Transferring Knowledge from Another Domain for Learning Action Models

Hankui Zhuo[1], Qiang Yang[2], Derek Hao Hu[2], and Lei Li[1]

[1] Software Research Institute, Sun Yat-sen University, zhuohank@gmail.com
[2] Hong Kong University of Science and Technology, {qyang,derekhh}@cse.ust.hk *

Abstract. Learning action models is an important and difficult task for AI planning, since it is both time-consuming and tedious for a human to encode the action models by hand using a formal language such as PDDL. In this paper, we present a new algorithm to learn action models from plan traces by transferring useful knowledge from another domain whose action models are already known. We call this algorithm $t-$LAMP, (**t**ransfer **L**earning **A**ction **M**odels from **P**lan traces) which can learn action models in PDDL language with quantifiers from plan traces where the intermediate states can contain noise and partial information. We apply Markov Logic Network to enable knowledge transfer, and show that using the transfer learning framework, the quality of the learned action models are generally better than the case when not using an existing domain for transfer.

1 Introduction

Planning systems require action models as input. A typical way to describe action models is to use action languages such as the planning domain description language (PDDL) [6]. A traditional way of building action models is to ask domain experts to analyze a planning domain and write a complete action model representation. However, it is very difficult and time-consuming to build action models in complex real world scenarios in such a way, even for experts. Thus, researchers have explored ways to reduce the human efforts of building action models by learning from observed examples or plan traces. Some researchers have developed methods to learn action models from complete state information before and after an action [1]. Other researchers, such as Yang, Wu and Jiang [2, 3] have developed an approach known as *Action Relation Modeling System* (ARMS) to learn action models in a STRIPS [5] representation using a weighted maximal satisfiability based approach.

In this paper, we present a novel action-model learning algorithm called $t-$LAMP, which stands for ***t**ransfer **L**earning **A**ction **M**odels from **P**lan traces*. In this algorithm, we use the shared common knowledge of one domain to help learn another domain. For instance, we may have a domain *elevator*[3] where action models are already encoded. One example action is *the action 'up(?f1,?f2)'* which means the elevator can go up from

* We thank the support of CERG grant HKUST 621307.

[3] http://www.cs.toronto.edu/aips2000/

a floor 'f1' to a floor 'f2'. This action has a precondition 'lift-at(?f1)' and an effect 'lift-at(?f2)'. Now suppose we wish to learn the logical model of an action 'move(?l1,?2)' in the *briefcase*[1] domain, which means that the case is moved from location 'l1' to location 'l2'. In this new domain we have a precondition 'is-at(?l1)' and an effect 'is-at(?l2)'. Because of the similarity between the two domains, these two actions share the common knowledge on the actions that can cause changes in locations. Thus, to learn one action model, transferring the knowledge from the other action model is likely to be helpful.

2 Problem Definition and Our Algorithm

A classical planning problem can be represented as $\mathcal{P} = (\Sigma, s_0, g)$, where $\Sigma = (S, A, \gamma)$ is the planning domain, s_0 is the initial state, and g is the goal state. In Σ, S is the set of states, A is the set of actions, γ is the deterministic transition function. A solution to a planning problem is called a plan, an action sequence $(a_0, a_1, \ldots, a_n)$. Each a_i is an action schema in the form of 'action name(parameters)' such as 'move(?m - location ?l - location)'. Furthermore, a plan trace is defined as $T = (s_0, a_0, \ldots, s_n, a_n, g)$, where $s_1, ..., s_n$ are intermediate state observations allowed to be partial or empty. Our learning problem can be stated as follows. We are given: (1) a set of plan traces $\mathcal{T}$ in a *target domain* (that is, the domain from which we wish to learn the action models), (2) the description of predicates and action schemas in the target domain, and (3) the completely available action models in a similar *source domain*. As output, $t-$LAMP will output the preconditions and effects of each action schema in our target domain $\mathcal{T}$.

The $t-$LAMP algorithm can be described shortly in four steps. In our first step, we will encode the plan traces into propositional formulae, which is rather standard and interested readers could refer to [9] for technical details. Then in Step 2, we will generate formulae according to some specific correctness constraints and provide the generated formulae as the input of the MLN (denoted as M). In Step 3, we will encode the action model from our *source domain* into another MLN, denoted as M^*, and then transfer from knowledge from M^* to M. After that we can learn the most likely subset of candidate formulae in M. In the last step, we will convert the formulae we learn to the final action models. We will describe Step 2, 3, 4 in detail, where Step 1 is omitted since readers can check it in [9] as we mentioned.

2.1 [Step 2] Generating candidate formulae

In Step 1, plan traces have been encoded as a set of propositional formulae, each of which is a conjunction of propositional variables. Thus plan traces can be represented by a set of propositional variables, whose elements are conjunctions. This set is recorded in a database called DB.

Next, we will generate candidate formulae for individual actions in the following steps from **F1** to **F4**. These candidate formulae attempt to ensure the correctness of action models, that the action models generated are sound. Due to space constraints, we omit the detailed formulae we will add as constraints into MLN in this paper. Nevertheless, we will describe what characteristics our constraints must satisfy so that the correctness and soundness of the action models are ensured.

F1: (*The effect of an action must hold after the execution of this action.*) If a literal p is an effect of some action a, then an instance of p must hold after a is executed.

F2: (*The negative effect of an action must have its corresponding positive counterpart hold before the execution of this action.*) Similar to F1, a literal p's negation is an effect of some action a, which means an instance of p is deleted after a is executed, but it exists before the execution of a.

F3: (*The precondition of an action will be the subset of the state before the execution of this action.*) A formula f (can be a single literal, or with quantifiers) is a precondition of a, which means the instance of f holds before a is executed.

F4: (*A conditional effects holds only when its condition holds before the action.*) A conditional effect, in PDDL form, like "*forall* $\bar{x}$ *(when* $f(\bar{x})$ $q(\bar{x})$*)*", is a conditional effect of some action a, which means for any $\bar{x}$, if $f(\bar{x})$ is satisfied then $q(\bar{x})$ will be added after a is executed.

By **F1**-**F4**, we can generate possible candidate formulae which are used to describe combinations that are possible for describing preconditions and effects based on the soundness requirement of individual actions.

2.2 [Step 3] Transfer learning weights of a MLN

Encoding source-domain action models as a MLN M^*: In this step, we convert all source domain action models into formulae F1 to F4 in order to transfer the source domain knowledge to target domain knowledge. To do this, we convert each action model to the formulae, and then give them the maximum weights. When these formulas are put together with the target domain formulae in the next step, they will influence the learning of action models in the target domain through the mapping function between the two domains. Note that the mapping function can be learned as well.

Transfer learning M from M^*: We find the best way to map M^* into M based on the quality of the mapping. The quality of a mapping is measured by the performance of M on DB, estimated by a weighted pseudo log-likelihood measure (WPLL) score.It sums over the log-likelihood of each node given its Markov blanket, weighting it appropriately to ensure that predicates with many literals do not dominate the result. We do a global mapping to establish a mapping from each predicate in M^* to a predicate in M and then use it to translate the entire M^* to a new MLN $\bar{M}$. The algorithm is shown below. Note that we do not need to require each mapping to be complete.

==========

Transfer Learning M from M^*:
Input: M and M^*
Output: M, whose weights of formulae are initiated
1. Find a mapping from each predicate in M^* to a predicate in M.
2. By the mapping, translate the entire M^* to a new MLN $\bar{M}$.
3. Learning weights of formulae in $\bar{M}$ with DB.
4. Compute the WPLL in this iteration with a previous one. If the new WPLL is better, then $\bar{M}_{best} = \bar{M}$.
5. If all the possible mappings are done, continue; else do a new mapping and goto 2.
6. Assign the weight of each formula in $\bar{M}_{best}$ to the same formula in M, leaving the

weights of other formulae as zero, and output M.
==========
In the first step of the algorithm, a mapping is found by the following process: firstly, for a predicate p^* in M^* and a predicate p in M, we build a *unifier* by mapping their corresponding names and arguments (we require that the number of arguments are the same in p^* and p, otherwise, we find next p to be mapped with p^*); and then substitute all the predicates in M by this unifier; for every p^* and p, we repeat the process of unifier-building and substitution. Next, by the mapping built in the first step, we translate the formulae of M^* to $\bar{M}$'s formulae, whose predicates belong to M. In the third step, The learning process can refer to [4]. The other steps are straightforward. After M is outputted, we can finally learn weights of M as presented in [4, 8].

2.3 [Step 4] Generating action models

The optimization of WPLL indicates that when the number of true grounding of f_i is larger, the corresponding weight of f_i will be higher. Thus, the final weight of a formula in the MLN is a confidence measure of that formula. Intuitively speaking, the larger the weight of a formula is, the more probable that formula will be. However, when generating the final action models from these formulae, we need to determine a threshold, based on the validation set of plan traces and our evaluation criteria (definition of error rate) to choose a set of formulae from MLN.

3 Experiments

3.1 Data Set and Evaluation Criteria

We collect plan traces from *briefcase* and *elevator* domains . Traces are generated by generating plans from the given initial and goal states in these planning domains using the human encoded action models and a planning algorithm, FF planner [4]. These domains have the characteristics we need to evaluate our $t-$LAMP algorithm: they have enough similarities and hence we have the intuition that one can borrow knowledge from the other while learning the action models (as shown in the examples of earlier sections). The initial and goal states we use in our experiments are from planning competition (IPC-2), which we use for generating plan traces as input.

We define error rates of our learning algorithm as the differences between our learned action models and the hand-written action models that are considered as the "ground truth" from IPC-2. If a precondition appears in our learned action models' preconditions but not in hand-written action models' preconditions, the error count of preconditions, denoted by $E(pre)$, increases by one. Similarly, if a precondition appears in hand-written action models' preconditions but not in our learned action models' preconditions, $E(pre)$ increases by one. Likewise, error count of effects are denoted by $E(eff)$. Furthermore, we denote the total number of all the possible preconditions and effects of action models as $T(pre)$ and $T(eff)$, respectively. In our experiments, the error rate of an action model is defined as $R(a) = \frac{1}{2}(E(pre)/T(pre) +$

[4] http://members.deri.at/ joergh/ff.html

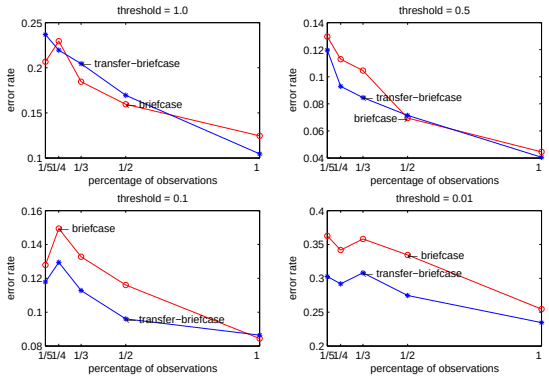

Fig. 1. Learning action models in *briefcase* with or without transferring knowledge from *elevator*. The blue (red) curve shows the result with (without) transfer learning.

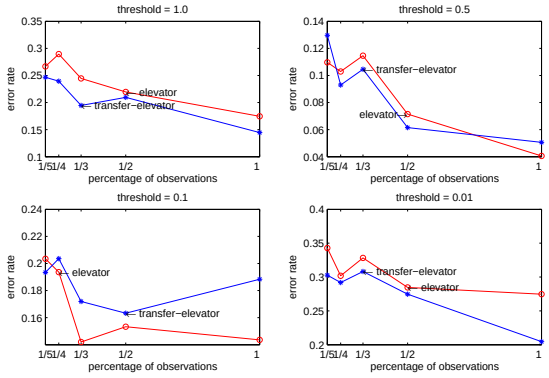

Fig. 2. Learning action models in *elevator* by transferring knowledge from *briefcase*. The blue (red) curve shows the result with (without) transfer learning.

$E(eff)/T(eff)$), where we assume the error rates of preconditions and effects are equally important, and the range of error rate $R(a)$ should be within [0,1]. Furthermore, the error rate of all the action models A is defined as $R(A) = \frac{1}{|A|}\sum_{a\in A} R(a)$, where $|A|$ is the number of A's elements.

3.2 Experimental Results

The evaluation results of t−LAMP in two domains are shown in Figure 1 and 2. Figure 1 shows the result of learning the action models in *briefcase* by transferring the knowledge from *elevator*, while Figure 2 shows the result of learning the action models in *elevator* by transferring the knowledge from *briefcase*. We have chosen different thresholds with weights 1.0, 0.5, 0.1 and 0.01 to test the effect of the threshold on the

performance of learning. The results show that generally the threshold can be neither too large nor too small, but the performance is not very sensitive to the value.

Since our $t-$LAMP algorithm does not require all intermediate states, we can still learn useful information from a partial set of intermediate states. In our experiment, we have chosen the observable percentage of $1/5, 1/4, 1/3, 1/2, 1$ where $1/5$ of observable intermediate states means we will keep only one intermediate state to be observable in every five successive actions. The other percentages $1/4, 1/3, 1/2$ and 1 have similar meanings. Our experiment shows that in most cases, the more states that are observable, the lower the error rate will be, which is consistent with our intuition. However, there are some other cases, e.g. when threshold is set to 1.0 and there are only $1/3$ of states that are observable, the error rate is higher than the case when $1/2$ of states are observable.

From experiments, we can see that transferring useful knowledge from another domain will help improve our action model learning result. On the other hand, determining the similarity of two domains is important, which will be given in our future work.

4 Conclusion

In this paper, we have presented a novel approach to learn action models through transfer learning and a set of observed plan traces. Our $t-$LAMP learning algorithm makes use of *Markov Logic Networks* to learn action models by transferring knowledge from another domain. Our empirical tests in two domains showed that the method is both accurate and effective in learning the action models via knowledge transfer. In the future, we wish to understand better the conditions under which transfer learning is effective in learning the action models, and to extend the learning algorithm to more elaborate action representation languages including resources and functions. We also wish to explore how to make use of other inductive learning algorithms to help us learn better.

References

1. Jim Blythe, Jihie Kim, Surya Ramachandran and Yolanda Gil: An integrated environment for knowledge acquisition. *Intelligent User Interfaces*, 13-20, 2001.
2. Qiang Yang, Kangheng Wu and Yunfei Jiang: Learning Actions Models from Plan Examples with Incomplete Knowledge. *In ICAPS*, 241-250, 2005.
3. Qiang Yang, Kangheng Wu and Yunfei Jiang: Learning action models from plan examples using weighted MAX-SAT. *Artif. Intell*, 171(2-3):107-143, 2007.
4. Matthew Richardson and Pedro Domingos: Markov Logic Networks. *Machine Learning*, 62(1-2):107-136, 2006.
5. Richard Fikes and Nils J. Nilsson: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4):189-208, 1971.
6. Maria Fox and Derek Long: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res. (JAIR)*, 20:61-124, 2003.
7. Lilyana Mihalkova, Tuyen Huynh and Raymond J. Mooney: Mapping and Revising Markov Logic Networks for Transfer Learning. *In AAAI*, 2007.
8. Stanley Kok, Parag Singla, Matthew Richardson and Pedro Domingos: The Alchemy system for statistical relational AI. *University of Washington, Seattle*, 2005.
9. Malik Ghallab, Dana Nau, and Paolo Traverso: Automated Planning: Theory and Practice. Morgann Kaufmann, 2004.