Transfer and Structure Learning in Markov Logic Networks

by

David A. Moore

Andrea Danyluk, advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Computer Science

WILLIAMS COLLEGE
Williamstown, Massachusetts

Acknowledgements

I'd like to thank my advisor, Andrea Danyluk, for all of her support, advice, and insight, as well as for her willingness and miraculous ability to find the time to supervise a thesis while simultaneously serving as the Dean of the Faculty. I'd also like to thank Tom Murtagh, my second reader, for his helpful comments and insightful questions. I'm especially grateful to Mary Bailey for setting up the cluster on which all of my experiments were run (and for her foregiveness of my constant abuse of the Unix lab machines), and to Jeannie Albrecht for donating large quantities of high-powered hardware to the effort. Many thanks to Jesse Davis for providing me with his implementation of DTM as well as the WebKB and Yeast datasets, and for giving generously of his time through many helpful conversations. Last but not least, I'd like to thank my parents for their love, encouragement, and support, without which this thesis would never have been written.

Abstract

Markov logic networks are a recently developed knowledge representation capable of compactly representing complex relationships and handling uncertainty in a principled manner. The deep transfer algorithm of Davis and Domingos proposes a method for learning the structure of an MLN by incorporating cross-domain knowledge — for example, using the relationships between yeast proteins to inform predictions about relationships in the movie business. This thesis explores several questions related to this algorithm and to cross-domain transfer in general. I first develop methods for simultaneous transfer from multiple domains, and propose that much of the success of deep transfer may be explained by a small number of commonly occurring structural properties, e.g. symmetry and transitivity. I then demonstrate empirically that deep transfer often performs best when using the target domain as its own transfer source, e.g. the most effective way to predict relationships in the movie business is to examine other relationships in the movie business. This suggests a reinterpretation of the deep transfer algorithm as a method for single-domain structure learning. Finally, I describe a new algorithm for cross-domain transfer which transfers a more specific form of structure than the second-order cliques used by Davis and Domingos, and show empirically that my approach performs competitively to the deep transfer algorithm, despite using a less sophisticated form of learning in the source domain.

Contents

List of Tables

Glossary

AUC	Area Under the precision-recall Curve. Measures the extent to which a learned model is able to correctly classify the ground atoms of a dataset.
CLL	Conditional Log-Likelihood. Measure the extent to which a learned model predicts accurate probabilities for the ground atoms of a dataset.
CSGL	Clique Scoring with Greedy seLection. A variant of the DTM algorithm used for structure learning, consisting of DTM applied without refinement within a single domain.
DTM	Deep Transfer for Markov logic networks. A method for "deep" transfer learning which explicitly represents domain-independent knowledge as second-order cliques.
ILP	Inductive Logic Programming. A collection of learning methods used to generate knowledge bases, consisting of formulas of first-order logic, which model a given dataset.
LHL	Learning through Hypergraph Lifting. A recent MLN structure learning algorithm which shares many traits with CSGL.
MLN	Markov Logic Network. A knowledge representation providing a probabilistic extension of first-order logic.
MSL	MLN Structure Learning. A common structure learning algorithm for MLNs; used as a baseline for transfer.
SRL	Statistical Relational Learning. The general study of representations and learning methods for handling uncertainty in relational domains, in particular including the development of MLNs and associated algorithms.
STM	Simple Transfer for Markov logic networks. A method for "deep" transfer learning which explicitly represents domain-independent knowledge as second-order formulas.
TAMAR	Transfer via Automatic Mapping And Revision. A transfer learning method which creates a direct mapping between the predicates of different domains.
WPLL	Weighted Pseudo Log-Likelihood. An approximation of likelihood, used as a metric during the optimization phase of MLN structure- and weight-learning algorithms.

1

Introduction

Traditional inductive learning derives propositional knowledge from propositional data. If we want to predict whether it will rain today (the proposition `Raining`), we might start by noting various aspects of the current weather, for example whether it is `Cloudy`, `Humid`, `Warm`, and so on. Taken together, the truth values of these propositions comprise a feature vector for which we want to predict a binary classification; namely, will it rain? Given a set of data points accumulated over many days of weather observations, there exist a wide variety of methods for learning how to make this prediction. Such methods often attempt to learn a probabilistic model: we want to predict the probability of `Raining`, conditioned on the evidence of the other features. If today is cloudy, and cloudy days have historically tended also to be rainy, then today might also be rainy.

Propositional learning methods have achieved great success in part because many learning tasks can be easily represented in propositional form. Often, however, we are presented with data that is inherently *relational*, where the examples are not independent and we need to use our knowledge about the relationships and dependencies in the data to make accurate predictions. For example, the essential insight of Google's PageRank algorithm (Page et al., 1998) was to realize that determining the relative authority of a web page requires an understanding of that page's place in the link structure of the Web as a whole; to predict relevance it is necessary to consider the relationships between pages in addition to their individual contents.

Networks are inherently relational — if we know something about one component of a network, we can make predictions about the components it is connected to. For example, web pages tend to link to other pages which cover similar topics. This idea extends beyond the Internet: social networks (both online and offline), food webs, political alliances, sensor networks, citation graphs, gene regulatory networks, buyer-seller relationships in a market, etc. are among the many systems which can be viewed with a network structure and reasoned about relationally (though the precise nature of the relationships is different in every case). Since many of these systems are generated by complex underlying phenomena, it is often difficult for a human knowledge engineer to precisely specify all of the rules necessary to reason about the relationships within any particular domain, especially if the domain is one where people have few pre-existing intuitions (e.g. predicting the interactions between yeast proteins). By sifting through data to uncover these rules automatically, machine learning techniques can help us reason and solve problems in an increasingly interconnected world.

One approach to capturing these sorts of relationships in data is to use first-order logic as the knowledge representation, since this allows the creation of predicates expressing arbitrary relationships between objects in the world. Logical representations of relational properties are preferable to graph-based approaches because logic expresses the general structure of relationships and does not require a commitment to some particular set of objects as the nodes of a network. For example, the formula `Friends(x,y)` $\wedge$ `Friends(y,z)` $\implies$ `Friends(x,z)` expresses a general constraint on friendship relationships, i.e. that they must be transitive, which it asserts to hold for any friendship graph regardless of the specific people involved. Inductive logic programming (ILP) (Lavrac and Dzeroski, 1994) learns logical hypotheses using the language of Horn-clause logic, a restricted form of first-order logic. Given a set of facts about the world, ILP searches for a set of formulas which fit the known facts as closely as possible; once found, the formulas can then be used to infer the truth values of arbitrary statements. In this way the hypotheses learned by an ILP system divide the world neatly into true statements, false statements, and statements for which the system is unable to make a predication.

A major drawback to logic-based representations is that logic taken alone is extraordinarily brittle; a single counterexample renders a formula just as false as a million counterexamples would. While this is a valuable trait in mathematics, where one contradiction renders an entire system hopelessly inconsistent, it is less helpful when using logic to model empirical data, which is often noisy, and which more importantly often contains useful tendencies which do not quite rise to the level of infallible mathematical truth. For example, the formula describing the transitivity of friendship (given above) does not hold universally; there are many counterexamples. However, it is still true with some probability greater than chance, and we would like to be able to represent this sort of tendency. This is the goal of *statistical relational learning*: to find approaches which allow us to represent relationships probabilistically, and therefore bring to bear the power of statistical learning in relational domains.

1.1 Markov Logic Networks

Many approaches to statistical relational learning have been proposed (see Getoor and Taskar (2007)), but this thesis is concerned primarily with the particular framework known as *Markov logic networks* (Richardson and Domingos, 2006), discussed more fully in Chapter 2. A Markov logic network (MLN) is simply a set of first-order logical formulas with associated weights. Though the weights are not probabilities themselves, they can be interpreted as inducing a probability distribution over the set of all possible worlds, where a world is an assignment of truth values to every grounded predicate in the domain. The interpretation stems from viewing the MLN along with a set of domain constants as defining a Markov network (Pearl, 1988), a form of probabilistic graphical model. Thus an MLN allows us to assign probabilities to arbitrary logical statements, by summing the individual probabilities of all possible worlds in which each statement holds. Although this form of exact inference is intractable in practice, approximate inference can be performed efficiently (Richardson and Domingos, 2006).

Learning an MLN representation for a domain can be reduced to two problems: weight learning and structure learning. Weight learning considers a set of formulas and determines appropriate weights from the available data, while structure learning attempts to find formulas which capture useful information about the relationships which hold within a domain. For example, given data describing a social network, structure learning might generate the formula $\texttt{Friends(x,y)} \wedge \texttt{Friends(y,z)} \implies \texttt{Friends(x,z)}$ which asserts the transitivity of friendship; we could then apply weight learning to quantify precisely to what degree this formula actually holds in our data.

1.2 Deep Transfer

Learning structure in general domains is a difficult but important problem, for which several algorithms have been proposed, including MSL (Kok and Domingos, 2005), BUSL (Mihalkova and Mooney, 2007), ISL (Biba et al., 2008), and LHL (Kok and Domingos, 2009). One approach to improving structure learning is to apply *transfer learning* (surveyed in Torrey and Shavlik (2009)). In general, transfer learning attempts to use knowledge gained in the course of learning to perform a source task to aid the learning process for a related target task. Davis and Domingos (2009) introduce an algorithm known as deep transfer, or DTM, which transfers MLN structure across domains. DTM abstracts the first-order formulas describing a source domain into second-order cliques, which act as templates for the formulas of the domain and represent particular types of higher-level structural regularities. These cliques constitute a form of domain-independent knowledge which can be used in the target domain to bias the structure search process towards more fruitful areas of the search space.

Davis and Domingos demonstrate empirically that their algorithm indeed does often lead to improved structure learning performance, as measured by the ability to predict the truth values of unknown propositions from biological, hypertext, and social domains. They further claim that the effectiveness of deep transfer is validated by the intuitively plausible nature of the regularities it discovers: the top-ranking cliques tend to represent well-known relationships such as symmetry, transitivity, and homophily (the "birds of a feather flock together" property, which states that objects sharing similar properties tend to be related in some way).

1.3 Motivation and Goals

Although deep transfer has been shown to be effective in some cases at improving structure learning performance, the algorithm has not been extensively studied and many aspects of its behavior are poorly understood. Questions include:

- Is it useful to transfer from multiple domains at once?
- Since it appears that a small set of cliques scores highly across a wide set of domains (as mentioned above, most of these cliques represent intuitive regularities such as transitivity), how much does the choice of a particular source domain matter? Could it be that there is some set of cliques which are widely useful across many domains?
- If source domain choice does matter, how can we select the best domain to transfer from?
- Can DTM be modified to transfer knowledge in forms other than second-order cliques, and is it useful to do so?

This thesis attempts to shed light on these questions.

1.4 Contributions and Organization

This main contributions of this thesis are as follows. First, I consider several methods for combining cliques from multiple domains and evaluate their effectiveness. Additionally, I try two approaches for combining cliques from four different domains in an attempt to find a single set of widely useful cliques, with some (qualified) success.

Next, motivated by the problem of finding the best source domain for transfer, I propose the method of *self-transfer*, which applies the DTM algorithm using a single domain as both the source and target of the transfer process. The intuition is that no other domain is as similar to a domain as it is to itself. I find that self-transfer leads to learning performance generally equalling or exceeding the best cases of transfer from external domains. This suggests that we can think of DTM as performing a novel form of structure learning, which I call "clique scoring with greedy selection" (CSGL). I compare CSGL to cross-domain DTM as well as to other algorithms for structure learning in MLNs, and point out similarities between CSGL and another recent approach based on hypergraph pathfinding (Kok and Domingos, 2009).

Finally, I develop a new algorithm for transfer learning in MLNs which uses second-order formulas rather than cliques as the domain-independent knowledge representation; I call this algorithm "simple transfer". I evaluate this method empirically and discuss the relative merits of this approach compared to those of deep transfer.

Chapter 2 discusses related work and provides a detailed introduction to Markov logic and the deep transfer algorithm. Chapter 3 describes the datasets and experimental setup used in the chapters which follow. Chapter 4 discusses methods for simultaneous transfer from multiple sources, and attempts to find a set of cliques which are useful as "universal" transfer sources. Chapter 5 discusses self-transfer/CSGL and evaluates it against other structure learning approaches. Chapter 6 introduces simple transfer and tests it empirically as well. Chapter 7 states conclusions and describes ideas for future work.

2

Background and Related Work

As introduced in the previous chapter, statistical relational learning (SRL) aims to develop learning algorithms capable of dealing with uncertainty in relational domains. This is an important goal, because many real-world learning tasks are neither clean nor propositional. Techniques for SRL have generally involved some combination of ideas from probability theory and logic, where probability is used to reason about uncertainty in a principled way, and logic provides a tool to represent complex abstract relationships. This thesis focuses on one particular representation, Markov logic networks (MLNs), and several algorithms associated with them.

2.1 Logic and Terminology

In first-order logic, a *constant* is a symbol which refers to a particular object in the domain, e.g. `Brando` might refer to Marlon Brando in a movie domain. This thesis makes the *unique names assumption* that different constants never refer to the same object. *Variables* stand in place of constants for the purpose of quantification; a *term* is a constant or a variable. A *predicate* represents a relation on objects in the domain and is either true or false for any given list of objects; the *arity* of a predicate is the number of arguments it takes (equivalently, the number of objects that it relates). Predicate arguments are typed. An *atom* is a predicate applied to a list of terms, and a *ground atom* is an atom in which all of the terms are constants. A *clause* is a disjunction of atoms, any of which may be negated; a *formula* may also include conjunction and implication as well as universal and existential quantifiers. In this thesis, standalone clauses are assumed to be implicitly universally quantified. A *knowledge base* is a set of formulas or clauses. A *ground formula* is a formula in which all of the atoms are ground atoms.

2.2 Markov Logic Networks

Traditional probabilistic models — including graphical models such as Bayesian or Markov networks (Pearl, 1988) — describe the joint probability distribution of a set of propositions. In other words, given some set of n propositions, e.g. `Cloudy`, `Humid`, `Raining`, etc., these models assign a probability to each of the 2^n possible states: cloudy, humid, and raining; cloudy humid, and not raining, and so on. From the joint distribution, we can calculate a probability for any logical formula by summing over the probabilities of the states in which that formula holds true. A representation of the joint distribution therefore allows us to answer arbitrary questions about the set of propositions, e.g. what is the probability that it is `Raining` given that it is `Cloudy`?

Any relational domain can be *propositionalized*, by first choosing a set of constants representing some set of objects taken from the domain, and then considering the set of all ground atoms which may be created by instantiating the predicates of the domain with the chosen constants in all possible ways. Each ground atom can then be viewed as a proposition, and the relationships between ground atoms modelled as described above. For example, in a social network domain this would mean choosing a fixed set of people and examining only the attributes of and relationships between those particular people. However, because the model is specific to the particular set of constants chosen, this approach captures only a slice of the overall relational structure. For example, if we consider a social network domain and take as our objects the members of the US Senate, then a propositional approach would model the probability that (for example) John Kerry and Lindsey Graham are friends (i.e. the proposition `Friends(Kerry, Graham)`), but would have nothing whatsoever to say about the members of the House of Representatives. This is because such models define the probabilities of particular propositions, but the higher-order tendencies we want to model—perhaps legislators who have written bills together are more likely to be friends—can only be expressed in a representation which explicitly generalizes the constants, such as first-order logic.

Markov logic networks (MLNs) (Richardson and Domingos, 2006) aim to provide a solution to this problem, by generalizing both probabilistic graphical models and first-order logic to create a flexible representation for statistical relational knowledge. Instead of representing the joint distribution over a set of propositions, an MLN can be seen as defining a *template* for creating such a distribution, given any particular set of constants (the resulting propositional distributions are represented as Markov networks; see Section 2.2.1).

0.7	`Actor(A)` $\implies$ $\neg$`Director(A)`
1.2	`Director(A)` $\implies$ $\neg$`WorkedFor(A,B)`
1.4	`Credits(T,A)` $\wedge$ `WorkedFor(A,B)` $\implies$ `Credits(T,B)`

Table 2.1: Example Markov Logic Network (Mihalkova, 2009)

An MLN consists of a set of first-order logical formulas with weights attached. These formulas signify general regularities which we expect to remain relevant for any reasonable sampling of objects from the domain. For example, Table 2.1 defines a simple MLN over a movie domain which asserts several tendencies: actors are usually not also directors, directors usually don't work under anyone else, and if a person's name appears in the credits of a film, then the people they've worked under will generally also appear in the credits of that film.

Given a world defined by some particular set of objects from the domain, the formulas and weights of an MLN serve to define a probability distribution over all possible states of that world (where a state is just an assignment of truth values to all possible ground atoms formed using that set of constants), in which the probability of any particular state depends on how many formulas it satisfies and the weights of those formulas. Formally, the probability of any particular state x is given by

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{2.1}$$

(Richardson and Domingos, 2006) where w_i is the weight of the ith formula and $n_i(x)$ gives the number of true groundings of the ith formula in state x, i.e. the number of distinct ways in which the variables in the formula can be substituted by constants in order to form a true statement. Z is a normalization term equal to the sum over all world states $Z = \sum_y \exp\left(\sum_i w_i n_i(y)\right)$. Note that the weights are not themselves probabilities, although we use them to calculate probabilities. The weight of a formula can be interpreted as the difference in log probabilities (i.e. the log odds) between states in which the formula is true and those in which it is not, all else held equal. Thus, formulas with positive weights serve to increase the probability of the states in which they are satisfied, formulas with negative weights decrease the probability of states in which they are satisfied, and zero-weight formulas have no effect. Negating a formula is equivalent to flipping the sign of its weight (this is non-obvious, but can be easily shown). Also note that a state in which no formulas are satisfiable will have probability $\frac{1}{Z}\exp(0) = \frac{1}{Z}$; such a state becomes increasingly unlikely when Z is large (that is, when there are many other states which do satisfy the formulas).

We have defined an MLN as a knowledge base consisting of arbitrary first-order formulas, but it is also possible to restrict the allowable formulas to clauses only. An MLN with this restriction is said to be in *clausal form.* Note that although the formulas of an MLN can always be converted into a set of clauses by writing each formula in conjunctive normal form and splitting the result into its separate clauses[1], the resulting clausal-form MLN is not in general equivalent to the original. To see this, consider the case of a single CNF formula with large weight, which is split into two clauses by the conversion process. The original MLN assigns either very low or very high probability to any given state of the world, depending on whether the single original formula is satisfied, while the clausal form MLN also recognizes an intermediate state in which only one of the clauses is satisfied. It follows that the dependencies of the original formula are not entirely captured by its factorization into clauses. Despite their limitations, however, clausal form MLNs are often more convenient to work with than arbitrary MLNs and are generated or required by many algorithms, including each of those discussed below.

Markov logic networks are an extremely general and powerful representation. For example, it can be shown that any probability distribution over discrete or finite-precision numeric variables can be represented as an MLN, and that first-order logic is the special case of Markov logic obtained in the limit when all weights are equal and tend to infinity (Richardson and Domingos, 2006).

2.2.1 Interpretation as Markov Networks

As described above, Markov logic networks can be interpreted as defining templates for the construction of Markov networks (also called Markov random fields), which are a type of undirected graphical model in which the edges specify dependency relationships between nodes (Pearl, 1988). Given an MLN and a set of constants, the structure of the corresponding ground Markov network is given by an undirected graph constructed by two principles:

1. The graph contains a node for each ground atom (i.e., every possible instantiation of an atom from the MLN using the available constants).

2. Two nodes are connected by an edge if and only if their corresponding ground atoms appear together in one of the ground formulas created by instantiating the MLN formulas with the chosen constants.

[1]We can remove universal quantifiers, because we assume all standalone clauses to be universally quantified. Existential quantifiers can be removed by Skolemization, which in its simplest case involves simply creating a new constant to represent the object which was posited by the quantifier.

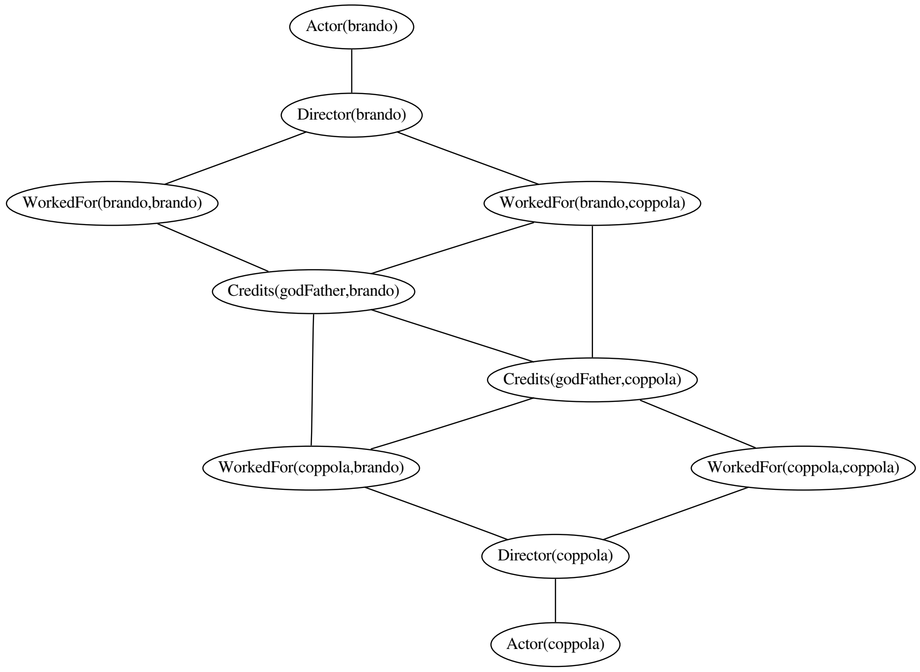
Actor(brando)
Director(brando)
WorkedFor(brando,brando)
WorkedFor(brando,coppola)
Credits(godFather,brando)
Credits(godFather,coppola)
WorkedFor(coppola,brando)
WorkedFor(coppola,coppola)
Director(coppola)
Actor(coppola)

Figure 2.1: The ground Markov network corresponding to the MLN of Table 2.1 (Mihalkova, 2009).

Applied to the movie domain given in Table 2.1 with constants `brando`, `coppola`, and `godFather`, this process produces the result shown in Figure 2.2.1. Note that a state of the world corresponds to an assignment of truth values to every node in the graph. In particular, the truth value of a ground atom is conditionally independent of all ground atoms it is not connected to, given the truth values of the atoms which it is connected to. This is because two atoms which never appear together in the same formula can have at most an indirect relationship to each other. The set of nodes connected to a particular node is known as the Markov blanket of that node; conditional independence given the Markov blanket is the defining property of a Markov network.

Like other forms of graphical models (e.g. Bayesian networks), Markov networks define a probability distribution over all possible states of the various propositions contained in the network. This joint distribution is given by (Richardson and Domingos, 2006)

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_k) \tag{2.2}$$

where Z is again just a normalization constant ($Z = \sum_y \prod_k \phi_k(y_k)$), the product is taken over all cliques in the graph, x_k is the state of the atoms in the kth clique when the graph as a whole is in state x, and

ϕ_k is a *potential function* defined on that clique. Note that the cliques of the graph correspond roughly to the ground formulas of the MLN: every ground formula has a corresponding clique which contains all of the ground atoms making up that formula, although not every clique has a unique associated ground formula. For the kth clique, we define the potential function ϕ_k to be 1 if no ground formulas are associated with the clique, and $\phi_k(x_k) = \prod_{f \in F} e^{w_f g_f(x_k)}$ otherwise, where F is the set of ground formulas which correspond to the kth clique, w_f is the weight given to the first-order formula of which the formula f is an instantiation, and $g_f(x)$ is 1 if formula f is true when the clique is in state x_k, and 0 otherwise.

From this representation it is easy to derive Equation 2.1 for the probability distribution specified by an MLN. Observe:

$$\begin{aligned} P(X = x) &= \frac{1}{Z} \prod_k \phi_k(x_k) \\ &= \frac{1}{Z} \prod_{f \in F} e^{w_f g_f(x_f)} \\ &= \frac{1}{Z} \prod_i (e^{w_i})^{n_i(x)} \end{aligned}$$

where i ranges over all of the first-order formulas of the MLN and $n_i(x)$ is the number of true groundings of the ith formula; this is simply collapsing together the terms for all of the ground formulas produced by the same first-order formula, since they have identical weights. We can then rewrite the distribution in log-linear form

$$\begin{aligned} P(X = x) &= \frac{1}{Z} \prod_i (e^{w_i})^{n_i(x)} \\ &= \frac{1}{Z} \exp\left(\log\left(\prod_i (e^{w_i})^{n_i(x)}\right)\right) \\ &= \frac{1}{Z} \exp\left(\sum_i n_i(x) \log\left(e^{w_i}\right)\right) \\ &= \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \end{aligned}$$

thus justifying Equation 2.1 (it should be easy to see that this Z is equal to the Z of Eqn. (2.1)). This demonstrates that the first-order MLN representation produces the same distribution as a ground Markov network built from the relevant ground atoms. Note that the ground Markov network is exponentially larger than the MLN representation which generated it, since the ground network must explicitly represent every ground atom.

2.2.2 Inference

The definition of MLNs gives us an expression for the probability of any particular state of the world. We can extend this to allow us to find the probability of arbitrary first-order formulas, or even the probability of one formula conditioned on another. Given an MLN L, a set of constants C, and any two first-order formulas F_1 and F_2, the conditional probability that F_1 holds given that F_2 holds is given by

$$P(F_1 \mid F_2) = \frac{P(F_1 \wedge F_2)}{P(F_2)} \tag{2.3}$$

$$= \frac{\sum_{x \in \{F_1\} \cap \{F_2\}} P(X = x)}{\sum_{x \in \{F_2\}} P(X = x)} \tag{2.4}$$

where $P(X = x)$ is defined as in Eqn. (2.1) and $\{F_1\}$,$\{F_2\}$ are interpreted as the sets of states in which those formulas hold. This is simply saying that the conditional probability $P(F_1|F_2, M_{L,C})$ is given by the probability of F_1 and F_2 both being satisfied, divided by the probability that F_2 is satisfied. The probabilities of these conditions can be computed by summing the probabilities of each of the states in which they hold, where those underlying probabilities are computed using the standard procedure given in Equation (2.1).

Unfortunately, while this method is theoretically valid, counting the number of satisfying assignments to first-order formulas is in general #P-complete, and therefore intractable to compute exactly in practice. Instead, a variety of more efficient techniques are used to perform approximate inference; see Richardson and Domingos (2006) and Mihalkova (2009) for details.

2.3 Learning in MLNs

The formulas and weights of an MLN can be specified entirely by hand, but in many cases we do not know the exact relationships which are most relevant to the structure of a domain, nor do we know the precise weight to give to each formula. The usefulness of Markov logic networks therefore rests in large part on our ability to learn these elements from data. That is, given information about the truth values of some set of ground atoms, we want to learn an MLN structure, which as discussed above will generate probability distributions over any set of ground atoms from the domain. For example, we may know the functions and interactions of a small set of proteins from a particular type of yeast[1], and want to learn from these a general pattern of relationships – i.e., an MLN consisting of formulas with weights —

[1]That is, we know the truth values of the ground atoms `HasFunction`(*protein1*, *functionX*), `HasFunction`(*protein2*, *functionX*), `Interaction`(*protein1*, *protein2*), etc. for some small set of proteins and their functions.

which will allow future prediction of the functions or interactions of other proteins. This task is broken into two subtasks which are generally approached separately: learning the weights to associate with each formula, and learning the formulas themselves (structure learning). This thesis focuses on the structure learning task (Lowd and Domingos (2007) discuss approaches to weight learning), and two algorithms for MLN structure learning are discussed below: one (MSL) which uses a local search procedure to find a knowledge base (i.e. a set of formulas, or in this case, clauses) which maximizes the likelihood of the observed data, and another (LHL) which directly constructs a candidate KB from the data.

2.3.1 MSL

Introduced by Kok and Domingos (2005), MSL (MLN Structure Learning)[1] is a local search algorithm, which builds a knowledge base one piece at a time, using hill-climbing to find a good solution. MSL works exclusively with MLNs in clausal form, and in the following description we will assume that all MLNs are in clausal form.

MSL attempts to find the KB which maximizes the likelihood of the training set. Because computing the true likelihood is #P-complete, and even finding good approximate solutions is computationally intensive (Kok and Domingos, 2005), MSL instead attempts to maximize a quantity based on the *pseudo-log-likelihood* (PLL) (Besag, 1975)

$$\log P^*_{KB,W}(X = x) = \sum_{l=1}^{n} \log P_{KB,W}(X_l = x_l | MB_x(X_l)) \tag{2.5}$$

which is the sum, over all ground atoms in the training data, of the log-likelihoods that each atom is in the state found in the data, given as evidence the atoms in its Markov blanket (the set of atoms with which it shares an edge in the ground Markov network, i.e. the atoms on which it is conditionally dependent). The subscript KB, W indicates that the likelihood depends on both the knowledge base and the choice of weights which define the MLN[2]. The PLL differs from a genuine likelihood because the likelihood of a state of the world cannot, in general, be factored into the likelihoods of its individual ground atoms. Since the PLL tends to give too much weight to predicates with greater arities (as each can be grounded

[1]This algorithm does not seem to have a consistent name; Kok and Domingos (2005) refer to it as "MLN(SLB)", Mihalkova (2009) calls it "KD", and Domingos and Lowd (2009) refer to it as "TDSL". The use of "MSL" in this thesis follows Davis and Domingos (2009).

[2]Note that since the likelihood depends on the weights as well as the formulas, we must also learn weights before evaluating a set of formulas. This adds computational difficulties, which are ignored in this overview but discussed by Kok and Domingos (2005).

in many different ways, and so produces disproportionately many ground atoms), Kok and Domingos (2005) introduced the *weighted pseudo-log-likelihood*

$$\log P^{\bullet}_{KB,W}(X = x) = \sum_{r \in R} c_r \sum_{g \in G_r} \log P_{KB,W}(X_g | MB(X_g)) \tag{2.6}$$

in which the single summation over all ground atoms is broken up into two parts: an outer sum over the set of predicates R, and an inner sum over the groundings G_r of each predicate. The contribution of the rth predicate is then scaled by a weight c_r, which MSL sets as $c_r = \frac{1}{|G_r|}$ in order to cancel out the effect of variable numbers of groundings and establish a uniform weight for each predicate.

The WPLL gives the probability that the data is in the observed state, given the structure of a particular MLN. From this, we can use Bayes' rule to infer the converse, i.e. the probability of any particular MLN structure given the observed data. In general this is given by

$$P(\text{KB} \mid \text{data}) = \frac{P(\text{data} \mid \text{KB})P(\text{KB})}{P(\text{data})}$$

and since the dataset does not change as we consider various different KBs to model it, the denominator $P(\text{data})$ may safely be ignored during the maximization. This means that the probability of an MLN structure varies according to the likelihood of the observed data under that structure, multiplied by the prior probability of the structure. To prevent overfitting and discourage the algorithm from making too many changes or adding too many new, specific clauses, MSL uses a structure prior given by $e^{-\alpha \sum_{i=1}^{C} d_i}$, where the sum is taken over all clauses in the KB, d_i is the number of predicates differing between the current version of a clause and its original version (if a clause is new, this is its length), and α is a normalization constant. Thus, the quantity which MSL maximizes is given by the penalized WPLL

$$P(\text{KB}|\text{data}) = P(\text{data}|\text{KB})P(\text{KB}) = P^{\bullet}_{KB,W}(X = x) \cdot e^{-\alpha \sum_{i=1}^{C} d_i}.$$

MSL can be used to revise an existing knowledge base, or to learn a new KB from scratch. In either case it starts with a KB consisting of an initial set of clauses (when learning from scratch, this is just the predicates of the domain, taken as unit clauses) which it grows by adding new clauses, one at a time. Each additional clause is chosen using a beam search process (described below) which attempts to find the single clause which, if added to the current set of clauses under consideration, would most improve the weighted pseudo-log-likelihood of that set. When no such clause can be found, then the process terminates, a final step prunes all clauses which can be removed without decreasing the WPLL, and the remaining clauses are returned.

The beam search to select a new clause begins by considering all legal additions or deletions of a single atom (possibly negated) from each of the current clauses under consideration. From the resulting modified clauses, it picks the top b clauses which would most improve the WPLL of the MLN if added to it, considers all legal mutations of those clauses, selects the top b, and this mutation/selection process is then repeated until none of the mutated clauses improves the WPLL any further. At that point, the best of the b clauses is added to the MLN, and the beam search process begins anew from the updated MLN. Further details of the MSL algorithm are given by Kok and Domingos (2005).

2.3.2 LHL

An alternate approach to MLN structure learning is given by Kok and Domingos (2009). Rather than explicitly searching through the space of possible knowledge bases, as MSL does, the LHL (Learning through Hypergraph Lifting) algorithm constructs a candidate KB directly from the data. The procedure views a database as a *hypergraph*, a generalization of a graph in which edges can connect arbitrary numbers of nodes. Every constant in the domain is a node in the hypergraph, and every true ground atom is a hyperedge, where the hyperedge links the nodes representing a set of constants if and only if those constants appear as arguments in the same ground atom.

The algorithm builds a knowledge base by finding paths in the hypergraph. A path in the hypergraph can be viewed as a conjunction of the ground atoms corresponding to the hyperedges of the path, and can be converted into a first-order clause by variabilizing the ground atoms (i.e. replacing each constant with a variable) and then applying De Morgan's laws to convert the conjunction into a disjunction. The resulting set of first-order clauses is then filtered using a greedy selection strategy which chooses the clause providing the largest increase in WPLL, and repeats until no clause improves the WPLL (this is the same greedy selection method used in the deep transfer method, described below).

The most sophisticated version of the algorithm first clusters the data into a lifted hypergraph, in which the nodes represent clusters of the domain constants, rather than the constants themselves. This clustering allows tractable pathfinding on large datasets. Kok and Domingos (2009) report that hypergraph lifting produces results generally superior to those of MSL and requires a shorter running time on large datasets. A basic version of the algorithm, which does not perform clustering, is known as LHL-FindPaths; this requires a significantly higher running time but produces results roughly equivalent to those from the full LHL algorithm.

2.3.3 Others

Domingos et al. (2006) note that the structure of an MLN can in principle be learned using any inductive logic programming technique. ILP techniques are those that learn clausal knowledge bases (that is, sets of logical clauses which express true facts about some data set) from relational databases. Initial efforts at learning MLNs used CLAUDIEN (Raedt and Dehaspe, 1997), an ILP learner. However, unmodified ILP techniques are not ideal for learning MLN structure, because they do not account for the ability of MLN formulas to express uncertainty.

Recent approaches have also adapted techniques from the literature on ILP and on graphical models to the more general framework on MLNs. The BUSL algorithm of Mihalkova and Mooney (2007) performs bottom-up structure learning by first using relational pathfinding (Richards and Mooney, 1992), the ILP technique which also forms the basis for LHL, to construct "nodes" representing first-order conjunctions of related atoms, then applying a Markov network structure learner to find dependency relationships between the nodes. The cliques in the resulting graph (which is effectively a Markov network) are then instantiated into clauses to construct the final knowledge base. Another method, ILS (Biba et al., 2008), uses an iterated local search strategy which searches through the space of clauses in a manner similar to MSL's beam search, but conducts the search from multiple randomly chosen starting points so as to better avoid falling into local optima.

2.4 Deep Transfer in Markov Logic Networks

Structure learning in relational domains is often a difficult problem. One potential approach to improving learning performance is to apply transfer learning (Torrey and Shavlik, 2009), that is, to use information from related domains in order to assist learning the structure of a particular target domain. In general, motivation for transfer stems from the observation that humans do not typically begin from scratch when learning to perform a new task; instead, we apply relevant knowledge from related experiences we have had in the past. Learning to understand the speech of a new friend is easy only because we already know how to understand the speech of thousands of other people. Since for many tasks there is limited task-specific training data available, it seems evident that achieving human-level learning performance on complex tasks will require us to take advantage of transfer. This is sometimes referred to as the problem of "lifelong learning" (Thrun and Mitchell, 1995).

Davis and Domingos (2009) present a method for transfer learning in MLNs; the method is known as deep transfer (or DTM, for Deep Transfer in Markov logic networks). The goal of deep transfer learning

`¬ Actor(x) ∨ ¬Director(x)`
`¬ Director(x) ∨ ¬WorkedFor(x,y)`
`¬ Credits(x,y) ∨ ¬WorkedFor(y,z) ∨ Credits(x,z)`

(a) First-order clauses

`¬r(x) ∨ ¬s(x)`
`¬r(x) ∨ ¬s(x,y)`
`¬r(x,y) ∨ ¬s(y,z) ∨ r(x,z)`

(b) Second-order clauses

`{r(x),s(x)}`
`{r(s),s(x,y)}`
`{r(x,y),s(y,z),r(x,z)}`

(c) Second-order cliques

Table 2.2: The clausal-form conversion of the MLN of Table 2.1, lifted first into second-order logic, then into second-order cliques, illustrating three levels of abstraction in DTM.

is to transform knowledge learned in some source domain into *domain-independent* abstract knowledge, which can be used to bias the process of structure learning in a destination domain and hopefully lead to improved performance. This domain-independent knowledge is represented in the form of second-order logic, which generalizes and groups first-order formulas together according to common structure.

The DTM transfer process begins with a clausal form MLN describing the source domain. Possible strategies for obtaining such an MLN are discussed below. The clauses of the source MLN are abstracted by replacing each predicate name with a predicate variable to produce a set of second-order clauses, as shown in Table 2.2. Each clause is considered independently, so a single second-order predicate variable can take the place of different first-order predicate names in different clauses, but within a single clause it must only take the place of a single predicate.

After arriving at a set of second-order clauses, the deep transfer process groups them into *cliques*, where a clique is simply a set containing the atoms of a formula, stripped of any logical structure (see Table 2.2)[1]. The correspondence between second-order cliques and clauses is analogous to that of the ground formulas of a Markov logic network to the cliques in the corresponding Markov network. DTM requires that no two cliques can be equivalent modulo variable renaming (e.g. `{r(z,y),r(x,y),s(z,x)}`

[1]The abstraction from formulas to cliques is a design decision intended to allow DTM to capture a very general sort of abstract structure. Chapter 6 describes a transfer algorithm which uses second-order formulas to transfer more specific structural properties.

is equivalent to {`r(x,y),r(z,y),s(x,z)`} under the renaming $z \leftrightarrow x$) and that within every clique a path of shared variables must connect each pair of atoms (e.g. {`r(x,y),s(y,y)`} is allowed, but {`r(x,y),s(z,z)`} is not, because in the latter case the two atoms share no variables).

For each clique of size k, DTM defines 2^k *features* or *states* of the clique, which are all possible assignments of truth values to the atoms comprising the clique. The features are represented as conjunctions which hold true only if their components take on the designated truth values. Like the set of cliques overall, the set of features within a clique is pruned for redundancy modulo variable renaming, resulting in fewer than 2^k effective features in most cases. For example, if {`r(x,y),r(y,x)`} is a clique, then its three features are `r(x,y)` ∧ `r(y,x)`, `r(x,y)` ∧ ¬`r(y,x)`, and ¬`r(x,y)`∧ ¬`r(y,x)`; the fourth possibility ¬`r(x,y)` ∧ `r(y,x)` is not explicitly included because it would be redundant. An additional restriction is that the variables of a feature are not allowed to unify, so e.g. the feature `r(x,y)` ∧ `r(y,x)` really represents `r(x,y)` ∧ `r(y,x)` ∧ $x \neq y$.

2.4.1 Clique Evaluation

All cliques are scored to determine to what extent they represent a meaningful second-order regularity in the source domain. To score a second-order clique, DTM takes the average score of its top m first-order instantiations (this is to favor cliques having multiple useful instantiations, which we expect to capture important second-order regularity). This requires calculating a score for each of the first-order cliques which can result from instantiating the predicate variables of the second-order clique; the score of a first-order clique indicates intuitively whether the clique has any explanatory value as a structural regularity, above and beyond that provided by its various subcliques. To measure this, we divide the clique into every possible pair of subcliques (e.g. for a size-2 clique, the subcliques would just be the two individual nodes) and, for each division, compute the Kullback-Leibler divergence

$$D(p\|q) = \sum_x p(X=x) \log \frac{p(X=x)}{q(X=x)}$$

where X is a random variable representing a state of the clique, the sum is taken over all possible states of the clique, p is the the probability distribution over states assigned to the entire first-order clique, and q is the probability distribution that the clique would have if the two subcliques were independent (i.e. the product of the distributions of the two subcliques). This value of this divergence is the additional number of bits which would be required to encode a state of the clique using an encoding based on the distribution q instead of the distribution p, i.e. the information gain $\sum_x -p(x)\log q(x) - \sum_x -p(x)\log p(x)$ of considering the clique as a structural whole instead of as a conglomeration of two independent subcliques. The score of the first-order clique is the *minimum* of the K-L divergences of its various divisions, because if there is even one way of splitting a clique that does not change the distribution of the states of that clique, then that shows us that the dependence assumptions of the clique are not valid (i.e. we might as well just abandon the clique and work with its subcliques).

The probability distributions used in the above scoring process are estimated from the available data in the source domain. For example, suppose we are working in a domain concerned with people's smoking habits, and we want to score the first-order clique

$$\{\texttt{Friends(a,b), Smokes(a), Smokes(b)}\}.$$

This requires us to estimate $p(x)$ for each of the eight possible states of the clique (i.e. `a` smokes, `b` smokes, and `a,b` are friends; `a` doesn't smoke, `b` smokes, and `a,b` are friends; and so on). Our estimate of the probability of any particular state is simply the number of ways in which constants from the domain can be assigned to `a` and `b` to produce that state, normalized by the total number of possible assignments to `a` and `b`. If there are 100 people in our database and only 10 pairs of friends who each smoke, then the probability of the state `Friends(a,b)` $\wedge$ `Smokes(a)` $\wedge$ `Smokes(b)` is estimated to be $\frac{10}{4950}$ — this is just the number of pairs of friends who smoke, divided by the total number of pairs of people.

To estimate $q(x)$ for any particular decomposition of the clique, say $\{$`Smokes(a), Smokes(b)`$\}$,$\{$`Friends(a,b)`$\}$, DTM follows the same procedure to estimate the probabilities of the corresponding substates of x for both halves of the decomposition. Then the two probabilities obtained are simply multiplied together to produce an estimate of the joint probability under the assumption of independence, which is exactly the definition of $q(x)$.

2.4.2 Performing Transfer

After the scoring process is complete, DTM selects the top k second-order cliques which have at least one true grounding in the target domain, and uses them to bias structure learning in the target domain. Davis and Domingos (2009) give three approaches for doing this, two of which are considered here (their third approach, seeding the beam, is more complex and was not found to provide significant benefits over the methods presented below):

- **Greedy Selection (without refinement):** This method first generates all clauses corresponding to legal instantiations of the top k second-order cliques with legal groundings in the target domain.

It instantiates a clique into a set of clauses by simply converting every feature of the clique into a disjunction, and then for each disjunction considering all ways in which the second-order predicate variables can be replaced with first-order predicate names, while maintaining type and arity constraints. For example, the second-order clique {`r(x,y)`,`r(y,x)`} has three features `r(x,y)` ∧ `r(y,x)`, `r(x,y)`∧ ¬`r(y,x)`, and ¬`r(x,y)`∧ ¬`r(y,x)`. In the smoking domain mentioned earlier, with `Friends` and `Smokes` as the only first-order predicates, this clique would be instantiated as the three clauses

```
¬Friends(x,y) ∨ ¬Friends(y,x)
Friends(x,y) ∨ ¬Friends(y,x)
Friends(x,y) ∨ Friends(y,x)
```

After generating all clauses which are legal instantiations of the top cliques, the final MLN is built iteratively by greedily selecting the clause which most improves WPLL, adding it to the MLN, and repeating the process until no remaining clauses would improve WPLL by their addition.

- **Greedy Selection with Refinement:** This builds off of the previous approach by adding a refinement step. MSL (as described above) is used to refine a preliminary KB produced the greedy selection method.

2.4.3 Sources of Transfer

The deep transfer algorithm can perform transfer from any source-domain knowledge base, whether hand-coded or learned through automated structure learning. Davis and Domingos (2009) evaluate two approaches for generating such a KB: beam search and exhaustive search (they did not consider transfer from a hand-coded KB, and neither does this thesis, although such a process is certainly conceivable). The beam search-based method runs MSL several times to construct a set of models which each predict various subsets of the full set of predicates in the domain, and then considers together the entire set of clauses contained in the individual models. Exhaustive search, on the other hand, simply generates all valid first-order clauses in the domain, up to a maximum length and maximum number of variables – it does not consider the data available in the source domain. Although exhaustive search is only tractable when limited to relatively short clauses, Davis and Domingos (2009) found that the results produced were nonetheless as good as or better than those obtained through beam search. One explanation for this might be that transfer works best with short clauses in any case (since longer formulas are likely to be more arbitrary and domain-specific), and there is no penalty for including clauses which say little about the domain, since DTM scores all cliques and keeps only the top scoring cliques. DTM does not consider the weights of the formulas used as sources for transfer.

Unless otherwise specified, in this thesis I will generally use DTM to refer to the form of the algorithm in which exhaustive search is used to generate cliques for transfer.

2.4.4 Other Transfer Methods for MLNs

The `TAMAR` algorithm (Transfer via Automatic Mapping and Revision) proposed by Mihalkova et al. (2007) performs transfer using a direct mapping between the predicates of different domains, rather than by creating abstract domain-independent knowledge as in DTM. For every clause in the source domain, TAMAR conjectures a mapping of each predicate to a predicate in the target domain (the mapping is found by a simple exhaustive search over all possible mappings for the predicates within the clause, choosing the mapping which gives the highest WPLL for the resulting target-domain clause), and then revising the resulting MLN. This approach is based on analogical reasoning, in which the mappings between predicates represent analogies between source-domain and target-domain concepts (e.g. `Professor` to `Director` and `Student` to `Actor` might be a possible mapping). Davis and Domingos (2009) attempted to compare TAMAR to DTM and found that TAMAR gave significantly inferior results in one case, and failed to complete in another case because the exhaustive search over all possible mappings becomes intractable when transferring to a domain with large number of predicates.

A related approach which follows a similar analogical style is given by Mihalkova and Mooney (2009). The SR2LR algorithm is based on the same idea of choosing a cross-domain mapping of predicates, but with a focus on optimizing performance when there is very little data available in the target domain. This is done by splitting the source-domain clauses into a set of “short-range” clauses, whose mappings can be directly evaluated using the available data in the target domain, and “long-range” clauses, for which the available data is insufficient to evaluate potential mappings. Results from the short-range clauses are used to select mappings for the long-range clauses, and together these mappings applied to the source-domain MLN specify the final target MLN; no revision is performed.

3

Domains and Methods

This chapter provides common background for the results of Chapters 4, 5, and 6. In those chapters, I evaluate several methods for structural transfer and other forms of structure learning in Markov logic networks. While the the details of each experiment will be discussed as they arise, they do share many common features. In particular, all experiments used data from four real-world domains: the movie business (IMDB), an academic department (UW-CSE), web pages of computer scientists (WebKB), and the proteins of the yeast *Saccharomyces cerevisiae* (Yeast Protein). The first section of this chapter describes the methods common to all experiments in this thesis; the second section describes in detail the four domains used for evaluation.

3.1 Methods

When evaluating a structure learning method, each dataset was divided into four independent folds. Evaluation was through leave-one-out cross-validation, in which the system was trained on every three-fold subset of the four total folds (in this context, training consists of learning an MLN structure, i.e. a set of first-order formulas) and tested on the fourth. The results given are averages over results from the four folds of each domain.

All experiments used the implementation of MSL from the publicly available Alchemy package (Kok et al., 2009) along with an Alchemy-based implementation of DTM provided by Jesse Davis. Weight learning was performed using the L-BFGS-based algorithm described by Richardson and Domingos (2006), which optimizes the pseudo-likelihood of the data. Source clauses for DTM were generated by exhaustive search over all clauses containing at most three literals and three object variables. MSL structure

	Types	Predicates	Constants	True Atoms	Total Atoms
IMDB	5	5	251	1039	55722
UW-CSE	8	10	442	1407	174785
WebKB	3	3	4953	283489	20663916
Yeast Protein	7	7	2470	15015	4533900

Table 3.1: Datasets used.

refinement was time-limited to 20 hours for each trial. Following Davis and Domingos (2009), I allowed MSL to learn clauses containing constants; out of all the object types represented in the various datasets, I permitted only role (IMDB), rank (UW-CSE), function (Yeast), and page class (WebKB) to appear as constants in learned clauses.

Following Kok and Domingos (2005) and others, I evaluated each set of learned formulas using the test set conditional log-likelihood (CLL) and the area under the precision-recall curve (AUC). The CLL has the advantage of directly measuring the quality of the probability estimates produced, while the AUC is useful because it is insensitive to the large number of true negatives in the data. The CLL is calculated by averaging, over all ground atoms, the predicted log-likelihood that each ground atom takes on its true value, calculated using the learned MLN and given as evidence the truth values of all other ground atoms in the test set. Individual points on the precision-recall curve are found by varying the CLL threshold above which a ground atom is predicted to be true, and calculating the precision and recall for each threshold setting; the AUC is the area under the resulting curve.

3.2 Domains and Datasets

Details on the four datasets used in my experiments are provided in Table 3.1. The WebKB and Yeast Protein datasets were provided by Jesse Davis and are used to allow comparison with the results of Davis and Domingos (2009). The IMDB and UW-CSE datasets are publicly available from `alchemy.cs.washington.edu` (as is a version of WebKB).

WebKB. This dataset consists of labeled web pages from the computer science departments of four universities, with predicates indicating the words occurring on each page, the class label of each page (faculty, student, course, etc.), and the links between pages. The data from each university is treated as a separate fold. I attempt to predict the truth values of all groundings of the `Linked` and `PageClass` predicates. These correspond to the “link prediction” and “collective classification” tasks, respectively, in which the available information about a page's situation within the broader network is used in order to predict which pages it links to and what its class label should be. The data is originally sourced from Craven and Slattery (2001), and the version used both in this paper and in Davis and Domingos (2009) is equivalent to the version publicly available at `alchemy.cs.washington.edu`, with the following modifications: the seven single-arity predicates `FacultyPage`, `CoursePage`, etc. indicating the class label of a page are collapsed into a single predicate `PageClass` of arity two, and only the `Has`, `Linked`, and `PageClass` predicates are considered.

For tractability on the WebKB data, I followed Davis and Domingos (2009) in using information gain on the training set to pick the fifty words most predictive of page class; these were used to train and evaluate the learned MLN.

Yeast Protein. This dataset contains information on protein location, function, phenotype, class, and enzymes within the yeast *Saccharomyces cerevisiae*, as well as protein interactions and protein complex data, all from the MIPS Comprehensive Yeast Genome Database as of February 2005 (Mewes et al., 2002). I used the version of the data from Davis and Domingos (2009), which is split into four disjoint subsamples which are used as folds, and attempted to predict the function of each protein as well as its interactions with other proteins, represented by the `Function` and `Interaction` predicates, respectively.

IMDB. This dataset describes a movie domain, consisting of movies, actors, directors, etc. and with predicates indicating their relationships. The data was collected from `imdb.com` by Mihalkova and Mooney (2007). The data is split into five disjoint folds, but I used only the first four folds in order to maintain consistency with my use of WebKB and Yeast. I attempted to predict the `WorkedUnder` and `WorkedInGenre` predicates. Following Kok and Domingos (2009) I omitted four equality predicates which are superseded by the equality operator available in Alchemy. In addition, for consistency with WebKB, I collapsed the single-arity `Actor` and `Director` predicates into a single predicate `HasRole` with arity two (similar to `PageClass` in WebKB).

UW-CSE. This dataset, from Richardson and Domingos (2006), describes anonymized relationships between students, faculty, and courses in the University of Washington Computer Science and Engineering Department. The data is split into five folds representing the subdisciplines of AI, graphics, programming languages, systems, and theory; again for consistency I used only four folds, omitting the systems data (chosen randomly). Following Richardson and Domingos (2006) I attempted to predict the `AdvisedBy` predicate. As with IMDB, I omitted nine redundant equality predicates, and I collapsed the single-arity `Student` and `Professor` into a single `HasRank`. I also simplified the `TaughtBy` and `TA` predicates of UW-CSE to ignore the particular quarter in which a course was taught, reducing the arity of each of those

IMDB	**UW-CSE**	**WebKB**	**Yeast**
Gender(person,male_or_female)	AdvisedBy(person person)	Has(page,word)	Complex(protein,complex_id)
HasRole(person,+dir_or_actor)	CourseLevel(course,level)	Linked(page,page)	Enzyme(protein,enzyme_id)
WorkedInGenre(person,genre)	HasPosition(person,position)	PageClass(page,+pclass)	Function(protein,+func_id)
WorkedOnMovie(movie,person)	HasRank(person,+prof_or_student)		Interaction(protein,protein)
WorkedUnder(person,person)	InPhase(person,phase)		Location(protein,location_id)
	Publication(title,person)		Phenotype(protein,phenotype_id)
	Ta(course,person)		ProteinClass(protein,pc_id)
	TaughtBy(course,person)		
	TempAdvisedBy(person,person)		
	YearsInProgram(person,integer)		

Table 3.2: Predicates and object types for each dataset. Plus signs indicate terms which are allowed to appear as constants in learned clauses.

predicates from three to two. This change was motivated by the fact that DTM can only transfer between predicates having the same arity; because there are no arity-three predicates in our other datasets these predicates would otherwise have been completely ignored by the transfer process.

Table 3.2 lists all of the predicates considered for each domain.

4

Preliminary Questions

This chapter asks and answers two basic questions about the DTM framework for cross-domain transfer. Although the deep transfer algorithm is designed to perform transfer from a single source domain to a target domain, Davis and Domingos (2009) do suggest the extension to multiple source domains as possible future work. The first section of this chapter investigates this possibility, gives two different mechanisms for doing so—both of which naturally extend the standard DTM algorithm for single-source transfer—and evaluates their performance. I then turn to the question of whether the structures discovered by DTM are so general as to not even be domain-specific in any meaningful sense; this is inspired by Davis and Domingos (2009) noting that certain patterns such as symmetry, homophily, and transitivity, occurred in every domain they evaluated. The second section of this chapter addresses this question by attempting to construct a single set of cliques which is effective as a universal transfer source, i.e. a set of cliques which captures widely useful structural properties which are common to many different domains, and which we would expect to perform well as a transfer source regardless of the target. I construct two candidates for such a set and evaluate their performance as sources for transfer to all four of the domains considered in this thesis.

4.1 Transfer from Multiple Sources

This section describes two relatively straightforward methods for transfer from two source domains simultaneously; each can be easily extended to transfer from three or more sources. Each method begins by identifying the top-ranked cliques in each source domain, using exhaustive search and clique-scoring exactly as if executing DTM to perform single-source transfer in the usual way. Recall that the score of a clique is the average score of its best k first-order instantiations (following Davis and Domingos (2009) I set $k = 3$), where the score of a first-order instantiation is given by the worst-case K-L divergence between the probability distribution of the clique and the distribution resulting from assuming that some of its subcliques are independent of each other. For each source domain, this scoring process produces a list of cliques with corresponding scores. Two different methods are considered for merging these lists into a single, unified list of cliques:

- **Top Cliques:** This method combines the highest scoring cliques from both domains, by taking the union of the top m cliques from each domain's list. If a clique occurs on both lists with different scores, in the merged list it is assigned the higher of the two scores.

- **Common Cliques** This method finds high scoring cliques which are shared across both domains. We consider the top m cliques of each domain (where m is larger than n, the number of cliques required for our merged list) and create a merged list by taking the intersection, so that a clique appears in the list only if it is within the top m cliques of *both* domains. Again, if a clique appears in both lists with different scores, then in the merged list it is assigned the higher of the two scores.

A related approach, not evaluated here, would be a modification of the Top Cliques method which assigns each clique the minimum of its two scores, rather than the maximum (if a clique has no true instantiations in a domain, it is given a score of zero in that domain). This would have the effect of preferring cliques which score well in both domains, but in a more nuanced manner than that of the Common Cliques method (if desired, the preference could be relaxed somewhat by using the mean instead of the minimum).

Regardless of the method used, once a merged list of cliques is generated, the top n cliques are considered, ranked using the scores of the merged list. The transfer process then proceeds exactly as in the standard DTM algorithm, instantiating these cliques in the destination domain and using greedy selection (with or without refinement) to generate a final set of first-order clauses.

4.1.1 Experiments and Results

Table 4.1 gives the areas under the curve for greedy transfer with refinement of the top and common cliques of every two-source transfer scenario. All lists of combined cliques were created by considering the top $m = 25$ cliques of the individual domains, and in each case the top $n = 10$ combined cliques were used for transfer. Results are omitted whenever a test predicate is contained in one of the source datasets; because

Domain	Predicate	I+U	I+W	I+Y	U+W	U+Y	W+Y	Best Individual	MSL
IMDB	WorkedInGenre				0.61	0.61	0.61	0.61	0.32
IMDB	WorkedUnder				0.23	0.23	0.21	0.23	0.03
UW-CSE	AdvisedBy		0.08	0.08			0.08	0.08	0.04
WebKB	Linked	0.02		0.01		0.04		0.09	0.004
WebKB	PageClass	0.86		0.86		0.86		0.86	0.87
Yeast	Function	0.34	0.34		0.34			0.34	0.27
Yeast	Interaction	0.04	0.04		0.04			0.10	0.04

(a) Top Cliques

Domain	Predicate	I+U	I+W	I+Y	U+W	U+Y	W+Y	Best Individual	MSL
IMDB	WorkedInGenre				0.61	0.56	0.61	0.61	0.32
IMDB	WorkedUnder				0.30	0.31	0.21	0.23	0.03
UW-CSE	AdvisedBy		0.10	0.10			0.08	0.08	0.04
WebKB	Linked	0.01		0.04		0.01		0.09	0.004
WebKB	PageClass	0.86		0.86		0.86		0.86	0.87
Yeast	Function	0.34	0.34		0.34			0.34	0.27
Yeast	Interaction	0.04	0.04		0.04			0.10	0.04

(b) Common Cliques

Table 4.1: AUC of the top and common cliques for each two-source transfer scenario, using greedy transfer with refinement. Domains are labeled in the column headings by the first letter of their full names.

the source cliques are discovered using all four folds of available data from each domain, to report these results would constitute testing on the training set. For each predicate, the "Best Individual" column gives the best-case performance of deep transfer from any single domain, using greedy selection with refinement and transferring $n = 10$ cliques from any of the three domains not containing the predicate. Results for MSL are also provided for purposes of comparison.

In no case did transfer of the top ten cliques taken from two different domains produce better results than did transfer from the better of the two source domains considered individually (see Chapter 5 for the results of transfer from each individual source domain). Transfer from the top ten common cliques did improve on the results of both component domains in four specific cases — namely, predicting `AdvisedBy` using IMDB+WebKB and IMDB+Yeast, and predicting WorkedUnder using UWCSE+WebKB and UWCSE+Yeast — although none of the improvements were significant (paired one-tail t-test, $p > 0.1$). Interestingly, the top ten cliques of the WebKB and Yeast domains are exactly identical (as shown in Table 5.3, so the transfer from WebKB+Yeast is equivalent to transfer from either of those two individual domains. However, because WebKB and Yeast assign different scores to the same cliques, they sometimes give different results in these experiments when combined with other domains.

4.1.2 Discussion

The results show no significant advantage for either method of transfer from multiple sources, compared to transfer from a single source (although there may be some small advantage for the common-cliques method which does not rise to the level of statistical significance). This does not preclude the possibility that there might be some case in which it would be advantageous to perform transfer from multiple source domains, but it does show that, in many typical transfer situations, there is no advantage to doing so using either of these methods. The methods described in this section extend the single-source DTM algorithm in a simple, natural way, but they do not exclude the possibility of a more sophisticated algorithm which might be more generally effective.

4.2 Universal Transfer

One intriguing property of deep transfer is that many of the same cliques tend to turn up regardless of the particular source domain used. This was noted by Davis and Domingos (2009), who interpreted some of these common cliques as representing certain intuitive structural properties such as homophily (which states that objects having similar properties tend to also have some direct relationship), transitivity, and symmetry. On the one hand, this could help to explain why it appears to often not be helpful to merge two source domains (i.e. because if two domains have similar cliques, then the merged version will not be too different from either domain individually), but on a broader level, it raises the question of whether the performance increases observed through deep transfer might simply be due to a small set of common structural properties present in almost every real-world domain, rather than any specific useful knowledge from the source domain. If the former, there ought to be some set of highly-common cliques that could be used as a "universal" transfer source, to improve structure learning performance without requiring the choice of a specific source domain. Note that although the no free lunch theorems (Wolpert, 1996) forbid the existence of a truly universal transfer source due to the philosophical problem of induction, this does not preclude the possibility that there may still exist some set of cliques which is effective as a fixed transfer source for many real-world domains.

To explore this possibility, I generated two sets of cliques using data from all four available datasets; these sets were intended to serve as candidates for a universal transfer source. The first set of cliques, referred to as "U4", consists of all cliques which were ranked among the top 25 cliques in each of the four datasets considered. There were ten such cliques, listed in Table 4.2(a). The second set of cliques, called "U3", consists of all cliques which were ranked among the top 15 in at least three of the four datasets;

Rank	Clique	Interpretation
1	r(x,y),r(x,z)	co-occurring feature pairs
2	r(x,y),r(z,y)	common feature values
3	r(x,y),r(z,y),s(x,z)	homophily
4	r(x,y),r(z,y),s(x,x)	
5	r(x,y),r(x,z),s(x,x)	
6	r(x,y),r(y,z),r(z,x)	transitivity
7	r(x,y),s(x,z)	
8	r(x,x),r(y,x),r(z,x)	
9	r(x,x),r(y,x)	
10	r(x,x),r(y,x),s(x,z)	

(a) U4

Rank	Clique	Interpretation
1	r(x,y),r(x,z)	co-occurring feature pairs
2	r(x,y),r(z,y)	common feature values
3	r(x,y),r(z,y),s(x,z)	homophily
4	r(x,y),r(z,y),s(x,x)	
5	r(x,y),r(x,z),s(x,x)	
6	r(x,y),r(y,z),r(z,x)	transitivity
7	r(x,y),s(x,z)	
8	r(x,y),r(y,x),r(z,x)	
9	r(x,y),r(y,x)	symmetry
10	r(x,x),r(y,x),r(z,x)	

(b) U3

Table 4.2: Candidate cliques for a universal transfer source.

thus allowing the additional consideration of cliques which were generally but not universally useful. This set, listed in Table 4.2(b), also included ten cliques. Note that U3 includes cliques representing every one of the intuitive properties mentioned by Davis and Domingos (2009), and U4 includes cliques representing all but symmetry. Although in principle any set of cliques can be treated as a possible source for universal transfer, the fact that U4 and U3 were constructed from the shared cliques of very different domains, in addition to their inclusion of these common intuitive properties, makes U4 and U3 particularly plausible candidates for such a source.

Domain	**Predicate**	**U3**	**U4**	**Best Individual**	**MSL**
IMDB	**WorkedInGenre**	0.56	0.56	0.61	0.32
IMDB	**WorkedUnder**	0.31	0.30	0.23	0.03
UW-CSE	**AdvisedBy**	0.10	0.08	0.08	0.04
WebKB	**Linked**	0.06	0.01	0.09	0.004
WebKB	**PageClass**	0.86	0.86	0.86	0.87
Yeast	**Function**	0.33	0.34	0.34	0.27
Yeast	**Interaction**	0.10	0.04	0.10	0.04

(a) AUC (higher is better)

Domain	**Predicate**	**U3**	**U4**	**Best Individual**	**MSL**
IMDB	**WorkedInGenre**	-0.16	-0.16	-0.13	-0.30
IMDB	**WorkedUnder**	-0.16	-0.16	-0.17	-0.23
UW-CSE	**AdvisedBy**	-0.03	-0.03	-0.03	-0.04
WebKB	**Linked**	-0.02	-0.02	-0.02	-0.02
WebKB	**PageClass**	-0.07	-0.07	-0.07	-0.07
Yeast	**Function**	-0.18	-0.18	-0.18	-0.19
Yeast	**Interaction**	-0.03	-0.04	-0.03	-0.04

(b) CLL (closer to zero is better)

Table 4.3: Transfer from common cliques.

4.2.1 Experiments and Results

Results of greedy transfer with refinement from U4 and U3 are presented in Table 4.3. For each predicate, the "Best Individual" column gives the best-case performance of deep transfer from a single source, using greedy selection with refinement, transferring $n = 10$ cliques from any of the remaining three domains not containing the predicate. Results for MSL are also provided for purposes of comparison.

Transfer from both U3 and U4 matches or exceeds the performance of MSL in every case, which indicates that these sets of cliques are genuinely aiding in the structure learning process for all four domains tested. In three cases the advantage over MSL in AUC was significant: when predicting `WorkedUnder` using either set of cliques, and U4 predicting `Function` (paired two-tail t-test, $p < 0.05$; several other transfer cases were almost significant but suffered from high variance among the four folds). U3 performed better overall than U4, presumably because its more relaxed method of construction allows it to consider cliques which are generally useful even if they are not ubiquitous. The performance of U3 matches or exceeds that of the best case of single-source DTM for four of the seven predicates, while

DTM maintains a slight advantage in the other three cases. For none of the seven predicates was the difference in AUC between U3 and the best single-domain transfer source statistically significant (paired two-tail t-test, $p > 0.1$).

4.2.2 Discussion

Both sets of cliques performed as well or better than MSL on all predicates in the domains I considered, and the improvement was significant in several cases. This shows that relative to structure learning in the target domain, transfer from U3 or U4 never hurts and often helps. Since both U3 and U4 contain cliques representing structural properties which might be expected to be commonly applicable, this empirical evidence adds strength to the argument that they should be considered as generally useful sources for transfer.

Compared to the best cases of single-domain transfer for each predicate, neither U3 nor U4 prevailed in every case. However, both sets of cliques equalled or exceeded the performance of the best single-domain source for four of the seven predicates tested. This shows that in many cases, these sets of cliques not only explain the success of transfer through DTM, but actually improve on it — it is sometimes better to simply use one of these fixed sets containing commonly useful cliques, than it would be to search laboriously for the most useful transfer source!

There is one major caveat to these conclusions, however. Because since U3 and U4 were each generated by combining the common cliques of the four domains, they suffer from the same malady as the results which were excluded from Table 4.1: the process which constructed them depended on the data from the same four domains which they were tested on. Thus, it is premature to draw from these results the conclusion that either U3 or U4 would be truly useful as a source for transfer to a wider range of domains. Convincing confirmation of this argument will require that they be tested for effectiveness in a wide variety of domains, beyond the four considered here.

5

Self-Transfer: DTM as a Structure Learner

The results of the previous chapter show that, in many cases, much of the effectiveness of deep transfer may be ascribed to the presence of a few cliques which represent common structural regularities. However, despite their effectiveness there is no reason to think these common cliques are, by themselves, the optimal transfer source for any particular target domain. This chapter explores the problem of choosing the best single-domain source for transfer, and arrives at a method which in almost all cases gives results equalling or exceeding those of the previous chapter.

The problem of choosing the most appropriate source domain for transfer is a common issue in transfer learning (Torrey and Shavlik, 2009). One can argue that if a source task is to have any chance of improving learning performance, it must bear some relation to the target task; it cannot be entirely irrelevant. At the same time, the knowledge gained through learning in the source task will be more effective if it is not entirely redundant with the target task, so there ought to be some distinction between the source and target tasks. Together, these conditions suggest that the best performance gains might be found by considering source tasks which are closely related, but not identical, to the target task. In fact, it has been shown within the context of propositional domains that considering a set of closely related tasks can allow the learning of a more general model (i.e. one less prone to overfitting) than could be learned by considering any task alone (Caruana, 1997).

Is there any principled way to judge the closeness of the relationship between two learning tasks? In many cases this is left to intuition, with the general guideline that tasks from the same general domain tend to be related; for example, the problem of reading John's handwriting is most likely related to the problems of reading Sally, Mark, and Ruth's handwriting. Unfortunately, this heuristic is often not applicable when considering potential transfer sources for DTM, because the central premise of deep transfer is that useful information can be transferred *even across across domains which appear to be wildly different*, as long as they share some common second-order structure. Since the brute force approach of empirically evaluating the performance of all possible transfer cases is clearly undesirable, it is reasonable to seek out a more practically tractable method for deciding when two domains are similar enough that transfer from one to the other might be productive.

One such approach might involve pre-computing the highest scoring second-order cliques of every available dataset. Learning the structure of some domain X, then, would dictate searching for the domain whose second-order cliques are most similar to those of domain X and simply using that domain as the source for transfer.

The "catch" to this seemingly straightforward approach is that the domain with cliques most similar to the cliques of domain X is, of course, domain X itself. If it were to be true in general that sharing similar second-order cliques indicates a potential for useful transfer, then it would follow immediately that the most useful source for transfer is generally the target domain itself. This might seem on the surface to be absurd, since one would expect no new information to be gained by performing transfer from a domain to itself. However, this chapter argues that not only is this counterintuitive proposition supported by empirical evidence, but that it can actually provide a compelling explanation for why DTM works as well as it does. In particular, I argue that the exhaustive search and clique-scoring mechanisms of DTM function together as a form of MLN structure learning, and that the cross-domain transfer for which DTM was designed is most properly interpreted as a suboptimal variant of an underlying single-domain structure learning algorithm.

The first section of this chapter serves to motivate the view of exhaustive search and clique scoring as a form of structure learning. The second section builds on this to propose two new related algorithms, CSGL and self-transfer, which perform MLN structure learning within a single domain. The next section evaluates these algorithms empirically and compares them to both deep transfer and MSL, and the final section discusses some interesting implications of these results.

5.1 DTM for Structure Learning

It was noted in Chapter 2 that although DTM can extract cliques from any set of formulas in the source domain, Davis and Domingos (2009) found that the best results came from considering, not the results of beam search or other sophisticated structure learning techniques, but a simple exhaustive listing of all possible clauses in the domain within a maximum length and number of terms. They argue that since the clique scoring process already suggests the most useful cliques for transfer, there is no additional benefit in trying to learn a theory in the source domain which would restrict the cliques that are considered for transfer.

An interesting effect of the use of exhaustive search is that unlike many transfer methods, DTM performs a qualitatively different sort of learning in the source domain than in the target domain. DTM's only interaction with the data of the source domain is through the clique-scoring process, while in the destination domain it uses a greedy selection process followed by beam search to refine the clauses produced through transfer. Although the clique scoring process produces a list of second-order cliques, not a list of first-order formulas as is standard for a structure learning algorithm, the cliques which it produces do reflect in some sense the structural properties of the source domain. This indicates that clique scoring is, in itself, performing a sort of structure learning, and that the success of the DTM algorithm rests in part on the effectiveness of this process.

5.2 The Self-Transfer/CSGL Algorithm

This section proposes a simple modification to DTM, which I refer to as "self-transfer". As the name implies, the self-transfer method effectively applies DTM to a single domain as both source and target. First, the algorithm constructs an exhaustive list of all possible first-order clauses in the domain, up to some maximum length and number of object variables. These clauses are then abstracted into second-order cliques and scored via the standard DTM clique-scoring process. The k top-scoring cliques are then instantiated back into first-order clauses of the domain (i.e. the same domain they were abstracted from). From this set of clauses, either the "greedy selection" or "greedy selection with refinement" methods can be used to derive a final set of clauses indicating the structure of the domain. That is, clauses are greedily selected from the instantiated first-order clauses until no additional clause improves the WPLL, and the resulting MLN is optionally refined using MSL.

Since the version of the algorithm utilizing only greedy selection (no refinement) does not rely on MSL or any other existing structure learning algorithms, and yet has the effect of inducing an MLN structure from the data of the domain, it is reasonable to consider it as a standalone structure learning algorithm. I will refer to this form of the algorithm as CSGL, or learning through $\underline{\text{c}}$lique $\underline{\text{s}}$coring and $\underline{\text{g}}$reedy se$\underline{\text{l}}$ection. When a refinement step is used, I will refer to the algorithm as CSGL-R.

One might expect that using the same domain as both source and target would forfeit some of the expected benefits of transfer, for example the improved generalization performance which often results from considering a broader class of related tasks (Caruana, 1997). Surprisingly, this does not appear to be the case: in our experiments the results of self-transfer were generally comparable to the best results from DTM. While there may exist circumstances in which cross-domain transfer through DTM is more effective (e.g. if the target domain has very little data available, or if we wish to learn structure for many domains using the same source to avoid repeating the clique-scoring process for each domain), we demonstrate that in many cases the benefits of transfer observed using DTM are achievable through simple self-transfer.

This observation strongly suggests that much of the fundamental utility of DTM comes, not from the ability to transfer knowledge between domains, but from the introduction of a new learning process, namely the process of clique scoring in conjunction with first-order instantiation and greedy selection (under this interpretation, the common structures explored in Chapter 4 are useful insofar as they tend to be found in many real-world domains, but the self-transfer approach would be effective even in domains which do not share those common structures). In fact, we find that clique scoring with greedy selection (CSGL) alone yields results comparable to MSL.

5.3 Experiments and Results

Both CSGL and CSGL-R were evaluated on the four available datasets and compared to other cases of deep transfer. In all cases of self-transfer (including both CSGL and CSGL-R), cliques were gathered and scored using only the training set, not the full dataset. This put these methods at a modest disadvantage in terms of the quantity of data available during the learning process, since in all other transfer cases we gathered and scored cliques using all data available from the source domain. That said, within each domain the cliques which were identified in the three folds of each training set did not differ significantly from those obtained using the full four folds, with only minor changes in ordering in most cases, so this is unlikely to have seriously affected the results.

Table 5.1 gives the AUC and CLL for all transfer scenarios using refinement, including self-transfer results (which are underlined) as well as MSL, which acts as a baseline. Each figure represents an average over the four different train/test trials. Note that the results for transfer from WebKB and Yeast are identical. This is because the ten highest-scoring cliques are the same in both domains (see Figure 5.3 for

	IMDB	UW-CSE	WebKB	Yeast	MSL
WorkedInGenre	0.63	0.61	0.61	0.61	0.32
WorkedUnder	0.77	0.23	0.21	0.21	0.03
AdvisedBy	0.08	0.08	0.08	0.08	0.04
Linked	0.01	0.01	0.09	0.09	0.004
PageClass	0.86	0.86	0.68	0.68	0.87
Function	0.34	0.34	0.33	0.33	0.27
Interaction	0.04	0.04	0.10	0.10	0.04

(a) AUC

	IMDB	UW-CSE	WebKB	Yeast	MSL
WorkedInGenre	-0.20	-0.13	-0.37	-0.37	-0.30
WorkedUnder	-0.09	-0.17	-0.21	-0.21	-0.23
AdvisedBy	-0.03	-0.03	-0.03	-0.03	-0.04
Linked	-0.02	-0.02	-0.02	-0.02	-0.02
PageClass	-0.07	-0.07	-0.12	-0.12	-0.07
Function	-0.18	-0.18	-0.18	-0.18	-0.19
Interaction	-0.04	-0.04	-0.03	-0.03	-0.04

(b) CLL

Table 5.1: Results for DTM (with refinement) including self-transfer cases (i.e. CSGL-R), transferring the top 10 cliques.

a listing of the top cliques in each domain), so deep transfer produces the same theories when transferring from either domain. Note that the top five cliques are also identical across WebKB and Yeast.

The results in Table 5.1 support the claims of Davis and Domingos, in that DTM improves on MSL in almost all cases. Examining self-transfer in particular, we see that self-transfer is at least competitive with other transfer scenarios on all predicates except `PageClass` (where the deficiency is due to a single outlier trial, in which refinement of the results from greedy selection led to a large decrease in AUC), and that in fact it performs significantly better than all other methods in AUC when predicting the predicate `WorkedUnder` (paired one-tail t-test, $p < 0.05$). For no predicate does the best case of cross-domain transfer perform significantly better, in AUC or CLL, than self-transfer does (paired one-tail t-test, $p > 0.10$). This is consistent with our claim that the performance gains of DTM over MSL do not depend on DTM's incorporation of source-domain knowledge.

Note that self-transfer generally matches or outperforms other transfer settings despite the limitation of having only the three folds of the training set from which to identify the top cliques, as opposed to

	CSGL-5	**CSGL-10**	**MSL**
WorkedInGenre	0.70	0.63	0.32
WorkedUnder	0.26	0.69	0.03
AdvisedBy	0.04	0.06	0.04
Linked	0.06	0.06	0.004
PageClass	0.86	0.86	0.87
Function	0.31	0.31	0.27
Interaction	0.10	0.10	0.04

(a) AUC

	CSGL-5	**CSGL-10**	**MSL**
WorkedInGenre	-0.16	-0.15	-0.30
WorkedUnder	-0.14	-0.11	-0.23
AdvisedBy	-0.04	-0.03	-0.04
Linked	-0.02	-0.02	-0.02
PageClass	-0.07	-0.07	-0.07
Function	-0.17	-0.17	-0.19
Interaction	-0.03	-0.03	-0.04

(b) CLL

Table 5.2: Results for CSGL (no refinement; transferring the top 5 and top 10 cliques) vs. MSL.

Rank	**IMDB**	**UW-CSE**	**WebKB**	**Yeast**
1	r(x,y),r(x,z)	r(x,y),r(x,z)	r(x,y),r(z,y)	r(x,y),r(z,y)
2	r(x,y),r(z,y)	r(x,y),r(z,y)	r(x,y),r(x,z)	r(x,y),r(x,z)
3	r(x,y),r(z,y),s(x,z)	r(x,y),r(z,y),s(x,z)	r(x,y),r(z,y),s(x,x)	r(x,y),r(y,x)
4	r(x,y),r(y,z),r(z,x)	r(x,y),r(z,y),s(x,x)	r(x,y),r(z,y),s(x,z)	r(x,y),r(z,y),s(x,z)
5	r(x,y),r(x,z),s(x,x)	r(x,y),r(x,z),s(x,x)	r(x,y),r(y,x)	r(x,y),r(z,y),s(x,x)
6	r(x,y),r(z,y),s(x,x)	r(x,x),s(x,x)	r(x,y),r(y,z),r(z,x)	r(x,y),r(y,z),r(z,x)
7	r(x,y),s(y,z)	r(x,y),s(x,z)	r(x,x),r(x,y),r(y,x)	r(x,y),r(x,z),r(y,x)
8	r(x,y),r(x,z),s(y,z)	r(x,x),s(x,x),t(x,y)	r(x,y),r(y,x),s(x,z)	r(x,y),r(y,x),s(x,z)
9	r(x,y),s(x,z)	r(x,y),s(y,z)	r(x,y),r(x,z),r(y,x)	r(x,x),r(x,y),r(y,x)
10	r(x,y),r(y,x),r(z,x)	r(x,x),s(x,x),t(y,x)	r(x,y),r(y,x),r(z,x)	r(x,y),r(y,x),r(z,x)

Table 5.3: The top ten cliques in each domain.

using the full four folds of source domain data which are available to the other transfer scenarios. Also recall that we modified the logical structure of each dataset so that all of its predicates had arity two, thus giving DTM the greatest possible freedom to transfer structure between all predicates. If we had allowed the single-arity `Student` and `Professor` predicates to remain uncollapsed in the UW-CSE dataset, for example, then DTM would have been unable to relate them to the analogous `PageClass` predicate in WebKB because it has arity two. By contrast, self-transfer generates cliques with arities appropriate to the predicates of each dataset.

Table 5.2 compares CSGL to MSL as standalone structure learning algorithms, with two versions of CSGL instantiating the top five and ten highest-scoring cliques respectively. Results from CSGL-5 and CSGL-10 were generally comparable, although CSGL-10 fared much better when predicting `WorkedUnder`. Note that CSGL-10 beats MSL in every case except for the `PageClass` predicate of WebKB, for which the two methods give approximately equal results. CGSL-10 also performs comparably to 10-clique self-transfer in most cases, and substantially better in the case `PageClass`, indicating that the additional, costly refinement step required by self-transfer may not be necessary in order to achieve satisfactory results.

5.4 Discussion

CSGL bears interesting similarities to the LHL algorithm of Kok and Domingos (2009), and particularly to the unlifted LHL-FindPaths variant, which constructs a knowledge base by viewing the training data as a hypergraph with constants as nodes and true ground atoms as hyperedges, interpreting the paths in the hypergraph as clauses, filtering out the clauses which are not more useful than any of their subclauses, and selecting greedily from the remaining clauses to optimize the weighted pseudo-log-likelihood (see Chapter 2.3.2). Like LHL-FindPaths, CSGL is a bottom-up structure learner which constructs clauses directly from the data, rather than following a top-down search process as MSL does. The exhaustive search step used by CSGL to generate initial clauses is equivalent to the process in LHL-FindPaths of enumerating and variabilizing paths in the unlifted hypergraph, except for the added restriction that every conjunction which LHL-FindPaths considers must have at least one support in the data (by contrast, recall that CSGL's exhaustive generation does not examine the data). Both methods evaluate clauses according to how well they represent structural regularities not found in their sub-clauses; in CSGL this is implemented by the clique-scoring process in which all but the top-scoring cliques are discarded, while in LHL this is done by simply discarding any clause having a WPLL less than one of its subclauses. Both methods consider as candidates many combinations of negated and non-negated atoms in the clauses that they generate; in CSGL this is part of the clique abstraction and instantiation process, while LHL explicitly constructs partially-negated variants of its clauses. Finally, both methods arrive at the final MLN structure by greedily selecting clauses from a list of candidates until no clause further improves the overall WPLL.

These parallels show a strong sibling resemblance between the two algorithms, and for this reason it is unlikely that CSGL represents a significant advance in the state of the art for structure learning, especially since LHL-FindPaths is already itself slow and unwieldy relative to full-fledged, lifted LHL (unfortunately, time constraints and lack of a portable implementation[1] of LHL have thus far prevented any direct comparison between CSGL and LHL). However, the similarities between CSGL and LHL-FindPaths do provide strong intuition for understanding CSGL, and therefore DTM, as a structure learning algorithm.

The structure learning view of CSGL and self-transfer helps to shed light on the results of Chapter 4, and on the performance of DTM in general. If we accept a near-isomorphism between CSGL and LHL-FindPaths, then the use of DTM for transfer learning can be seen as equivalent to a version of LHL-FindPaths in which the early stages of the algorithm are run on a potentially suboptimal domain. As long as there is enough data available in the target domain to represent a reasonable variety of cliques (and empirically, this appears to have been the case even for our smallest domains, IMDB and UW-CSE), then there is no obvious reason why we should expect to find any advantage in using another domain to perform the clique scoring instead. We would also not necessarily expect that any single hard-coded

[1]Personal communication with Stanley Kok.

set of results for these early stages (the equivalent of a universal transfer source) would be superior to executing them in the proper domain (note, indeed, that self-transfer generally outperforms the transfer from U3 and U4 considered in the previous chapter). An explanation for the observed successes of DTM, then, is that it attempts to outperform MSL by mimicking the approach of a more powerful structure learning algorithm, i.e. LHL. If this explanation is accurate, then we would expect LHL to outperform even the best instances of deep transfer for the majority of target domains.

In some sense, self-transfer can be viewed as a sort of multi-strategy learning, in which several different learning mechanisms, i.e. clique-scoring, greedy selection, and MSL-style refinement, combine to produce results superior to those attainable by any of the individual methods alone. This characterization serves to explain the apparent dilemma posed in the first section of this chapter: although we would expect a transfer algorithm to perform at its best when the source domain is related but not identical to the target domain, this does not seem to be the case for DTM, which we have shown delivers strong performance when transferring from a domain to itself. The answer is that, as explained above, DTM uses multiple learning methods on multiple datasets, and these results (DTM performance does not decrease even when applied to only a single dataset) imply that it is the multiple learning methods, not the multiple datasets, which are the source of DTM's effectiveness.

6

Simple Transfer via Second-Order Formulas

6.1 Alternative Representations for Transfer

The second-order cliques used by DTM are an abstraction of second-order formulas, which are themselves an abstraction of first-order formulas. The abstraction from first- to second-order is necessary to create domain-independent knowledge suitable for transfer, since first-order formulas are, by definition, tied to the particular predicates of the domain they occur in. However, the further abstraction from second-order formulas to cliques does not carry the same force of clear necessity; while there is an appealing symmetry between the relationship of second-order formulas to second-order cliques and the relationship of ground formulas to the cliques of the ground Markov network, DTM does not exploit this symmetry in any deep way. As a knowledge representation, cliques are also less precise than formulas: the additional level of abstraction (corresponding to the conversion to clause form and the removal of all negations) prevents them from capturing the same level of detail about specific structure in the source domain. It is unknown whether the greater generality of the clique representation is advantageous in general.

A notable disadvantage of the DTM procedure for transferring second-order cliques is that there does not appear to be any benefit to using standard MLN structure learning algorithms to learn the structure of the source domain. Davis and Domingos (2009) do consider an MSL-style beam search as a potential method for selecting source clauses for DTM, but find that because the clique-scoring process benefits from having a large set of input clauses, there is no use in devoting extra energy to searching for particularly good ones; a simple exhaustive listing actually delivers better performance. In essence, DTM's choice of second-order cliques as the main knowledge representation forces it to use the clique-scoring process as its one and and only method of structure learning in the source domain. It seems, therefore, that there might be some advantage to a transfer mechanism which eschewed cliques and the clique-scoring process, and instead used second-order formulas as the primary knowledge representation, because such a mechanism would have the flexibility to make direct use of any MLN structure learner. This would effectively reduce the problem of second-order transfer to the problem of structure learning, thus allowing for automatic improvement in transfer performance as the state of the art in structure learning progresses.

In this chapter I propose such a mechanism, a Simple Transfer algorithm for Markov logic networks (STM). Because STM uses second-order formulas as a domain-independent knowledge representation, it has the advantageous properties described above. Like DTM, STM can be viewed as a form of "deep" transfer, in that it transfers second-order structural properties and can work across domains containing different sets of predicates (however, the phrase "deep transfer" should still be taken to refer specifically to the DTM algorithm unless otherwise noted). The next section of this chapter describes the STM algorithm, and the third section contains an evaluation of STM in which its performance is compared to both MSL and DTM.

6.2 The Simple Transfer Algorithm

The simple transfer algorithm consist of four steps, which are illustrated in Figure 6.1:

1. **Structure learning in the source domain.** Any structure learner can be used to induce an MLN from the data available in the source domain. The experiments below use a version of MSL, modified (as per Chapter 3) to learn constant terms for certain predicates. The weights of the resulting MLN are not used in the transfer process, so theoretically even an ILP system could be used to learn structure, but an MLN-native system, capable of handling uncertainty and inconsistency, would be expected to deliver better results. Note that although MSL and other structure-learning algorithms generally produce MLNs in clausal form (thus, when MSL is used as the source-domain structure learner, all of the formulas involved in the transfer process are actually clauses), this is not strictly necessary for STM; STM is capable of transferring arbitrary formulas.

2. **Abstraction to second-order logic.** The formulas resulting from the structure-learning process are abstracted directly into second-order logic. The process is similar to that of deep transfer, in

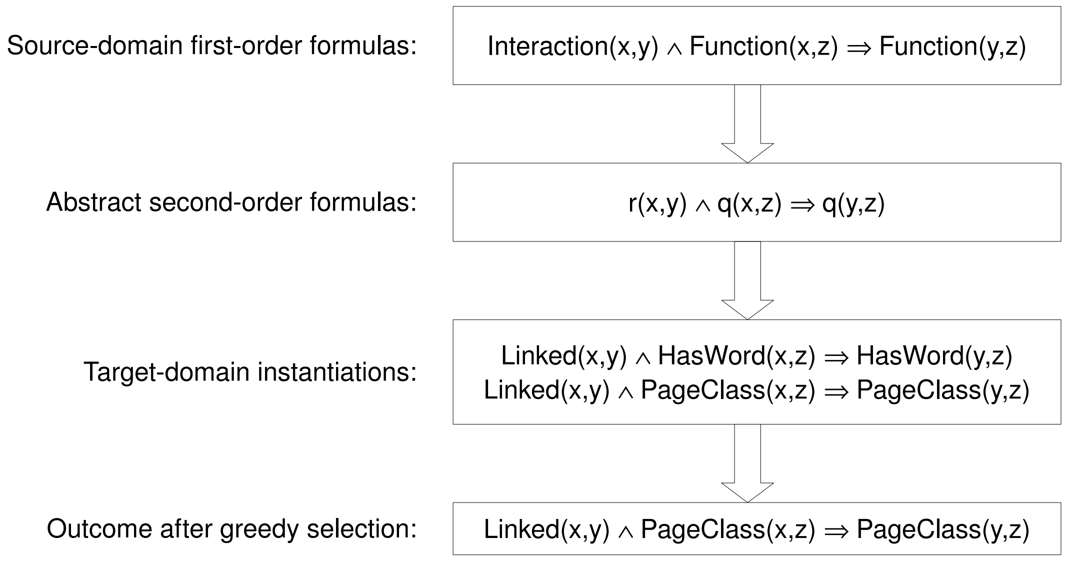
Source-domain first-order formulas:
Interaction(x,y) ∧ Function(x,z) ⇒ Function(y,z)
Abstract second-order formulas:
r(x,y) ∧ q(x,z) ⇒ q(y,z)
Target-domain instantiations:
Linked(x,y) ∧ HasWord(x,z) ⇒ HasWord(y,z)
Linked(x,y) ∧ PageClass(x,z) ⇒ PageClass(y,z)
Outcome after greedy selection:
Linked(x,y) ∧ PageClass(x,z) ⇒ PageClass(y,z)

Figure 6.1: A hypothetical transfer from the Yeast Protein to WebKB domain.

that first-order predicates are variabilized on the level of each individual formula. For example, in `r(x,y)` the `r` might refer to multiple different first-order predicates in the context of different formulas. An additional feature is that terms abstracted from first-order constants are specially marked so that they can later be re-instantiated with constants. For example, the first-order formula `PageClass(x,Student)` $\Longrightarrow$ ¬`PageClass(x,Faculty)` from the WebKB domain, which states that students are not faculty members, is abstracted to `r(x,+y)` $\Longrightarrow$ ¬`r(x,+z)`, where the '+' preceding a variable indicates that any first-order instantiation of the formula should contain a constant term in that position.

3. **Instantiation in the target domain.** Because the structure learning mechanism in the source domain is expected to produce a relatively small set of useful formulas, simple transfer does not require any mechanism for ranking or otherwise paring down the set of transferred formulas (the role which clique scoring plays in DTM). Instead, every formula abstracted from the source domain is instantiated into the target domain. The instantiation consists of all possible assignments of first-order predicates to the second-order predicate variables which maintain the type-consistency of their associated terms (e.g. in the formula `r(x,y)` ∧ `r(y,x)`, the variable `r` can only be instantiated by

a predicate whose first and second arguments are of the same type). As noted in Chapter 3, for each domain there are certain particular predicates which are marked as being able to take a constant as one of their terms; only these predicates were considered when instantiating a second-order atom containing a term marked as representing a constant. In these cases, separate versions of the formula were instantiated for every possible type-consistent assignment of constants to the appropriate terms, as well as for the case in which no constants are used. To prevent exponential blowup in the number of first-order formulas, all second-order formulas with more than three constant-derived terms were discarded prior to instantiation.

4. **Greedy selection and revision (optional).** Once instantiated in the target domain, the formulas undergo the same greedy selection process used by DTM, in which the formulas are selected one at a time until no further selection improves WPLL. This can be followed up, as in DTM, by an optional structure refinement process using MSL or any other structure learner capable of refining an existing MLN.

6.3 Experiments and Results

Tables 6.1 and 6.2 contain the results of evaluating STM from all four source domains, using greedy selection and greedy selection with refinement, respectively, to instantiate formulas in the target domains (for convenience, we will use STM-R to refer to the version of the algorithm which uses a refinement step, and STM-G to refer specifically to the version which does not). Cases of self-transfer are included. MSL is used as the source-domain structure learner; thus, the source-domain MLN is in clausal form. The STM algorithm is compared to MSL and DTM; the DTM results are for the best non-self-transfer case, transferring the top 10 cliques and using either greedy selection (DTM-G) or greedy selection with refinement (DTM-R), as appropriate for each table.[1]

The presence or absence of a refinement step appears to have a large impact on the observed performance of STM. Without refinement, STM-G outperforms MSL in terms of AUC in four of the twenty-one cases of transfer from an external source (i.e. cases which are not self-transfer), but MSL outperforms STM-G in ten additional cases. The algorithms are roughly equal (unrounded AUC within 0.005) in the seven remaining cases. The difference between the algorithms is significant in two cases, in each of which

[1]Here I use DTM-G and DTM-R to refer to the versions of the deep transfer algorithm which perform greedy selection and greedy selection with refinement, respectively. Note that the CSGL algorithm of Chapter 5 is a special case of DTM-G, while CSGL-R is the corresponding special case of DTM-R.

Domain	**Predicate**	**IMDB**	**UW-CSE**	**WebKB**	**Yeast**	**DTM-G (Best)**	**MSL**
IMDB	**WorkedInGenre**	0.46	0.61	0.61	0.07	0.61	0.32
IMDB	**WorkedUnder**	0.09	0.25	0.03	0.03	0.22	0.03
UW-CSE	**AdvisedBy**	0.01	0.04	0.05	0.01	0.06	0.04
WebKB	**Linked**	0.002	0.002	0.002	0.002	0.06	0.004
WebKB	**PageClass**	0.16	0.16	0.16	0.87	0.87	0.87
Yeast Protein	**Function**	0.18	0.18	0.37	0.35	0.32	0.27
Yeast Protein	**Interaction**	0.01	0.01	0.01	0.01	0.10	0.04

(a) AUC

Domain	**Predicate**	**IMDB**	**UW-CSE**	**WebKB**	**Yeast**	**DTM-G (Best)**	**MSL**
IMDB	**WorkedInGenre**	-0.38	-0.13	-0.13	-0.30	-0.13	-0.30
IMDB	**WorkedUnder**	-0.70	-0.08	-0.14	-0.14	-0.17	-0.23
UW-CSE	**AdvisedBy**	-0.05	-0.04	-0.05	-0.05	-0.03	-0.04
WebKB	**Linked**	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02
WebKB	**PageClass**	-0.23	-0.23	-0.10	-0.09	-0.07	-0.07
Yeast Protein	**Function**	-0.19	-0.19	-0.16	-0.16	-0.17	-0.19
Yeast Protein	**Interaction**	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04

(b) CLL

Table 6.1: Results for STM with greedy selection (STM-G), including self-transfer cases.

MSL outperforms STM-G (paired two-tail t-test, $p < 0.05$). Although STM-G sometimes produces very competent outcomes, these results suggest that it does not in general improve on the performance of standard structure learning within the target domain.

Once the refinement step is added, however, STM-R outperforms MSL (in terms of AUC) in eleven of the twenty-one non-self-transfer cases, while MSL outperforms STM-R in only three cases, and the two are roughly equal in the seven remaining cases. The difference between STM-R and MSL is significant (paired two-tail t-test, $p < 0.05$) in four cases, all of which favor STM-R. This implies that refinement provides a significant boost to the performance of simple transfer. We can interpret the process of transfer with refinement as specifying a particular starting point for the MSL beam search, and because STM-R generally outperforms MSL, this suggests that the transfer process is often providing useful information.

Of course, the more interesting benchmark is how well STM compares to DTM. To ascertain this, I compare performance of STM to the results of Chapter 5. If we allow both algorithms to refine their transferred theories, then, comparing case-by-case, STM-R outperforms DTM-R in terms of AUC in ten of twenty-one cases, DTM-R outperforms STM-R in another ten cases, and in the one remaining case

Domain	**Predicate**	**IMDB**	**UW-CSE**	**WebKB**	**Yeast**	**DTM-R (Best)**	**MSL**
IMDB	**WorkedInGenre**	0.36	0.61	0.55	0.37	0.61	0.32
IMDB	**WorkedUnder**	0.09	0.31	0.40	0.25	0.23	0.03
UW-CSE	**AdvisedBy**	0.04	0.06	0.05	0.11	0.08	0.04
WebKB	**Linked**	0.002	0.002	0.03	0.01	0.09	0.004
WebKB	**PageClass**	0.87	0.87	0.87	0.86	0.86	0.87
Yeast Protein	**Function**	0.35	0.36	0.43	0.39	0.34	0.27
Yeast Protein	**Interaction**	0.03	0.02	0.05	0.02	0.1	0.04

(a) AUC

Domain	**Predicate**	**IMDB**	**UW-CSE**	**WebKB**	**Yeast**	**DTM-R (Best)**	**MSL**
IMDB	**WorkedInGenre**	-0.22	-0.13	-0.17	-0.25	-0.13	-0.30
IMDB	**WorkedUnder**	-0.26	-0.11	-0.19	-0.20	-0.17	-0.23
UW-CSE	**AdvisedBy**	-0.04	-0.03	-0.03	-0.03	-0.03	-0.04
WebKB	**Linked**	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02
WebKB	**PageClass**	-0.07	-0.07	-0.08	-0.07	-0.07	-0.07
Yeast Protein	**Function**	-0.16	-0.16	-0.16	-0.16	-0.18	-0.19
Yeast Protein	**Interaction**	-0.04	-0.04	-0.04	-0.04	-0.03	-0.04

(b) CLL

Table 6.2: Results for STM with greedy selection and refinement (STM-R), including self-transfer cases.

they perform roughly equally. None of these differences are significant (paired two-tail t-test, $p > 0.05$). If the refinement step is omitted, then DTM-G wins against STM-G in eleven cases (ten of which are significant), STM-G wins in four cases (one if which is significant), and the two are roughly equal in the remaining six cases. For three of the seven predicates tested (namely `WorkedUnder`, `AdvisedBy`, and `Function`), the best case of STM-R transfer produced an AUC higher than the best case of DTM-R (though the difference was significant only for the `Function` predicate). The best cases of the two algorithms were roughly equal on another two predicates (`WorkedInGenre` and `PageClass`), while the best case of DTM-R outperformed STM-R on the remaining two predicates (`Linked` and `Interaction`), although for neither of those two predicates was the difference significant.

The comparison of STM-R to DTM-R shows that given a refinement step, STM is competitive with DTM, and in fact sometimes outperforms it, though (as can be seen from the tables) its performance also varies much more widely across different source domains than does DTM's. Interestingly, the refinement step seem to be much more necessary for STM than it is for DTM; without refinement, the overall performance of STM drops much more sharply than does that of DTM. This is probably because the set of formulas instantiated by STM is generally much smaller than the set of clauses instantiated by DTM (given ten cliques of up to eight features each, our version of DTM might transfer the equivalent of up to eighty second-order clauses, while in these experiments STM never transferred more than seven clauses), so it works better as a starting point for refinement than as a source for greedy selection.

Examining cases of STM self-transfer shows that that self-transfer under STM-R surpasses the performance of MSL for almost every predicate (`Interaction` is the sole exception). This implies that the extra step of abstracting and reinstantiating the formulas produced by MSL in the source domain allowed MSL to continue refining them to produce superior results, presumably because it forced MSL to consider a superset of the original formulas which included all formulas having similar structure. This suggests that MSL might be improved by the addition of a periodic step which performed this sort of abstraction/reinstantiation. Although the boost provided by the abstraction/reinstantiation process may also have been a factor in other (non self-transfer) cases of simple transfer, note that under both STM-G and STM-R, self-transfer is rarely as effective as the most effective source of external transfer, and often it is much less effective. This is the opposite of the findings of the last chapter with respect to DTM, and I interpret this as a sign that STM with external transfer is, indeed, successfully incorporating knowledge from the source domain.

6.4 Discussion

The STM algorithm was motivated in part by questions about the clique representation used by DTM, in particular whether its additional level of abstraction leads to better results or simply throws away useful information. These results show that the clique representation is at least not obviously inferior to the more specific second-order formula representation used by STM, since DTM-G significantly outperforms STM-G in the majority of cases. Although STM-G works quite well in a few situations, DTM-G delivers more consistently strong performances, perhaps due to the generality of the clique representation (although, as explored in Chapter 4, the general properties represented by a set of cliques are not necessarily tied very strongly to the source domain). It is only after the refinement step that STM is competitive with DTM. Because each clique implicitly represents a larger and much more general set of first-order clauses than a second-order formula would (since each clique has many different states, each of which is in itself equivalent to a different second-order clause), this suggests that part of the value of the clique representation is that the use of cliques can save significant time during the transfer process: it is possible to obtain good results simply by selecting from the set of instantiated clauses, without needing to engage in a costly refinement process.

On the other hand, our implementation of STM uses only MSL as the source-domain structure learning, while DTM uses a combination of exhaustive search and clique scoring. Since I showed in Chapter 5 that the latter approach has an advantage over MSL for most of the predicates tested, this gives the clique representation something of an unfair advantage: it is being paired with a more effective structure learner than is the formula representation used by STM. It is certainly conceivable that a version of STM which used LHL or a similar algorithm as the source-domain structure learner would deliver improved performance, although it has not been possible to test this possibility for this thesis. Since we have seen that STM-R outperforms MSL, its base structure learner, in many cases, it would be interesting to investigate whether it is generally true that any given version of STM will (for some reasonable choice of source domain) outperform the base structure learning algorithm it is constructed from. If so, this would give STM significant advantages over DTM in the future, since the development of increasingly effective structure learning algorithms would only boost the performance of STM.

7

Conclusions and Future Work

This thesis has explored a variety of methods for structure and transfer learning in Markov logic networks, focusing on modifications to the deep transfer algorithm of Davis and Domingos (2009). I began by exploring possibilities for transfer from multiple sources, concluding that there was no significant advantage for multiple-source transfer over single-source transfer on the domains considered. In my observations of the top cliques of four different domains, I confirmed that there was significant overlap (as suggested by Davis and Domingos (2009)), raising the possibility that there might be a set of cliques which represent near-universally useful sources for transfer. I used two different methods to construct sets of cliques which were highly ranked in most or all of the domains under consideration, and evaluated them as sources for transfer, finding that both were often competitive with the best cases of transfer from a specific domain.

Since these results did show, however, that the choice of source domain does make a difference when performing deep transfer (and in many cases the best individual domain outperformed both sets of common cliques), I proposed a novel approach which sidesteps the problem of finding the best source domain. Specifically, I hypothesized that the cliques of the target domain itself ought to function as effective sources for transfer and then showed empirically that this method of "self-transfer" performs generally as well as transfer from the best external sources. These results led me to argue that DTM self-transfer constitutes a novel method for standalone MLN structure learning. I showed that this method bears many similarities to LHL, an existing state-of-the art structure learning technique, and concluded that the deep transfer algorithm may be most naturally interpreted as a form of multi-strategy structure learning, rather than as an algorithm for knowledge transfer.

Finally, I proposed simple transfer, a new algorithm for transfer learning in Markov logic networks which makes use of existing MLN structure learners in both the source and target domains. The STM algorithm transfers a more specific form of structure than does DTM, and exceeds the performance of DTM in a variety of cases, though it performs less well in many other cases. Because it uses an external structure learner, STM also has the potential to improve its performance as new algorithms for structure learning are developed.

7.1 Future Work

Several obvious directions for additional work involve expanding on the experiments in this thesis. How effective are U3 and U4 as transfer sources for domains other than the four from which they were generated? Does the performance of STM improve when using a more sophisticated structure learning algorithm (e.g. LHL) in the source domain? These questions are motivated in Chapters 4 and 6, respectively.

Another goal for future research would be to search for circumstances under which cross-domain transfer might still be beneficial under DTM. One such case might occur when trying to learn quickly given very little data in the target domain, which is a traditional strength of transfer learning. This particular task, however, has already seen more specialized algorithms devoted to it (e.g. Mihalkova and Mooney (2009)), so it may not constitute a compelling justification for DTM.

Other directions might include identifying other transfer learning mechanisms for which a similar self-transfer "trick" could be applied to produce improved within-task learning performance, and also further exploring the connection between clique scoring and hypergraph pathfinding. Given their strong similarities, would it be possible to integrate the most sophisticated elements of CSGL (e.g. the clique-scoring process) with those of LHL (e.g. hypergraph lifting through clustering), combining the most effective elements of each to create a better overall method for structure learning?

A final opportunity for future work is inspired by a particular weakness which is notably shared by all existing algorithms for transfer learning in MLNs, including STM, DTM, and TAMAR. Specifically, each of these algorithms depends strongly on the surface logical form of the datasets they are asked to perform transfer between; none of which are capable of making the appropriate connections between a dataset representing web page class with a two-arity predicate, e.g. `PageClass(page,pclass)` (this is the approach taken by WebKB in this thesis), and one which represents the same concept as a set of single-arity predicates `StudentPage(page)`, `CoursePage(page)`, etc., even though the two forms are logically equivalent. Similarly, they are unable to relate the `TaughtBy(course,person,quarter)` predicate, which appears in some versions of the UW-CSE data, to the `TaughtBy(course,person)` predicate used in this thesis, although the two might be expected to behave similarly in many cases. This property does not prevent them from performing well in many cases — especially on the datasets used for this thesis, which were modified especially to minimize these concerns — but it does suggest that, at least in some sense, neither STM, DTM, nor TAMAR is really getting at the essence of the matter. An effective MLN transfer algorithm ought to be invariant to any particular choice of logical representation, if only because there is no guarantee that real-world datasets from different sources will make the same decisions as to which representations to use. Future research might be able to develop a transfer learner which is less sensitive to these distinctions.

Appendix A

Learned MLNs

This appendix contains examples of the clauses and weights learned by MSL and CSGL for each of the four domains discussed in this thesis. Each table gives the MLN which was learned by training on the first three folds of the specified dataset, using the specified algorithm.

A.1 IMDB

Weight	Formula
-5.04788	`workedUnder(a1,a2)`
-0.000892385	`genre(a1,a2)`
-7.4258e-05	`workedUnder(a1,a1)`
0.589336	¬`role(a1,Actor)` ∨ `genre(a1,a2)` ∨ ¬`genre(a1,a3)` ∨ ¬`genre(a4,a3)`
-0.0375249	`genre(a1,a2)` ∨ `genre(a3,a2)` ∨ ¬`genre(a1,a4)` ∨ ¬`genre(a3,a4)`
0.00112482	`genre(a1,a2)` ∨ `genre(a3,a4)` ∨ ¬`genre(a1,a5)` ∨ ¬`genre(a3,a2)`
0.0082254	`genre(a1,a2)` ∨ `genre(a3,a2)` ∨ ¬`genre(a1,a4)` ∨ ¬`genre(a3,a5)`
0.0106387	¬`role(a1,Actor)` ∨ `genre(a1,a2)` ∨ ¬`genre(a1,a3)` ∨ ¬`genre(a1,a4)` ∨ ¬`genre(a5,a3)`
3.16416	¬`role(a1,Actor)` ∨ `genre(a1,a2)` ∨ `genre(a3,a4)` ∨ ¬`genre(a1,a5)` ∨ ¬`genre(a3,a5)`
0.235249	¬`workedUnder(a1,a1)` ∨ `genre(a1,a2)` ∨ `genre(a1,a3)` ∨ ¬`genre(a1,a4)` ∨ ¬`genre(a5,a4)`

Table A.1: Cliques learned using MSL on the first three folds of the IMDB dataset.

Weight	Formula
-0.983939	`genre(a1,a2)`
-1.84027	`workedUnder(a1,a2)`
7.88469	¬`workedUnder(a1,a2)` ∨ ¬`role(a2,Actor)`
6.13453	¬`role(a1,Actor)` ∨ ¬`genre(a1,a2)`
3.12152	¬`workedUnder(a1,a2)` ∨ ¬`genre(a1,a3)`
1.07589	¬`workedUnder(a1,a2)` ∨ ¬`workedUnder(a1,a3)`
-5.67677	¬`workedUnder(a1,a2)` ∨ ¬`movie(a3,a1)` ∨ ¬`movie(a3,a2)`

Table A.2: Cliques learned using CSGL on the first three folds of the IMDB dataset.

A.2 UW-CSE

Weight	Formula
-4.13795	`advisedBy(a1,a2)`
-0.0689903	`advisedBy(a1,a1)`
-2.37679	`personlevel(a1,Student)`
2.37679	`personlevel(a1,Prof)`
-1.98876	`advisedBy(a1,a2)` ∨ `personlevel(a1,Student)` ∨ ¬`personlevel(a2,a3)`
-2.73012	`advisedBy(a1,a2)` ∨ `inPhase(a2,a3)` ∨ `personlevel(a2,Prof)` ∨ ¬`personlevel(a2,Student)`

Table A.3: Cliques learned using MSL on the first three folds of the UW-CSE dataset.

Weight	Formula
0.271905	`advisedBy(a1,a2)`
0.957868	`personlevel(a1,Student)`
-0.631464	`personlevel(a1,Prof)`
-8.3145	`personlevel(a1,a2)` ∨ ¬`personlevel(a1,a3)`
4.36165	¬`advisedBy(a1,a2)` ∨ ¬`personlevel(a2,Student)`
2.05605	¬`advisedBy(a1,a2)` ∨ `inPhase(a1,a3)`

Table A.4: Cliques learned using CSGL on the first three folds of the UW-CSE dataset.

A.3 WebKB

Weight	**Formula**
3.10087	`PageClass(a1,Person)`
-3.88424	`PageClass(a1,ResearchProject)`
-2.65059	`PageClass(a1,Course)`
-6.953	`PageClass(a1,Staff)`
-6.93197	`PageClass(a1,Faculty)`
-6.86145	`PageClass(a1,Student)`
-3.32053	`PageClass(a1,Department)`
-7.09801	`Linked(a1,a2)`
-3.54712	`PageClass(a1,Course)` $\vee$ $\neg$`PageClass(a1,Person)`
2.86116	`PageClass(a1,a2)` $\vee$ `PageClass(a1,Staff)` $\vee$ `PageClass(a1,Faculty)` $\vee$ `PageClass(a1,Student)` $\vee$ $\neg$`PageClass(a1,Person)`

Table A.5: Cliques learned using MSL on the first three folds of the WebKB dataset.

Weight	Formula
-2.31374	`PageClass(a1,Student)`
-1.59818	`Linked(a1,a2)`
3.38395	`PageClass(a1,Person)`
-3.88284	`PageClass(a1,ResearchProject)`
-2.23963	`PageClass(a1,Faculty)`
1.15867	`PageClass(a1,Staff)`
-3.3067	`PageClass(a1,Department)`
-2.86408	`PageClass(a1,Course)`
11.0142	`PageClass(a1,Person)` ∨ ¬`PageClass(a1,Student)`
8.40782	`PageClass(a1,Person)` ∨ ¬`PageClass(a1,Staff)`
-6.44947	`PageClass(a1,Staff)` ∨ ¬`PageClass(a1,Student)`
7.97889	¬`PageClass(a1,Student)` ∨ ¬`PageClass(a1,Faculty)`
-4.94361	`PageClass(a1,Staff)` ∨ ¬`PageClass(a1,Faculty)`
-9.29807	`PageClass(a1,Person)` ∨ `PageClass(a1,Faculty)`
-2.91168	`Linked(a1,a2)` ∨ `Linked(a2,a1)`

Table A.6: Cliques learned using CSGL on the first three folds of the WebKB dataset.

A.4 Yeast Protein

Weight	Formula
-4.50674	`function(a1,Func_id_1001003)`
-1.08985	`function(a1,Func_id_1)`
-4.63587	`function(a1,Func_id_38)`
-2.98542	`function(a1,Func_id_10001009)`
-4.16601	`function(a1,Func_id_16021)`
-1.39615	`function(a1,Func_id_42)`
-4.06702	`function(a1,Func_id_20009016009)`
-2.96655	`function(a1,Func_id_40)`
-3.88858	`function(a1,Func_id_41001)`
-1.95007	`function(a1,Func_id_11)`
-4.44516	`function(a1,Func_id_1001006)`
-2.00657	`function(a1,Func_id_32)`
-4.70347	`function(a1,Func_id_32005)`
-2.03155	`function(a1,Func_id_20001)`
-4.99425	`function(a1,Func_id_40010002002)`
-0.973373	`function(a1,Func_id_16)`
-4.99425	`function(a1,Func_id_42027)`
-2.40035	`function(a1,Func_id_12)`
-4.9185	`function(a1,Func_id_1005001007)`
-4.63587	`function(a1,Func_id_18001)`
-4.99425	`function(a1,Func_id_1025)`
-2.37792	`function(a1,Func_id_34011)`
-4.9185	`function(a1,Func_id_30005002)`
-1.69151	`function(a1,Func_id_11002)`
-4.84478	`function(a1,Func_id_43001002)`
-0.995262	`function(a1,Func_id_10)`
-4.9185	`function(a1,Func_id_36020035)`
-2.66548	`function(a1,Func_id_18002)`
-4.99425	`function(a1,Func_id_1001009001)`
-3.5056	`function(a1,Func_id_34)`
-4.99425	`function(a1,Func_id_1001011003)`
-1.35941	`function(a1,Func_id_20)`
-4.84478	`function(a1,Func_id_1001011004)`
-2.73819	`function(a1,Func_id_2)`
-4.99425	`function(a1,Func_id_1001003001)`
-2.35629	`function(a1,Func_id_43001003)`
-4.99425	`function(a1,Func_id_1001005)`
-3.62951	`function(a1,Func_id_20009018009)`
-6.19855	`interaction(a1,a2)`
-1.03535	`function(a1,Func_id_14)`
0.366821	`interaction(a1,a1)`
-2.55746	`function(a1,Func_id_30)`
6.56858	`function(a1,Func_id_99)`
4.79196	¬`function(a1,a2)` ∨ ¬`function(a1,Func_id_99)`
-3.82411	¬`function(a1,Func_id_42)` ∨ ¬`function(a1,Func_id_40)` ∨ ¬`function(a1,Func_id_43001003)`

Table A.7: Cliques learned using MSL on the first three folds of the Yeast Protein dataset.

Weight	**Formula**
-4.81153	function(a1,Func_id_38)
-1.31397	function(a1,Func_id_1)
-4.36002	function(a1,Func_id_16021)
-3.15925	function(a1,Func_id_10001009)
-4.2321	function(a1,Func_id_20009016009)
-1.3001	function(a1,Func_id_42)
-4.06321	function(a1,Func_id_41001)
0.165813	function(a1,Func_id_40)
-4.64761	function(a1,Func_id_1001006)
-2.14493	function(a1,Func_id_11)
-4.85751	function(a1,Func_id_32005)
-2.20462	function(a1,Func_id_32)
-5.19631	function(a1,Func_id_40010002002)
-0.463194	function(a1,Func_id_20001)
-5.18921	function(a1,Func_id_42027)
-0.984035	function(a1,Func_id_16)
-5.12105	function(a1,Func_id_1005001007)
-2.58208	function(a1,Func_id_12)
-5.19041	function(a1,Func_id_1025)
-4.84094	function(a1,Func_id_18001)
-5.10948	function(a1,Func_id_30005002)
-0.171735	function(a1,Func_id_34011)
-5.05663	function(a1,Func_id_43001002)
-4.69674	function(a1,Func_id_1001003)
-5.12781	function(a1,Func_id_36020035)
-1.26076	function(a1,Func_id_14)
-5.1708	function(a1,Func_id_1001009001)
1.00096	function(a1,Func_id_43001003)
-5.20785	function(a1,Func_id_1001011003)
-1.35228	function(a1,Func_id_99)
-5.01548	function(a1,Func_id_1001011004)
-3.78893	function(a1,Func_id_20009018009)
-5.21235	function(a1,Func_id_1001003001)
-0.45604	function(a1,Func_id_30)
-5.19631	function(a1,Func_id_1001005)
-1.8933	function(a1,Func_id_11002)
-1.24337	interaction(a1,a2)
-2.84769	function(a1,Func_id_18002)
-2.92405	function(a1,Func_id_2)
-1.22411	function(a1,Func_id_10)
0.784473	function(a1,Func_id_20)
-3.6696	function(a1,Func_id_34)
-3.77977	function(a1,Func_id_40) ∨ function(a1,Func_id_43001003)
-2.61367	interaction(a1,a2) ∨ interaction(a2,a1)
-2.85907	function(a1,Func_id_34011) ∨ function(a1,Func_id_30)
-2.82638	function(a1,Func_id_20001) ∨ function(a1,Func_id_20)
3.19944	¬function(a1,Func_id_16) ∨ ¬function(a1,Func_id_99)

Table A.8: Cliques learned using CSGL on the first three folds of the Yeast Protein dataset.

References

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician*, pages 179–195. 14

Biba, M., Ferilli, S., and Esposito, F. (2008). Structure learning of Markov logic networks through iterated local search. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, pages 361–365, Amsterdam, The Netherlands. IOS Press. 3, 17

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28:41–75. 35, 38

Craven, M. and Slattery, S. (2001). Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1-2):97–119. 25

Davis, J. and Domingos, P. (2009). Deep transfer via second-order Markov logic. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, Montreal, Quebec. 3, 14, 17, 20, 21, 22, 24, 25, 27, 28, 30, 31, 36, 39, 45, 53

Domingos, P., Kok, S., Poon, H., Richardson, M., and Singla, P. (2006). Unifying Logical and Statistical AI. *AAAI-06*, pages 2–7. 17

Domingos, P. and Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool. 14

Getoor, L. and Taskar, B. (2007). *Introduction to statistical relational learning*. The MIT Press. 3

Kok, S. and Domingos, P. (2005). Learning the structure of Markov logic networks. In *Proceedings of the 21st International Conference on Machine Learning (ICML-05)*, pages 441–448. 3, 14, 15, 16, 24

Kok, S. and Domingos, P. (2009). Learning Markov logic network structure via hypergraph lifting. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, Montreal, Quebec. 3, 5, 16, 25, 41

Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., and Domingos, P. (2009). The Alchemy System for Statistical Relational AI. `http://alchemy.cs.washington.edu`. Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. 23

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York. 2

Lowd, D. and Domingos, P. (2007). Efficient weight learning for markov logic networks. In *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, pages 200–211, Berlin, Heidelberg. Springer-Verlag. 14

Mewes, H. W., Frishman, D., Gildener, U., Mannhaupt, G., Mayer, K., Mokrejs, M., Morgenstern, B., Minsterktter, M., Rudd, S., and Weil, B. (2002). Mips: a database for genomes and protein sequences. *Nucleic Acids Res*, 30:31–34. 25

Mihalkova, L., Huynh, T., and Mooney, R. J. (2007). Mapping and revising Markov logic networks for transfer learning. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI-22)*, pages 608–614. 22

Mihalkova, L. and Mooney, R. (2009). Transfer learning from minimal target data by mapping across relational domains. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*. 22, 54

Mihalkova, L. and Mooney, R. J. (2007). Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*, pages 625–632, New York, NY, USA. ACM. 3, 17, 25

Mihalkova, L. S. (2009). *Learning with Markov Logic Networks*. PhD thesis, University of Texas, Austin. v, 9, 11, 13, 14

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. 1

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann. 3, 8, 10

Raedt, L. D. and Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, 26:99–146. 17

Richards, B. and Mooney, R. (1992). Learning Relations by Pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA. 17

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62(1-2):107–136. 3, 8, 9, 10, 11, 13, 23, 25

Thrun, S. and Mitchell, T. M. (1995). Learning one more thing. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1217–1225. 17

Torrey, L. and Shavlik, J. (2009). Transfer learning. In Soria, E., Martin, J., Magdalena, R., Martinez, M., and Serrano, A., editors, *Handbook of Research on Machine Learning Applications*. IGI Global. 3, 17, 35

Wolpert, D. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390. 30