovieMember, Phase → Gender, and Position → Genre.

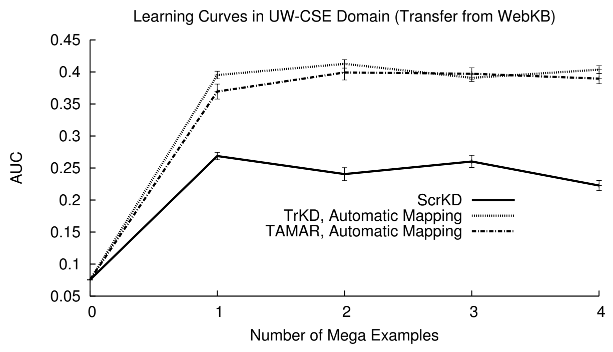


a)

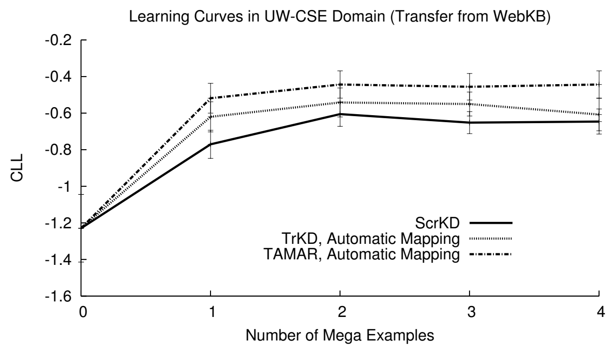


b)

Figure 19: Learning curves in UW-KB → IMDB for a) AUC and b) CLL.

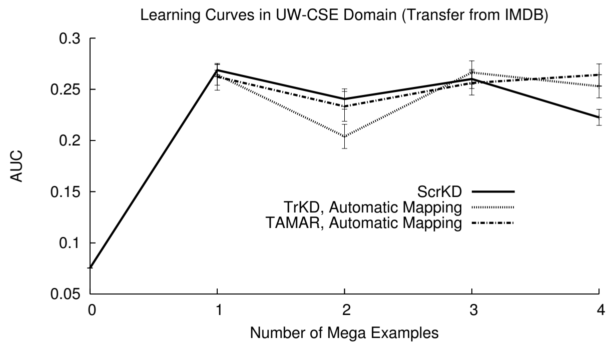


a)

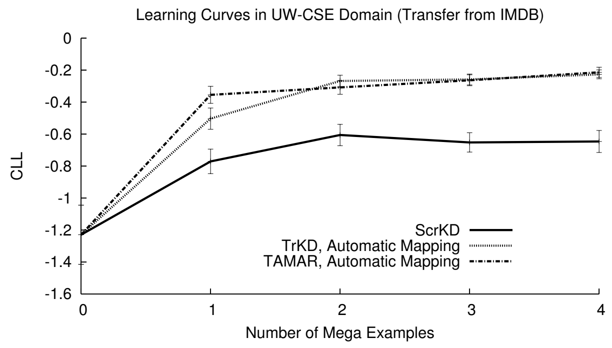


b)

Figure 20: Learning curves in WebKB → UW-CSE for a) AUC and b) CLL.



a)



b)

Figure 21: Learning curves in IMDB → UW-CSE for a) AUC and b) CLL.