Cross-Domain Action-Model Acquisition for Planning Via Web Search

Hankz Hankui Zhuo[a] and **Qiang Yang**[b] and **Rong Pan**[a] and **Lei Li**[a]

[a]Department of Computer Science, Sun Yat-sen University, Guangzhou, China
{zhuohank,panr,lnslilei}@mail.sysu.edu.cn

[b]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
qyang@cse.ust.hk

Abstract

Applying learning techniques to acquire action models is an area of intense research interest. Most previous works in this area have assumed that there is a significant amount of training data available in a planning domain of interest, which we call target domain, where action models are to be learned. However, it is often difficult to acquire sufficient training data to ensure that the learned action models are of high quality. In this paper, we develop a novel approach to learning action models with limited training data in the target domain by transferring knowledge from related auxiliary or source domains. We assume that the action models in the source domains have already been created before, and seek to transfer as much of the the available information from the source domains as possible to help our learning task. We first exploit a Web searching method to bridge the target and source domains, such that transferrable knowledge from source domains is identified. We then encode the transferred knowledge together with the available data from the target domain as constraints in a maximum satisfiability problem, and solve these constraints using a weighted MAX-SAT solver. We finally transform the solutions thus obtained into high-quality target-domain action models. We empirically show that our transfer-learning based framework is effective in several domains, including the International Planning Competition (IPC) domains and some synthetic domains.

Introduction

AI planning techniques often require a given set of action models as input. Creating action models, however, is a difficult task that costs much manual effort. The problem of action-model acquisition has drawn a lot of interest from researchers in the past. For instance, McCluskey et al. (Blythe et al. 2001; McCluskey, Liu, and Simpson 2003) designed a system to interact with a human expert to generate action models. Amir (Amir 2005) introduced a tractable and exact technique for learning action models known as Simultaneous Learning and Filtering, where the state observations were needed for learning. Yang et al. (Yang, Wu, and Jiang 2007) presented a framework for automatically discovering STRIPS (Fikes and Nilsson 1971) action models from a set of successfully observed plans; Zhuo et al. (Zhuo et al. 2009) proposed to learn method preconditions and action models simultaneously for HTN planning, assuming that a set of hierarchical structures was given, just to name a few. Despite the success of the previous systems, they are all based on the assumption that there are enough training examples for learning high-quality action models.

In this paper, we consider the problem of learning action models with only limited amount of data. Given just a limited amount of training data, we make use of action-models already created beforehand in other related domains, which are called source domains, to help us learn actions in a target domain. The motivation for this problem is that, in many situations, it is usually difficult or expensive to collect a large amount of training data to learn action models in the target domain, while it is easier to find related knowledge or information in some auxiliary domains. If we can find a bridge between these domains, transfer learning can then be used to help learn the action models in the target domain. For instance, in the NASA rovers planning domain, it is not an esay task to collect a large number of plan traces for learning action models for the robotic rovers to explore Mars. However, there may exist one or more related domains, such as robots exploring in a coal mine, or driving truck *driverlog*[1], where action models have already been created. These domains provide helpful knowledge for learning action models in the rovers planning domain. In this paper, we aim at identifying similar parts of these source domains to help improve the learning of the target domains.

A key challenge in this problem is how to *bridge* the target and source domains. We observe that although the actions in the source domain and the target domain are different, some of them are similar in their semantics. For instance, the action "*navigate*" in the *rovers*[1] domain (viewed as the target domain), is similar to the action "*walk*" in the *driverlog*[1] domain (viewed as the source domain). The former action indicates that a rover navigates from one position to another, while the latter indicates that a driver walks from one location to another. Both of them describe the action of changing locations, and this relationship may have already been recorded by human editors on some Web sites. As a result, these two actions are linked through some Web pages, which allow us to build a mapping between these domains via Web search. In this paper, we propose to bridge the tar-

[1]http://planning.cis.strath.ac.uk/competition/

get domain and a related source domain by searching Web pages related to the target domain and the source domain, and then building a mapping between them by calculating the similarity between their corresponding Web pages.

In this paper, we present a novel algorithm to **L**earn **A**ction models with transferring knowledge from a related source domain via **W**eb **s**earch; this system is called LAWS. We focus on learning STRIPS action models (Fikes and Nilsson 1971) rather than the full PDDL action models (Fox and Long 2003). In our algorithm LAWS, we build a similarity function between two sets of Web pages, and then calculates the similarity between the target domain and a related source domain using the similarity function after searching Web pages related to them. After that, we build a set of weighted constraints, which we call *web constraints*, based on the similarity calculated. We also build other constraints based any available example plan traces in the target domain, which we call *state constraints*, *action constraints* and *plan constraints*, respectively. We solve all the constraints (web/state/action/plan constraints) using a weighted MAX-SAT solver (Borchers and Furman 1998), and generate target-domain action models based on the solution to the constraint satisfaction problem.

We organize the paper as follows. We introduce the related work in the next section, and then give the formulation of our learning problem. After that, we present our algorithm LAWS and evaluate it in the experiment section. Finally, we conclude the paper together with the future work.

Related Work

Yang et al. presented the ARMS system (action-relation modeling system) (Yang, Wu, and Jiang 2007) for automatically discovering STRIPS (Fikes and Nilsson 1971) action models from a set of successfully observed plans. The ARMS system automates the process of knowledge acquisition for action model learning in previous works, where a computer system interacted with human experts to generate the needed action models (Blythe et al. 2001; McCluskey, Liu, and Simpson 2003). Amir (Amir 2005) presented a tractable and exact technique for learning action models known as Simultaneous Learning and Filtering, where the state observations were needed for learning. Zhuo et al. (Zhuo et al. 2010) proposed an algorithm called LAMP to learn complex action models with quantifiers and implications using Markov Logic Networks (MLNs) (Richardson and Domingos 2006). Zettlemoyer et al. (Zettlemoyer, Pasula, and Kaelbling 2005) investigated into learning a model of the effects of actions in noisy stochastic worlds. Despite the success of these previous learning systems, they learn action models in one domain with the assumption that there are large enough training data available in the task domain. Zhuo et al. (Zhuo, Yang, and Li 2009) studied the way to learn action models by building mappings between source domains and the target domain, which focused on syntax-level mapping. In contast, in this paper, we aim to explore the semantic mapping using Web searching for action-model learning.

Our work is related to transfer learning (Pan and Yang 2010; Caruana 1997). We exploit Web search technology to connect two domains. In the past, some previous research works considered learning common sense knowledge from the Web to assist model training, which was inspired by the research of Etzioni et al. (Etzioni et al. 2004). Bollegala et al. (Bollegala, Matsuo, and Ishizuka 2009) proposed a relational similarity measure, using a Web search engine, to compute the similarity between semantic relations implied by two pairs of words. Perkowitz et al. (Perkowitz et al. 2004) proposed to mine the natural language descriptions of activities from ehow.com as labeled data, and translate them into probabilistic collection of object terms. Zheng et al. (Zheng, Hu, and Yang 2009) developed a mapping between activities in two domains by learning a similarity function via Web search.

The MAX-SAT problem (Borchers and Furman 1998) for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that minimizes the number of unsatisfied clauses (or equivalently, that maximizes the number of satisfied clauses). In propositional logic a variable x_i may take values *false* or *true*. A literal l_i is a variable x_i or its negation $\bar{x}_i$. A clause is a disjunction of literals, and a CNF formula ϕ is a conjunction of clauses. The length of a clause is the number of its literals. The size of ϕ, denoted by $|\phi|$, is the number of all its clauses. An assignment of truth values to the propositional variables satisfies a literal x_i if x_i takes the value *true* and satisfies a literal $\bar{x}_i$ if x_i takes the value *false*, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. An empty clause, denoted by $\Box$, contains no literals and cannot be satisfied. An assignment for a CNF formula ϕ is complete if all the variables occurring in ϕ have been assigned; otherwise, it is partial. MaxSatz (LI, Manya, and Planes 2007; LI et al. 2009) implements a lower bound computation method that consists of incrementing the lower bound by one for every disjoint inconsistent subset that can be detected by unit propagation. Moreover, the lower bound computation method is enhanced with failed literal detection. The variable selection heuristics takes into account the number of positive and negative occurrences in binary and ternary clauses.

Problem Formulation

A planning problem can be described as a triple $P = (\Sigma, s_0, g)$, where s_0 is an initial state, g is a goal, and Σ is defined by $\Sigma = (S, \mathcal{A}, \gamma)$, where S is a set of states, $\mathcal{A}$ is a set of action models, and γ is a transition function defined by $\gamma : S \times \mathcal{A} \rightarrow S$. A solution to a planning problem is an action sequence (or a plan) denoted as $(a_1, a_2, \ldots, a_n)$, where a_i is an action. An *action model* is defined as $(a, \text{PRE}, \text{ADD}, \text{DEL})$, where a is an action name with zero or more parameters, which is called *action schema*, PRE is a precondition list specifying the condition under which the action a can be applied, ADD is an adding list and DEL is a deleting list. Notice that in this paper we focus on the STRIPS action model description.

A *plan trace* t is an action sequence with partially observed states, i.e., $t = (s_0, a_1, s_1, \ldots, a_n, s_n)$, where s_i can be a partially observed state and a_i is an action. We define

I P T: action models from a source domain ***driverlog.***
(:action **walk**
 :parameters (?driver - driver ?loc-from - location ?loc-to - location)
 :precondition (and (at ?driver ?loc-from) (path ?loc-from ?loc-to))
 :effect (and (not (at ?driver ?loc-from)) (at ?driver ?loc-to))))
...

I P T: action schemas, predicates and plan traces from the target domain ***rovers.***
Action schemas:
navigate (?x - rover ?y - waypoint ?z - waypoint)
sample soil (?x - rover ?s - store ?p - waypoint)
...
Predicates:
(at ?x - rover ?y - waypoint) (empty ?x - store) (full ?x - store) ...
Plan traces:
 (at rover0, waypoint0), (empty store0), ⋯
navigate(rover0, waypoint0, waypoint1) sample soil(rover0, store0, waypoint1)
 (at rover0, waypoint1) (full store0)
...

O TP T: action models in the target domain ***rovers.***
(:action **navigate**
 :parameters (?x - rover ?y - waypoint ?z - waypoint)
 :precondition (and (can traverse ?x ?y ?z) (available ?x) (at ?x ?y) (visible ?y ?z))
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
...

Figure 1: An example of inputs and outputs

our learning problem as: given a set of action schemas, a set of predicates, a small set of plan traces $\mathcal{T}$ from the target domain D_t, and a set of action models $\mathcal{A}_s$ from a source domain D_s, how do we construct a set of action models in the target domain D_t? We assume that action models from D_s were already created before, which means they are available to be used to help learn the unknown action models in the target domain. Here is an example of our learning problem in Figure 1 for illustration.

Our Transfer Learning Framework

In this section, we present our algorithm `LAWS` to learn action models. We show an overview of `LAWS` in Algorithm 1. We first build a similarity function to bridge a source domain and the target domain, and generate a set of weighted constraints according to the similarity function (steps 1 and 2 of Algorithm 1). After that, we build a set of weighted constraints from the plan traces from the target domain (step 3). Finally, we solve all weighted constraints with a weighted MAX-SAT solver, and convert the solving result to a set of action models (steps 4 and 5).

We will give the detailed description of each step of Algorithm 1 in the following.

Building Constraints via Web Searching

In steps 1 and 2 of Algorithm 1, we aim to build constraints from a source domain via Web searching. We will first describe how to build the similarity function (step 1), and then how to build weighted constraints based on the similarity function (step 2) in the following two subsections.

Algorithm 1 An overview of our `LAWS` algorithm

```
input: (1) A set of action models $\mathcal{A}_s$ of a source domain, (2) an action schema set $A$ and a predicate set $P$ in the target domain, (3) a set of plan traces $\mathcal{T}$ from the target domain.
output: A set of action models $\mathcal{A}_t$ of the target domain.
1: build the similarity function build_sim_fun($\mathcal{A}_s$, $A$, $P$);
2: generate weighted constraints according to the similarity function;
3: build weighted constraints according to $\mathcal{T}$;
4: solve all the weighted constraints;
5: convert the solving result to $\mathcal{A}_t$;
6: return $\mathcal{A}_t$;
```

Building the Similarity Function For each predicate $p \in P$ and each action $a \in A$, if $\text{PARA}(p) \subseteq \text{PARA}(a)$ holds, then p is probably a precondition or an effect of a, where $\text{PARA}(p)$ (or $\text{PARA}(a)$) denotes a set of parameters of p (or a). Thus, we build a set of predicate-action pairs from the target domain

$$PA_t = \{\langle p, a\rangle | p \in P \wedge a \in A \wedge (\text{PARA}(p) \subseteq \text{PARA}(a))\}.$$

Furthermore, we also build sets of predicate-action pairs from the source domain

$$PA_s^{pre} = \{\langle p, a\rangle | a \in \mathcal{A}_s \wedge p \in \text{PRE}(a)\},$$
$$PA_s^{add} = \{\langle p, a\rangle | a \in \mathcal{A}_s \wedge p \in \text{ADD}(a)\},$$
$$PA_s^{del} = \{\langle p, a\rangle | a \in \mathcal{A}_s \wedge p \in \text{DEL}(a)\}.$$

For each predicate-action pair $t \in PA_t$ and $s \in PA_s^{pre}$ (or $s \in PA_s^{add}$, $s \in PA_s^{del}$), we can exploit Web searching to extract Web pages related to them. For instance, using the example in Figure 1, for a pair $\langle at, navigate\rangle$ from the target domain *rovers*, where *at* is a predicate and *navigate* is an action, we can search with query "navigate and at". Then we can get a list of search results on the page. By clicking all search results, we can get a set of Web pages. Likewise, we can extract Web pages by searching a pair $\langle at, walk\rangle$ from the source domain *driverlog* with query "walk and at". Note that we ignore the difference of the searching results from different predicate-action orders, such as "at and walk" and "walk and at". Although the Web pages contain a lot of information, only a small amount of it is related to the semantics of the searched query. So we apply the information retrieval to retrieve the useful information for each Web page.

In particular, for each Web page, we first extract the plain text as a document d_i. Such a document d_i can be further processed as a vector x_i, each dimension of which is the term frequency-inverse document frequency (tf-idf) (Jones 1972) of each word w of d_i:

$$tf\text{-}idf_{i,w} = \frac{n_{i,w}}{\sum_l n_{i,l}} \times \log \frac{|\{d_i\}|}{|\{d_i : w \in d_i\}|},$$

where $n_{i,w}$ is the number of occurrences of the word w in document d_i. Besides, $|\{d_i\}|$ is the total number of collected documents, and $|\{d_i : w \in d_i\}|$ is the number of documents where the word w appears. The first term $\frac{n_{i,w}}{\sum_l n_{i,l}}$ of the tf-idf equation is called **term frequency**, which denotes the frequency of the word w that appears in the document d_i. If the word w appears more frequently in the document d_i, then $n_{i,w}$ is larger, and thus the whole term is larger. The second term

$$\log \frac{|\{d_i\}|}{|\{d_i : w \in d_i\}|}$$

is called **inverse document frequency**, which denotes the inverse document frequency for the word w. If the word w appears in more documents of the corpus, then $|\{d_i : w \in d_i\}|$ is larger, and thus the whole term is smaller. For example, for the word "the", it is used in almost all the documents, but it is a stop word without any meaning. Hence, its inverse document frequency will vanish to zero, thus the tf-idf value of the word approaches to zero. It means that such a word does not encode any semantics of the searched keyword, so it can be removed from the Web data's feature vector.

Therefore, for a predicate-action pair s, e.g., $\langle at, walk\rangle$, we can search it and get a set of documents

$$\mathcal{D}_s = \{x_i | i = 1, \ldots, m_s\}$$

, with each x_i as a tf-idf vector. Similarly, for another pair t, e.g., $\langle at, navigate\rangle$, we can get another set of documents

$$\mathcal{D}_t = \{y_i | i = 1, \ldots, m_t\}$$

, with each y_i as a tf-idf vector.

After extracting the Web data $\mathcal{D}_s$ and $\mathcal{D}_t$, we measure the similarity between the action-predicate pairs s and t. Note that a possible choice to calculate the similarity between two data distributions is using the Kullback-Leibler (KL) divergence (Kullback and Leibler 1951). However, generally the Web text data are high-dimensional and it is hard to model the distributions over the two different data sets. Therefore, we propose to use the Maximum Mean Discrepancy (MMD) (Borgwardt et al. 2006) to calculate the similarity, which can directly measure the distribution distance without the density estimation.

Definition 1: Let $\mathcal{F}$ be a class of functions $f\colon X \to \mathbb{R}$. Let p and q be Borel probability distributions, and let $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ be samples composed of independent and identically distributed observations drawn from distributions p and q respectively. Then the Maximum Mean Discrepancy (empirical estimation) is

$$MMD[\mathcal{F}, X, Y] = \sup_{f \in \mathcal{F}} \left(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i)\right).$$

Considering the universal reproducing kernel Hilbert spaces (RKHS), we can interpret the function f as the feature mapping function of a Gaussian kernel (Borgwardt et al. 2006), and we have the following result given by (Borgwardt et al. 2006)

$$MMD^2[\mathcal{F}, X, Y] = \frac{1}{m(m-1)}\sum_{i \neq j}^{m} k(x_i, x_j) + \frac{1}{n(n-1)}\sum_{i \neq j}^{n} k(y_i, y_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j),$$

where $k(x_i, y_j)$ is defined as

$$k(x_i, y_j) = \exp\left(-\frac{\|x_i - y_j\|^2}{2\sigma^2}\right),$$

where σ is the kernel width for the Gaussian kernel function.

Given the Web data D_s and D_t, we can finally have the similarity function between s and t defined as

$$similarity(s, t) := MMD^2[\mathcal{F}, D_s, D_t].$$

Generating Web Constraints

After defining the similarity function, we present the process of generating the constraints based on the function in step 2 of Algorithm 1. We call this kind of constraints *web constraints*.

For each predicate-action pair $s = \langle p_s, a_s\rangle$ in PA_s^{pre}, and each pair $t = \langle p_t, a_t\rangle$ in PA_t, we generate a constraint, $p_t \in \text{PRE}(a_t)$, and associate it with $similarity(s, t)$ as its weight. Intuitively, $similarity(s, t)$ measures the degree of the constraint, that p_t is a precondition of a_t, being satisfied, since p_s is a precondition of a_s in the source domain. Likewise, we can generate weighted constraints $p_t \in \text{ADD}(a_t)$ and $p_t \in \text{DEL}(a_t)$ with respect to PA_s^{add} and PA_s^{del}.

Building STRIPS Semantics Constraints

Besides the constraints extracted from the source domain, we also build constraints based on the plan traces from the target domain, which is presented by step 3 of Algorithm 1. We will build three kinds of constraints, i.e., *state constraints*, *action constraints* and *plan constraints*, each of which will be presented in the following.

State constraints By observation, we find that if a predicate p frequently appears just before an action a is executed, then p is probably a precondition of a. We formulate this idea as the constraint

$$\text{PARA}(p) \subseteq \text{PARA}(a) \Rightarrow p \in \text{PRE}(a),$$

where PARA(p) (PARA(a)) means a set of parameters of p (a), the condition of $\text{PARA}(p) \subseteq \text{PARA}(a)$ is required by the STRIPS description, that the action should contain all the parameters of its preconditions or effects. Likewise, if a predicate p frequently appears just after an action a is executed, then p is probably an effect of a. We also formulate this idea as the constraint

$$\text{PARA}(p) \subseteq \text{PARA}(a) \Rightarrow p \in \text{ADD}(a).$$

We calculate the weights of this kind of constraints with occurrences of them in the plan traces. In other words, if a predicate p has occurred just before a is executed for three times in the plan traces, and p's parameters are included by a's parameters, then the weight of the constraint that $p \in$ PRE(a) is 3.

Action constraints We would like to ensure that the action models learned are consistent, i.e., an action a should not generate two conflict conditions such as p and $\neg p$ at the same time. Such an idea can be formulated by the following constraint,

$$p \in \text{ADD}(a) \Rightarrow p \notin \text{DEL}(a)$$

Since we would like to require that this kind of constraints to be satisfied maximally, we assign the weights of this kind of constraints with the largest value of the weights of state constraints.

Plan constraints Each plan trace in the target domain provides us the information that it can be executed successfully from the first action to the last one. In other words, actions in a plan trace are all executable, i.e., their preconditions are all satisfied before they are executed. This information can be represented by the following constraint,

$$p \in \text{PRE}(a_i) \Rightarrow p \in \text{EXE}(i-1)$$

where $p \in \text{EXE}(i-1)$ means p either exists in the initial state and is not deleted by the action sequence $\langle a_1, \ldots, a_{i-1}\rangle$, or is added by some action a' prior to a_i and is not deleted by actions between a' and a_i.

Furthermore, consider an observed state o_i, which is composed of a set of predicates. We require that each predicate in o_i should either be newly added by actions or exist in the initial state. Likewise, we formulate the idea with the following constraint,

$$p \in o_i \Rightarrow p \in \text{EXE}(i-1)$$

We also require that this kind of constraints to be maximally satisfied, and assign the largest value of the weights of state constraints as the weights of this kind of constraints.

Solving All Constraints

In step 4, we solve all the weighted constraints built by steps 2 and 3 using a weighted MAX-SAT solver (Borchers and Furman 1998). Before feeding the weighted constraints to the weighted MAX-SAT solver, we adjust the weights of web constraints by replacing the original weights (denoted as w_o ($0 \leq w_o \leq 1$), which is calculated by the similarity function), with

$$\frac{\gamma}{1-\gamma} \times w_m \times w_o,$$

where w_m, as the scale factor for w_o, is the maximal value of weights of state constraints, and γ is a parameter to vary the importance of web constraints. By varying γ from 0 to 1, we can adjust the weights of web constraints from 0 to ∞. We will show the experiment result of varying γ in the next section. After adjusting the weights, we can solve the all the weighted constraints using the weighted MAX-SAT solver. The solving result is an assignment of all the atoms, e.g., the atom of $p \in$ PRE(a) is assigned as *true*. After that, we will directly convert the solving result to action models $\mathcal{A}_t$ in step 4 of Algorithm 1. For instance, if $p \in$ PRE(a) is *true*, then p will be converted to a precondition of a.

Experiments

We test our learning algorithm `LAWS` in these domains *rovers*[1], *driverlog*[1], *zenotravel*[1], *laundry* and *dishwashing*. *laundry* and *dishwashing* are created based on the description of these two domains in the MIT PLIA1 dataset[2]. We use *rovers*, *zenotravel* and *laundry* as target domains, and *driverlog*, and *dishwashing* as source domains. We collect 75 plan traces from each target domain as training data.

We evaluate our `LAWS` algorithm by comparing its learned action models with the artificial action models which are viewed as the ground truth. We define the error rate of the learning result by calculating the missing and extra predicates of the learned action models. Specifically, for each learned action model a, if a precondition of a does not exist in the ground-truth action model, then the number of errors increases by one; if a precondition of the ground-truth action model does not exist in a's precondition list, then the number of errors also increases by one. As a result, we have the total number of errors of preconditions with respect to a. We define the error rate of preconditions (denoted as $Err_{pre}(a)$) as the proportion of the total number of errors among all the possible preconditions of a, that is,

$$Err_{pre}(a) = \frac{\text{the total number of errors of preconditions}}{\text{all the possible precondition of } a}.$$

Likewise, we can calculate the error rates of adding effects and deleting effects of a, and denote them as $Err_{add}(a)$ and $Err_{del}(a)$ respectively. Furthermore, we define the error rate of all the action models $\mathcal{A}$ (denoted as $Err(\mathcal{A})$) as the

[2]http://architecture.mit.edu/house_n/data/PlaceLab/PLIA1.htm

average of $Err_{pre}(a)$, $Err_{add}(a)$ and $Err_{del}(a)$ for all the actions a in $\mathcal{A}$, that is,

$$Err(\mathcal{A}) = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\frac{1}{3}(Err_{pre}(a) + Err_{add}(a) + Err_{del}(a)),$$

and define the accuracy as $Acc = 1 - Err(\mathcal{A})$.

Comparison among LAWS, t-LAMP and ARMS

We repeat our LAWS five time calculating the average of accuracy. Each time we randomly select one of each five sequential intermediate partial states being observed in plan traces leaving other states empty, and each partial state is selected by 50% of propositions in the corresponding full state (a state is assumed to be represented by a set of propositions). We compare our LAWS algorithm to the previous learning system t-LAMP (Zhuo, Yang, and Li 2009) and ARMS (Yang, Wu, and Jiang 2007), where t-LAMP learns action models by transferring information from source domains via building syntax mappings between the target and source domains; and ARMS learns action models without any information about the source domains. We show the experimental results in Figure 2, where "driverlog $\Rightarrow$ rovers" suggests we learn action models in the domain *rovers* by transferring knowledge from the domain *driverlog*, likewise for "driverlog $\Rightarrow$ zenotravel" and "dishwashing $\Rightarrow$ laundry". The parameter γ, which is introduced in the previous section, is set as 0.5 when running our algorithm LAWS.

From Figure 2, we can see that the accuracy of our algorithm LAWS is higher than the other two, which indicates that exploiting the semantic information from Web performs better than building syntax mappings (as t-LAMP does) and learning without knowledge transfer (as ARMS does). We can also observe that when the number of plan traces is small, the difference between our algorithm LAWS and t-LAMP (or ARMS) is larger. However, when the number of plan traces becomes large, the gap shrinks. This phenomenon indicates that our algorithm LAWS provides better effect on learning the action models when we don't have enough plan traces, since when the number of plan traces becomes larger, there will be more knowledge available from the plan traces themselves, which can be enough to learn the action models. The result also reveals that even when the number of plan traces is very small (e.g., 15), the learning accuracy of our algorithm LAWS will be no less than 75%, which means that exploiting the knowledge from Web searching can really help learning action models.

Ablation Study

Furthermore, we would like to study the interaction between the impact of the knowledge extracted from Web searching in the learning framework and the impact of the knowledge only from the plan traces. We perform this study by calculating the accuracies of the following three cases: (1) the accuracy of the learned action models just from STRIPS semantics constraints, (2) the accuracy of the learned action models with the STRIPS semantics constraints and the web constraints, setting the weights of the web constraints to be the same, (3) the accuracy of the learned action models with the full constraint set, defining their weights by the similarity function, (4) the accuracy of the learned action models with respect to different number of states in plan traces, setting $\gamma = 0.5$ and the percentage of propositions in each state as 50%. The results are shown in Figures 3-4

From Figure 3, we can see that the accuracy of Case III is generally higher of the other cases. This suggests that we can get higher accuracies when we exploit the similarity function, which is consistent with our motivation of measuring the correlation between domains by building the similarity function. Furthermore, the accuracy of Case II is generally better than the one of Case I, which indicates that the web constraints is helpful in improving the learning result. We can also find that the accuracies generally increase when the number of plan traces increases.

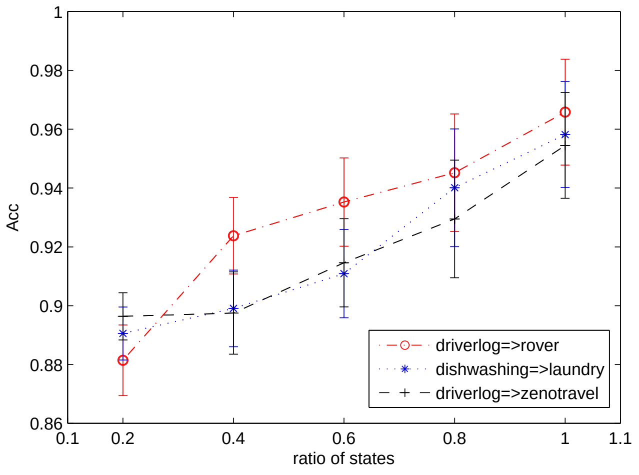

Figure 4: The accuracy with respect to different ratios of states

In order to see the accuracy of the learned action models with respect to different number of states available in plan traces, we test different ratios of states by setting $\gamma = 0.5$. We perform this test with 60 plan traces and 50% as the ratio of propositions in each state of plan traces. From Figure 4, we can see that the accuracy generally increases when the ratio increases. This is consistent with our intuition, since the information increases and could help improve the learning process when the ratio increases.

We varied the value of the parameter γ from 0 to 1 to see the trend of the accuracy, by fixing the number of plan traces to 60. We show the results in Figure 5. From Figure 5, we can see that when γ increases from 0 to 0.5, the accuracy increases, which exhibits that when the effect of the knowledge from Web searching enlarges, the learning accuracy gets higher. However, when γ is larger than 0.5 (note that when $\gamma = 0.5$, $\frac{\gamma}{1-\gamma}$ will be equal to 1, which means the weights of web constraints remain unchanged), the accuracy becomes lower when γ increases. This is because the impact of the knowledge from the plan traces is

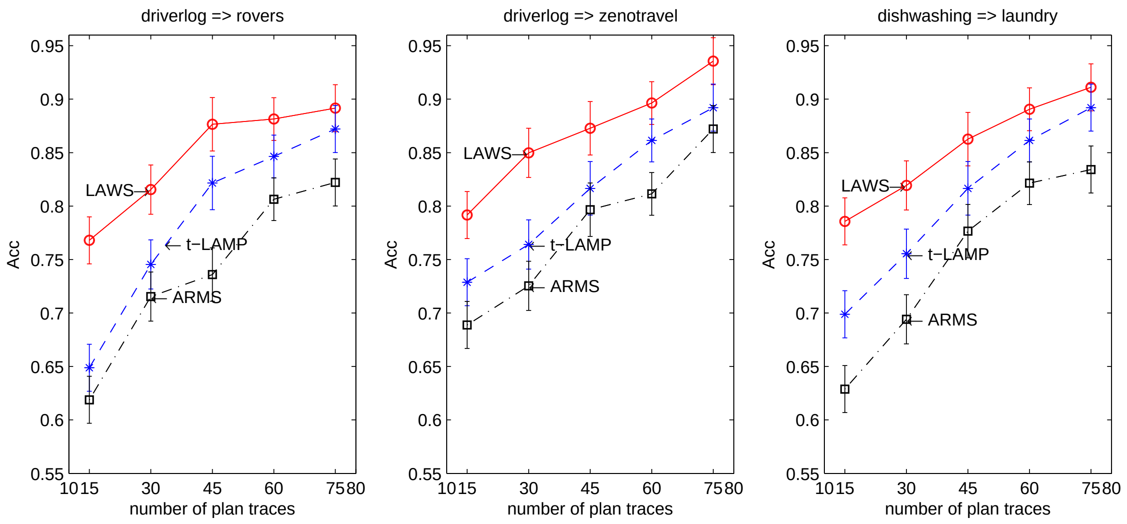

Figure 2: The comparison of LAWS, t-LAMP and ARMS

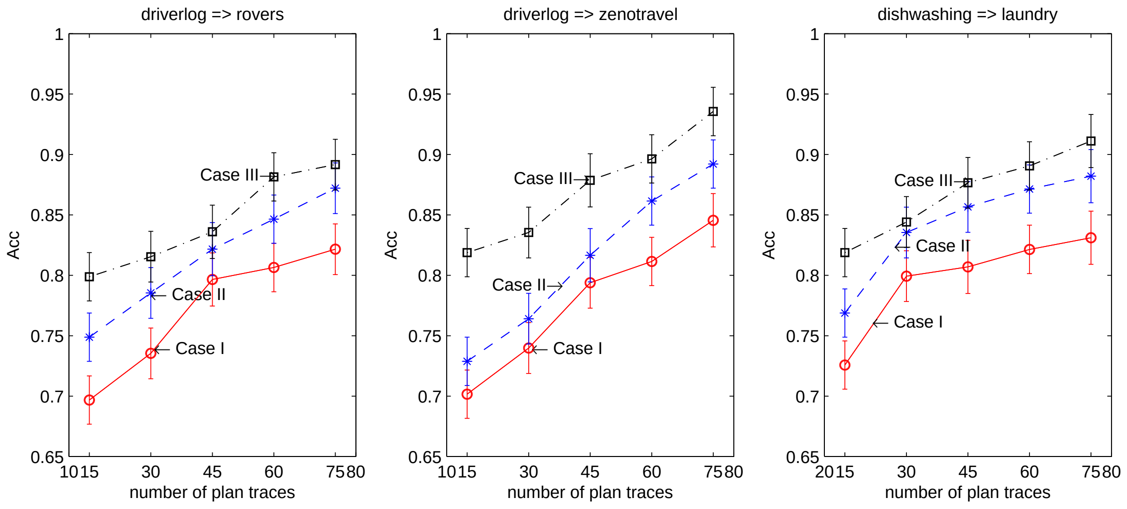

Figure 3: Comparison of different cases (Case I: the accuracy with STRIPS constraints ($\gamma = 0$); Case II: The accuracy with STRIPS & web constraints where weights of web constraints are all the same ($\gamma = 0.5$ and $w_o = 1$); Case III: The accuracy with STRIPS & web constraints where web constraints are defined by the similarity function ($\gamma = 0.5$))

relatively reduced when γ becomes very large, and implies that the knowledge from the plan traces is also important for learning high-quality action models. In summary, with the knowledge of the current limited plan traces, exploiting knowledge from Web searching does help improve the learning accuracy.

We do not have space to show the running time of our algorithm LAWS. Suffice it to say that the CPU time of the learning process is quite reasonable. The maximal time of our LAWS algorithm is smaller than 1,000 seconds for learning action models in our experiment on a typical 2 GHZ PC with 1GB memory. However this time did not include the Web searching time, since it mainly depends on the specific network quality.

Conclusion

In this paper, we proposed a novel cross-domain action-models acquisition algorithm LAWS to learn action models by transferring the knowledge from related source domains to a target domain via Web search. We first built a similarity function to measure the relation between two domains and generate web constraints associated with weights defined by the similarity function. We then built STRIPS semantics constraints from plan traces and solve all the constraints using a weighted MAX-SAT solver. From our experiments, we can see that our LAWS algorithm can learn the action models

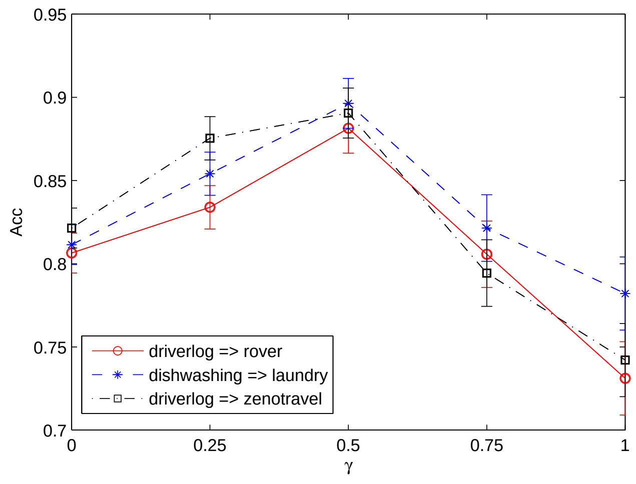

Figure 5: The accuracy with respect to different γ

with the help of the source domains, which is more accurate than other algorithms that do not exploit any knowledge from other domains. We also showed that exploiting knowledge from Web search is helpful in building a mapping function between the domains.

In the future, we would like to study the feasibility of applying our transferring framework to learning more expressive action models, such as PDDL, rather than STRIPS models, and study the feasibility of transferring knowledge from multiple domains.

Acknowledgement

Hankz Hankui Zhuo thanks China Postdoctoral Science Foundation funded project(Grant No.20100480806) and Macao FDCT 013/2010/A for support of this research. Qiang Yang thanks Hong Kong RGC/NSFC grant N HKUST624/09 for supporting this research.

References

Amir, E. 2005. Learning partially observable deterministic action models. In *Proceedings of IJCAI*, 1433–1439.

Blythe, J.; Kim, J.; Ramachandran, S.; and Gil, Y. 2001. An integrated environment for knowledge acquisition. In *Proceedings of IUI*, 13–20.

Bollegala, D.; Matsuo, Y.; and Ishizuka, M. 2009. Measuring the similarity between implicit semantic relations from the web. In *Proceedings of WWW*.

Borchers, B., and Furman, J. 1998. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization* 2(4):299–306.

Borgwardt, K.; Gretton, A.; Rasch, M.; Kriegel, H.; Scholkopf, B.; and Smola, A. 2006. Integrating structured biological data by kernel maximum mean discrepancy. In *Proceedings of ISMB*, 49–57.

Caruana, R. 1997. Multitask learning. *Machine Learning* 28(1):41–75.

Etzioni, O.; Cafarella, M. J.; Downey, D.; Popescu, A. M.; Shaked, T.; Soderland, S.; Weld, D. S.; and Yates, A. 2004. Methods for domain-independent information extraction from the web: An experimental comparison. In *Proceedings of AAAI*, 391–398.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal* 189–208.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Jones, K. S. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28:11–21.

Kullback, S., and Leibler, R. A. 1951. On information and sufficiency. *Annals of Mathematical Statistics* 22(1):79–86.

LI, C. M.; Manya, F.; Mohamedou, N.; and Planes, J. 2009. Exploiting cycle structures in Max-SAT. In *In proceedings of 12th international conference on the Theory and Applications of Satisfiability Testing (SAT-09)*, 467–480.

LI, C. M.; Manya, F.; and Planes, J. 2007. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research* 30:321–359.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPO II: HTN planning in a tool-supported knowledge engineering environment. In *Proceedings of ICAPS*, 92–101.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *In IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)* 22(10):1345–1359.

Perkowitz, M.; Philipose, M.; Fishkin, K.; and Patterson, D. 2004. Mining models of human activities from the web. In *Proceedings of WWW*, 573–582.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 1-2(62):107–136.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence Journal* 171:107–143.

Zettlemoyer, L. S.; Pasula, H. M.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *Proceedings of AAAI*.

Zheng, V. W.; Hu, D. H.; and Yang, Q. 2009. Cross-domain activity recognition. In *Proceedings of UbiComp*, 61–70.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Munoz-Avila, H. 2009. Learning htn method preconditions and action models from partial observations. In *Proceedings of IJCAI*, 1804–1810.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.

Zhuo, H.; Yang, Q.; and Li, L. 2009. Transfer learning action models by measuring the similarity of different domains. In *Proceedings of PAKDD*.