In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*. pp. 608-614, Vancouver, Canada, July 2007.

Mapping and Revising Markov Logic Networks for Transfer Learning

Lilyana Mihalkova and **Tuyen Huynh** and **Raymond J. Mooney**
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
{lilyanam,hntuyen,mooney}@cs.utexas.edu

Abstract

Transfer learning addresses the problem of how to leverage knowledge acquired in a source domain to improve the accuracy and speed of learning in a related target domain. This paper considers transfer learning with Markov logic networks (MLNs), a powerful formalism for learning in relational domains. We present a complete MLN transfer system that first autonomously maps the predicates in the source MLN to the target domain and then revises the mapped structure to further improve its accuracy. Our results in several real-world domains demonstrate that our approach successfully reduces the amount of time and training data needed to learn an accurate model of a target domain over learning from scratch.

Introduction

Traditional machine learning algorithms operate under the assumption that learning for each new task starts from scratch, thus disregarding any knowledge gained previously. In related domains, this *tabula rasa* approach would waste data and computer time to develop hypotheses that could have been recovered faster from previously acquired knowledge. Transferring previous knowledge could not only speed up learning but also increase its accuracy. *Transfer learning* addresses the problem of how to leverage previous knowledge in order to improve the efficiency and accuracy of learning in a new domain, related to the original one.

We consider transfer learning with *Markov logic networks* (MLNs), i.e. when an MLN learned for the source domain is used to aid learning of an MLN for the target domain. MLNs are a powerful formalism that combines the expressiveness of first-order logic with the flexibility of probability (Richardson & Domingos 2006). There are two aspects to learning an MLN: the structure, or first-order clauses, and the weights. While weight learning is relatively quick, structure learning is very computationally intensive. Therefore, we focus on MLN structure learning because it could particularly benefit from transfer.

We view transferring an MLN to a new domain as consisting of two subtasks: *predicate mapping* and *theory refinement*. In general, the set of predicates used to describe data in the source and target domains may be partially or completely distinct. Therefore, the first transfer task is to establish a mapping from predicates in the source domain to predicates in the target domain. For example, consider transferring an MLN learned to model data about individuals and their relationships in an academic department to modeling data from the International Movie Database (IMDB). A predicate mapping between the two domains might establish that directors are like professors, actors are like students, and movies are like research papers. Similar predicate mappings are produced by analogy systems such as the *Structure Mapping Engine* (SME) (Falkenhainer, Forbus, & Gentner 1989). Once a mapping is established, clauses from the source domain can be translated to the target domain. However, these clauses may not be completely accurate and may need to be revised, augmented, and re-weighted in order to properly model the target data. This step is similar to previous work in theory refinement (Richards & Mooney 1995; Wrobel 1996; Ramachandran & Mooney 1998), except the theory to be revised is *learned* in a previous domain rather than manually constructed by a human expert.

This paper presents novel techniques for mapping and refining an MLN in order to transfer it to a new domain. A complete MLN transfer system was developed by augmenting the Alchemy MLN software (Kok *et al.* 2005). Experimental results on several real-world relational datasets demonstrate that our approach successfully reduces the amount of time and training data needed to learn an accurate model of a target domain by exploiting an existing MLN for a related source domain.

Background

General Terminology and Notation

First-order logic distinguishes among four types of symbols—constants, variables, predicates, and functions (Russell & Norvig 2003). Constants describe the objects in a domain and can have types. Variables act as placeholders to allow for quantification. Predicates represent relations in the domain, such as *WorkedFor*. Function symbols represent functions of tuples of objects. The arity of a predicate or a function is the number of arguments it takes. Each argument can have a type that specifies the type of constant that can be used to ground it. Here, we assume that the domains contain no functions. We denote constants by strings starting

1.5	Director(A)∨Actor(A)
0.1	MovieMember(M, A)∨¬Director(A)
1.3	¬WorkedFor(A, B)∨¬Director(A)
0.5	Director(A)∨¬MovieMember(M,A)∨¬WorkedFor(B, A)

Figure 1: Example of an MLN

with lower-case letters, variables by single upper-case letters, and predicates by strings starting with upper-case letters. A term is a constant, a variable, or a function applied to terms. Ground terms contain no variables. An atom is a predicate applied to terms. A positive literal is an atom, and a negative literal is a negated atom. We will call a ground literal, i.e. one that contains only ground terms, a *gliteral*. A world is an assignment of truth values to all possible gliterals in a domain. A formula consists of literals connected by logical connectives (i.e. $\vee$ and $\wedge$). A formula in clausal form, also called a clause, is a disjunction of literals. A clause with only one positive literal is called a definite clause. Using the fact that $\neg p \vee q$ is logically equivalent to $p \Rightarrow q$, we can rewrite any clause as an implication, without modifying its meaning. For example, we can rewrite the clause Director(A) $\vee$ Actor(A) in two different ways: 1) $\neg$Actor(A) $\Rightarrow$ Director(A) or 2) $\neg$Director(A) $\Rightarrow$ Actor(A). We will call the literal(s) on the left side of the implication *antecedents*, and the literal on the right side *conclusion*. Clauses whose antecedents are not satisfied hold "trivially."

Markov Logic Networks

An MLN consists of a set of weighted formulae and provides a way of softening first-order logic by making situations, in which not all formulae are satisfied, less likely but not impossible (Richardson & Domingos 2006). Here we assume that the formulae are in clausal form (the Alchemy software makes this conversion automatically). Figure 1 gives an example of an MLN for one of our domains. More formally, let $\mathbf{X}$ be the set of all propositions describing a world (i.e. all gliterals formed by grounding the predicates with the constants in the domain), $\mathcal{F}$ be the set of all clauses in the MLN, w_i be the weight associated with clause $f_i \in \mathcal{F}$, $\mathcal{G}_{f_i}$ be the set of all possible groundings of clause f_i with the constants in the domain, and Z be the normalizing partition function. Then the probability of a particular truth assignment $\mathbf{x}$ to $\mathbf{X}$ is given by the formula $P(\mathbf{X} = \mathbf{x}) = (1/Z) \exp\left(\sum_{f_i \in \mathcal{F}} w_i \sum_{g \in \mathcal{G}_{f_i}} g(\mathbf{x})\right)$ (Richardson & Domingos 2006). The value of $g(\mathbf{x})$ is either 1 or 0, depending on whether g is satisfied. Thus the quantity $\sum_{g \in \mathcal{G}_{f_i}} g(\mathbf{x})$ counts the number of groundings of f_i that are true given the current truth assignment to $\mathbf{X}$.

In order to perform inference over a given MLN, L, one needs to ground it into its corresponding a *Markov network* (Pearl 1988). As described by Richardson and Domingos (2006), this is done by including a node for every possible gliteral of the predicates in L and a feature for each grounding of a clause in L. The nodes appearing together in a ground clause form cliques. To calculate the probability that each of a set of query gliterals is true, one can perform Gibbs sampling over the Markov network. Gibbs sampling starts by assigning a truth value to each query gliteral. It then proceeds in rounds to resample a value for gliteral X, given the truth values of its Markov blanket MB_X (i.e. the nodes with which it participates in ground clauses) using the formula:

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

Here, $S_X(x, \mathbf{m}) = \sum_{g_i \in \mathcal{G}_X} w_i g_i(X = x, \text{MB}_X = \mathbf{m})$, where $\mathcal{G}_X$ is the set of ground clauses in which X appears and $\mathbf{m}$ is the current truth assignment to MB_X.

Kok and Domingos introduced the current state-of-the-art MLN structure learning algorithm (2005), which we call KD after its authors. KD can use either beam search or shortest-first search but we will compare to the beam-search version.[1] KD performs several iterations of beam search, and after each iteration adds to the MLN the best clause found. Clauses are evaluated using a weighted pseudo log-likelihood measure (WPLL), introduced in (Kok & Domingos 2005), that sums over the log-likelihood of each node given its Markov blanket, weighting it appropriately to ensure that predicates with many gliterals do not dominate the result. The beam search in each iteration starts from all single-vliteral clauses. It generates candidates by performing all possible revisions to the initial clauses, keeps the best $beamSize$ clauses, from which it generates new candidates by performing all possible revisions, keeps the best $beamSize$ and continues in this way until candidates stop improving the WPLL. At this point, the best candidate found is added to the MLN, and a new beam search iteration begins. Weights need to be learned for a given structure before its WPLL can be computed. Weight-training can be efficiently implemented as an optimization procedure such as L-BFGS used by Richardson and Domingos (2006) and therefore we do not transfer the weights. Because KD can start learning either from scratch or from a provided MLN, it can be used for revision. However, KD does not include any predicate mapping capability.

Relational Pathfinding

Relational pathfinding (RPF) is a data-driven approach to rule discovery designed to overcome plateaus and local maxima (Richards & Mooney 1992). We will use it in the revision step to discover relationships specific to the target domain. RPF views the relational domain as a graph G in which the constants are the vertices and two constants are connected by an edge if they appear together in a true gliteral. The true gliterals are those stated to hold in the data, while the rest are false. Intuitively, RPF forms clauses in which the conclusion is a particular true gliteral, and the antecedents consist of gliterals that define a path in the relational graph G. These clauses are then variablized. More specifically, RPF searches G for an alternate path of length at least 2 between any two constants, c_1 and c_2, connected by an edge. If such path is found, it is transformed into a

[1] The shortest-first search constructs candidates in the same way but conducts a more complete search, which, however, requires longer training times.

Director(jack) Actor(jill) MovieMember(movie1, jack) MovieMember(movie1, jill) MovieMember(movie2, jill) WorkedFor(jill, jack)

Figure 2: Example relational database

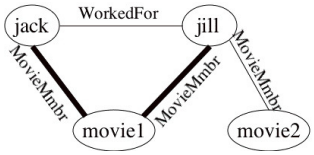

Figure 3: Example of a relational graph

clause as follows. The antecedents are formed as a conjunction of the predicates that label each edge in the path. The literal that labels the edge connecting c_1 and c_2 becomes the conclusion. Hill-climbing search is used to further improve the clause by possibly adding unary predicates to the antecedents.

For example, suppose Figure 2 lists all true facts in the domain. Figure 3 shows the relational graph for this domain. The highlighted edges form an alternative path between $jack$ and $jill$, from which we construct the clause MovieMember(T,A) $\wedge$ MovieMember(T,B) $\Rightarrow$ WorkedFor(A,B). Hill-climbing search might lead to the addition of Actor(A) and Director(B) to the antecedents.

MLN Structure Transfer

In MLN structure transfer, first the source MLN is mapped to the target domain, and, second, its clauses are revised.

Predicate Mapping

The goal of this step is to find the best way to map a source MLN into a target MLN. The quality of a mapping is measured by the performance of the mapped MLN on the target data, as estimated by the WPLL score. There are two general approaches to mapping. One is to establish a mapping for each source predicate to a target predicate and then use it to translate the entire source MLN. This is called *global* mapping. The other approach, called *local* mapping, is to find the best mapping of each source clause separately by constructing a mapping only for the predicates that appear in that clause, independent of how other clauses were mapped. In most cases, finding the best global mapping is computationally prohibitive because the size of the search space is exponential in the number of predicates in the source domain. In general, the local mapping approach is more scalable than the global one since the number of predicates in a single source clause is usually smaller than the total number of source predicates. In this work, we will focus on finding the best local predicate mapping.

To find the best predicate mapping for a clause, we use exhaustive search through the space of all legal mappings. As we will demonstrate in the Experimental Results section, this search is very manageable in our domains. A mapping is legal if each source predicate in a given clause is mapped either to a compatible target predicate or to the "empty" predicate, which erases all literals of that source predicate from the clause. Two predicates are compatible if they have the same arity and the types of their arguments are compatible with the current type constraints. For any legal mapping, a type in the source domain is mapped to at most one corresponding type in the target domain, and the type constraints are formed by requiring that these type mappings are consistent across all predicates in the clause. For example, if the current type constraints are empty, then the source predicate *Publication(title,person)* is considered to be compatible with the target predicate *Gender(person,gend)*, and the type constraints *title* $\rightarrow$ *person* and *person* $\rightarrow$ *gend* are added to the current type constraints. All subsequent predicate mappings within the current clause must conform to these constraints. For example, with these constraints, the source predicate *SamePerson(person,person)* is compatible with the target predicate *SameGender(gend,gend)* but not compatible with the target predicate *SamePerson(person,person)*. After a legal mapping is established, we evaluate the translated clause by calculating the WPLL of an MLN consisting of only this translated clause. The best predicate mapping for a clause is the one whose translated clause has the highest WPLL score. This process is repeated to find the best predicate mapping for each source clause. Algorithm 1 finds a legal mapping for a source clause. A recursive version of this procedure is used to find all the legal mappings of a source clause. Fig 4 shows the best predicate mapping found by the algorithm for a given source clause.

Note that because the predicate mapping algorithm may sometimes choose to map a predicate to an "empty" predicate, the entire structure is not necessarily always mapped to the target domain.

Algorithm 1 Find a legal mapping for a source clause

```
1:  procedure LEGAL_MAPPING(srcClause, tarPreds)
2:      predsMapping ← ∅
3:      typeConstraints ← ∅
4:      repeat
5:          Pick an unmapped source predicate srcPred
6:          for each unmapped target predicate tarPred do
7:              if isCompatible(srcPred, tarPred) then
8:                  Add this mapping to predsMapping
9:                  Update typeConstraints
10:                 Exit this for loop
11:             end if
12:         end for
13:     until All predicates in srcClause are mapped
14:     return predsMapping
15: end procedure
```

Revising the Mapped Structure

Next we describe how the mapped structure is revised to improve its fit to the data. The skeleton of the revision algorithm has three steps and is similar to that of FORTE (Richards & Mooney 1995), which revises first-order theories.

1. **Self-Diagnosis:** The purpose of this step is to focus the

Source clause:
Publication(T, A) ∧ Publication(T, B) ∧ Professor(A)
∧ Student(B) ∧¬SamePerson(A, B) ⇒ AdvisedBy(B, A)
Best mapping:

Publication(title,person)	→ MovieMember(movie,person)
Professor(person)	→ Director(person)
Student(person)	→ Actor(person)
SamePerson(person,person)	→ SamePerson(person,person)
AdvisedBy(person,person)	→ WorkedFor(person,person)

Figure 4: An output of the predicate mapping algorithm

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

2. **Structure Update:** Clauses marked as too long are shortened, while those marked as too short are lengthened.
3. **New Clause Discovery:** Using RPF, new clauses are found in the target domain.

Next, we describe each step in more detail.

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

The algorithm considers a view of each ground clause $C \in \mathcal{C}_X$ in which the literal corresponding to P^* is treated as a conclusion. If a clause contains more than one literal of P^*, all possible rewrites are produced. C can be placed in one of four bins with respect to X and the current truth assignments to the rest of the gliterals. These bins consider all possible cases of the antecedents being satisfied and the conclusion being correct. We label a clause as Relevant if the antecedents are satisfied and Irrelevant otherwise. Similarly, we mark a clause as Good if its conclusion is correct and Bad if the conclusion is incorrect. The four bins are defined by all possible ways of marking a clause as Relevant/Irrelevant and Good/Bad.

Let v be the actual value of X. This value is known from the data, even though for the purposes of sampling we have set it to unknown. As an illustration, we will use some groundings of the clauses in Figure 1 with respect to the data in Figure 2 listing the current truth assignments to the gliterals (the ones present are true; the rest are false). Let X =Director(jack) with $v = true$.

- **[Relevant; Good]** This bin contains clauses in which the antecedents are satisfied and the conclusion drawn is correct. For example, let C be ¬Actor(jack)⇒Director(jack). Alternatively, the clauses in this bin hold true only if X has value v, the value it has in the data.
- **[Relevant; Bad]** This bin contains clauses whose antecedents are satisfied but the conclusion drawn is incorrect. For example, let C be ¬MovieMember(movie2, jack) ⇒ ¬Director(jack). Alternatively, this bin contains clauses that are only satisfied if X has value $\neg v$, the negation of its correct value in the data.
- **[Irrelevant; Good]** This bin contains clauses whose antecedents are not satisfied, and therefore the clauses do not "fire," but if they were to fire, the conclusion drawn would be incorrect. For example, let C be WorkedFor(jack,jill) ⇒ ¬Director(jack). Thus the clauses in this bin hold regardless of the value of X in the data; however, the literal corresponding to X in C is true only if X has value $\neg v$.
- **[Irrelevant; Bad]** The clauses in this bin are those whose antecedents are not satisfied, but if the clauses were to fire, the conclusion would be correct. For example, let C be MovieMember(movie2, jack) ∧WorkedFor(jill, jack) ⇒ Director(jack). We can alternatively describe the clauses in this bin as ones that hold regardless of the value of X and in which the literal corresponding to X in C is true only if X has value v.

This taxonomy is motivated by Equation (1). The probability of $\mathbf{X} = \mathbf{x}$ is increased only by clauses in the **[Relevant;Good]** bin and is decreased by clauses in the **[Relevant;Bad]** bin. Clauses in the other two bins do not have an effect on this equation because their contribution to the numerator and denominator cancels out. However, if some of the literals other than X in an **[Irrelevant;Bad]** clause, are deleted so that X's value becomes crucial, it will be moved to the **[Relevant;Good]** bin. Similarly, if we add some literals to a **[Relevant;Bad]** clause so that it starts to hold regardless of the value of X, it will enter the **[Irrelevant;Good]** bin and will no longer decrease the probability of X having its correct value.

As the probability of a gliteral is recalculated in each iteration of Gibbs sampling, for each clause in which the gliteral participates, we count the number of times it falls into each of the bins. Finally, if a clause was placed in the **[Relevant;Bad]** bin more than p percent of the time, it is marked for lengthening and if it fell in the **[Irrelevant; Bad]** bin more than p percent of the time, it is marked for shortening. We anticipated that in the highly sparse relational domains in which we tested, clauses would fall mostly in the **[Irrelevant; Good]** bin. To prevent this bin from swamping the

other ones, we set p to the low value of 10%. This process is repeated for each predicate, P^*, in the target domain.

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

New Clause Discovery The revision procedure can update clauses transferred from the source domain but cannot discover new clauses that capture relationships specific only to the target domain. To address this problem, we used RPF to search for new clauses in the target domain. The clauses found by RPF were evaluated using the WPLL, and the ones that improved the overall score were added to the MLN. RPF and the previous structure update step operate independently of each other; in particular, the clauses discovered by RPF are not diagnosed or revised. However, we found that better results are obtained if the clauses discovered by RPF are added to the MLN before carrying out the revisions.

Experimental Methodology

We compared the performance of the following systems. KD run from scratch (ScrKD) in the target domain; KD used to revise a source structure translated into the target domain using our automatic predicate mapping procedure (TrKD); and our complete transfer system using automatic predicate mapping and the revision procedure (TAMAR, for Transfer via Automatic Mapping And Revision). In early experiments we tested systems that did not use RPF and found that adding it never hurt performance, although it did not always help. Space considerations did not allow us to include a complete treatment of the effect of RPF.

We used three real-world relational domains—IMDB, UW-CSE, and WebKB. Each dataset is broken down into *mega-examples*, where each mega-example contains a connected group of facts. Individual mega-examples are independent of each other. The IMDB database is organized as five mega-examples, each of which contains information about four movies, their directors, and the first-billed actors who appear in them. Each director is ascribed genres based on the genres of the movies he or she directed. The Gender predicate is only used to state the genders of actors. The UW-CSE database was first used by Richardson and Domingos (2006).[2] The database is divided into mega-examples based on five areas of computer science. It lists facts about people in an academic department (i.e. Student, Professor) and their relationships (i.e. AdvisedBy).[3] The WebKB database contains information about entities from the "University Computer Science Department" data set, compiled by Craven et al. (1998). The original dataset contains web pages from four universities labeled according to the entity they describe (e.g. student, course), as well as the words that occur in these pages. Our version of WebKB contains the predicates Student(A), Faculty(A), CourseTA(C, A), CourseProf(C, A), Project(P, A) and SamePerson(A, B). The textual information is ignored. This data contains four mega-examples, each of which describes one university. The following table provides additional statistics about these domains:

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

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

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

We used the two metrics employed by Kok and Domingos (2005), the area under the precision-recall curve (AUC) and the conditional log-likelihood (CLL). The AUC is useful because it demonstrates how well the algorithm predicts the few positives in the data. The CLL, on the other hand, determines the quality of the probability predictions output by the algorithm. To calculate the AUC and CLL of a given MLN, one needs to perform inference over it, providing some of the gliterals in the test mega-example as evidence and testing the predictions for the remaining ones. We used the MC-SAT inference algorithm (Poon & Domingos 2006) and, like Kok and Domingos (2005), tested for the gliterals of each of the predicates of the domain in turn, providing the rest as evidence, and averaging over the results.

Learning curves for each performance measure were generated using a leave-1-mega-example-out approach, averaging over k different runs, where k is the number of mega-examples in the domain. In each run, we reserved a different mega-example for testing and trained on the remaining $k-1$, which were provided one by one. All systems observed the same sequence of mega-examples. Because of space constraints, rather than presenting the complete learning curves, we summarize them using two statistics: the transfer ra-

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

[3] Our results on this dataset are not comparable to those presented by Kok and Domingos (2005) because due to privacy issues we only had access to the published version of this data, which differs from the original (Personal communication by S. Kok).

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

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

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

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

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

We also present results on the training times needed by all systems, the number of clauses they considered in their search, the time to construct the mapping, and the number of legal mappings. Timing runs within the same transfer experiment were conducted on the same dedicated machine.

Experimental Results

In terms of AUC (Table 1), both transfer systems improve over ScrKD in all but one experiment. Neither transfer learner consistently outperforms the other on this metric. Table 2 shows that transfer learning always improves over learning from scratch in terms of CLL, and TAMAR's performance is better than that of TrKD in all cases. Except for the last line in Table 1, the learning curves of the transfer systems (not shown) always dominate over those of ScrKD. Moreover, as can be seen from Table 3, TAMAR trains faster than TrKD, and both transfer systems are faster than ScrKD (however, note that the training time of the transfer systems does not include the time necessary to learn the source structure). TAMAR also considers fewer candidate clauses during its beam search. According to a t-test performed for each point on each of the learning curves, at the 95% level with sample size 5 per point, these differences were significant in 15 out of 20 cases for speed and 18 out of 20 for number of candidates. TrKD considers more candidates than ScrKD but takes less time to train. This can happen if TrKD considers more candidates earlier in the learning curves when each candidate is evaluated faster on less data. Table 3 also demonstrates that the time taken by the mapping algorithm is only a tiny fraction of the total training time. The average total number of legal mappings over the five experiments was 49.4, and the average number of legal mappings per clause was 5.31. Because there are few possible legal mappings, the exhaustive search through the space of legal mappings is tractable. Figure 5 shows a sample learning curve in the UW-CSE → IMDB experiment. Here we additionally tested the performance of systems that do not use the automatic mapping but are provided with an intuitive hand-constructed global mapping that maps Student → Actor, Professor → Director, AdvisedBy/TempAdvisedBy → WorkedFor, Publication → MovieMember, Phase → Gender, and Position → Genre. The last two mappings are motivated by the observation that Phase in UW-CSE applies only to Student and Gender in IMDB applies only to Actor, and similarly Position and Genre apply only to Professor and Director respectively. The systems using the automatic mapping perform much better because the flexibility of local mapping allows the source knowledge to adapt better to the target domain.

Experiment	ScrKD	TrKD	TAMAR
WebKB→IMDB	62.23	32.20(0.89)	11.98(0.89)
UW-CSE→IMDB	62.23	38.09(9.18)	15.21(9.10)
UW-KB→IMDB	62.23	40.67(9.98)	6.57(9.99)
WebKB→UW-CSE	1127.48	720.02(6.71)	13.70(6.75)
IMDB→UW-CSE	1127.48	440.21(42.48)	34.57(42.28)

Table 3: Average (over all learning curve points) total training time in **minutes**. The numbers in parentheses give the average number of **seconds** needed to construct the predicate mapping.

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

Table 4: Average (over all learning curve points) number of candidate clauses considered.

Related Work

Transfer learning has been studied in two main settings. In multi-task learning, the algorithm is given all domains simultaneously during training and can build common structure shared by the learned models (Caruana 1997; Niculescu-Mizil & Caruana 2005). Our setting differs in that the learner encounters the domains one at a time. It has been studied for a variety of problems, including text classification (Raina, Ng, & Koller 2006) and reinforcement learning (Torrey *et al.* 2005; Taylor, Whiteson, & Stone 2007). The latter work also automatically maps the source and target tasks.

The predicate mapping task is similar to the case of *abstraction mapping* in SME (Falkenhainer, Forbus, & Gen-

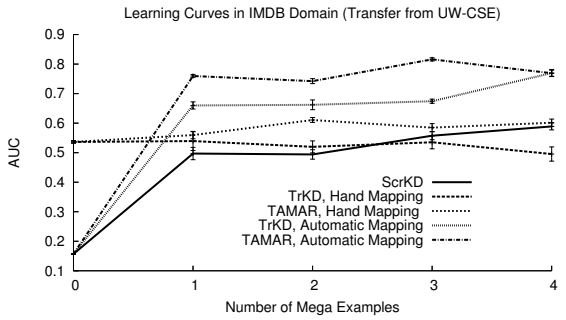

Figure 5: Learning curves for AUC in UW-CSE → IMDB. The zeroth points are obtained by testing the MLN provided to the learner at the start.

tner 1989). However, *abstraction mapping* generates candidate mappings based on an analogy between some generalized objects in the two domains. Moreover, SME requires a specific set of matching rules for each pair of domains. In contrast, our predicate mapping is based on the compatibility between the predicates of the two domains and no matching rules are required. The mappings between both types and predicates in the source domain to those in the target domain are constructed completely automatically.

Our algorithm approaches transferring a mapped MLN to the target domain as a revision problem. Thus it is related to revision algorithms for other models, such as Bayesian Networks (Ramachandran & Mooney 1998), Bayesian Logic Programs (Paes *et al.* 2005), and first-order knowledge bases (Richards & Mooney 1995; Wrobel 1996).

Conclusions and Future Work

We have presented a complete transfer learning system, TAMAR, that autonomously maps a source MLN to the target domain and then revises its structure to further improve its performance. Our empirical results demonstrate that TAMAR successfully maps the transferred knowledge so that it can be used to improve the accuracy of the learned model. Because TAMAR diagnoses the mapped structure and only revises its incorrect portions, it learns significantly faster and considers many fewer candidates than the current best MLN learner.

Future directions include applying TAMAR to new transfer scenarios; developing predicate mapping search techniques for domains where the number of predicates makes exhaustive search prohibitive; and creating measures for the similarity between two domains to determine whether transfer would be beneficial.

Acknowledgements

We thank Pedro Domingos, Matt Taylor, and the three anonymous reviewers for their helpful comments. We also thank the Alchemy team at the University of Washington for their great help with Alchemy. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and managed by the Air Force Research Laboratory (AFRL) under contract FA8750-05-2-0283.The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of DARPA, AFRL, or the United States Government. The second author was also supported by the Vietnam Education Foundation Fellowship. Most of the experiments were run on the Mastodon Cluster, provided by NSF Grant EIA-0303609.

References

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

Cohen, P. R.; Chang, Y.; and Morrison, C. T. 2007. Learning and transferring action schemas. In *IJCAI-2007*.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 1998. Learning to extract symbolic knowledge from the World Wide Web. In *AAAI-98*.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1989. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence* 41(1):1–63.

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

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

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

Paes, A.; Revoredo, K.; Zaverucha, G.; and Costa, V. S. 2005. Probabilistic first-order theory revision from examples. In *ILP-05*.

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

Poon, H., and Domingos, P. 2006. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI-2006*.

Raina, R.; Ng, A. Y.; and Koller, D. 2006. Constructing informative priors using transfer learning. In *ICML-2006*.

Ramachandran, S., and Mooney, R. J. 1998. Theory refinement for Bayesian networks with hidden variables. In *ICML-98*.

Richards, B. L., and Mooney, R. J. 1992. Learning relations by pathfinding. In *AAAI-92*.

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

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

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

Taylor, M. E.; Whiteson, S.; and Stone, P. 2007. Transfer via inter-task mappings in policy search reinforcement learning. In *AAMAS-07*.

Torrey, L.; Walker, T.; Shavlik, J.; and Maclin, R. 2005. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML-05*.

Wrobel, S. 1996. First order theory refinement. In De Raedt, L., ed., *Advances in Inductive Logic Programming*. Amsterdam: IOS Press.