Transfer Learning Action Models by Measuring the Similarity of Different Domains

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

[1] Software Research Institute, Sun Yat-sen University, Guangzhou, China.
zhuohank@gmail.com,lnslilei@mail.sysu.edu.cn
[2] Hong Kong University of Science and Technology, Hong Kong. qyang@cse.ust.hk *

Abstract. AI planning requires action models to be given in advance. However, it is both time consuming and tedious for a human to encode the action models by hand using a formal language such as PDDL, as a result, learning action models is important for AI planning. On the other hand, the data being used to learn action models are often limited in planning domains, which makes the learning task very difficult. In this paper, we present a new algorithm to learn action models from plan traces by transferring useful information from other domains whose action models are already known. We present a method of building a metric to measure the shared information and transfer this information according to this metric. The larger the metric is, the bigger the information is transferred. In the experiment result, we show that our proposed algorithm is effective.

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.

However, previous algorithms and experiments show that action model learning is a difficult task and the performances of the state-of-the-art algorithms are not very satisfying. A useful observation is that in many different planning domains, there exists some useful information that may be "borrowed" from one domain to another, provided that these different domains are similar in some aspects. In particular, we say that two domains A and B are similar if there is a mapping between some predicates of the two domains, in that the underlying principle of these actions, although their corresponding predicates are similar, resemble inherent similarities, then such a mapping can enable us to learn the action model in domain B by the mapping from the learned action model in domain A [9].

In this paper, we present a novel action model learning algorithm called t-LAMP (***t**ransfer **L**earning **A**ction **M**odels other domains*). We use the shared common information from *source* domains to help to learn action models from a *target* domain (we call

* we thank the support of Hong Kong CERG Grant 621307.

the domains whose information is transferred *source* domains, while the domain from which the action models need to be learned a *target* domain). We propose a method of building a metric to measure the "similarity" between two domains, which is a difficult and not being answered question in planning domains. t-LAMP functions in the following three steps. Firstly, we encode the input plan traces into propositional formulas that are recorded as a DB. Secondly, we encode action models as a set of formulas. Finally, we learn weights of all formulas by transferring knowledge from source domains, and generate action models according to the weights of formulas.

The rest of the paper is organized as follows. We first give the definition of our problem and then describe the detailed steps of our algorithm. Then we will discuss some related works. In the experiment section, we will evaluate our algorithm in five planning domains of transfer learning action models and evaluate our transfer learning framework. Finally, we conclude the paper and discuss future work.

2 Related Work

Recently, some researchers have proposed various methods to learn action models from plan traces automatically. Jim, Jihie, Surya, Yolanda [3] and Benson [1] try to learn action models from plan traces with intermediate observations. What they try to learn are STRIPS-models [5, 6]. One limitation of their algorithm is all the intermediate states need to be known. Yang, Wu and Jiang designed an algorithm called ARMS [2], which can learn action models from plan traces with only partial intermediate observations, even without observations.

Another related work is Markov Logic Networks (MLNs)[4]. MLN is a powerful framework that combines probability and first-order logic. A MLN is a set of weighted formulae to soften constraints in first-order logic. The main motivation behind MLN to "soften" constraints is that when a world violates a formula in a knowledge base, it is less probable, but not impossible.

In the transfer learning literature, Lilyana, Tuyen and Raymond[7] address 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. [9] proposes to learn action models by transferring knowledge from another domain, which is the first try to transfer knowledge across domains.

3 Problem Definition

We represent a planning problem 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, which is $S \times A \rightarrow S$. A solution to a planning problem is called a plan, an action sequence $(a_0, a_1, \ldots, a_n)$ which makes a projection from s_0 to g. Each a_i is an *action schema* composed of a name and zero or more parameters. A *plan trace* is defined as $T = (s_0, a_0, s_1, a_1, \ldots, s_n, a_n, g)$, where $s_1, ..., s_n$ are partial intermediate state observations that are allowed to be empty.

We state our learning problem as: given as input (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),

Fig. 1. an example of our problem definition (input and output)

input:	source domains:	Depots, elevator, ⋯
	Target domain: briefcase	**predicates:** (at ?y-portable ?x-location) (in ?x-portable) ... **action schemas:** (move ?m-location ?l-location) **…**
		plan trace 1: (is-at l1) (at o1 l1) (o2 l2), (put-in o1 l1) (move l1 l2) (put-in o2 l2) (move l2 home), (is-at home) (at o1 home) (at o2 home) plan trace 2: ⋯ **…**
output:	(move ?m-location ?l-location) preconditions: (is-at ?m) Effects: (and (is-at ?l) (not (is-at ?m)) (forall (?x-portable) (when (in ?x) (and (at ?x ?l) (not (at ?x ?m)))))	
	… ...	

(2) the description of predicates and action schemas in the target domain, and (3) the completely available action models in *source domains*, Our algorithm t-LAMP outputs preconditions and effects of each action model. An example of the input and output are shown in Fig.1.

4 The Transfer Learning Algorithm

Before giving our algorithm t-LAMP, we present an overview of the algorithm as shown in Fig.2. In the following subsections, we give the detail description of the main steps

Fig. 2. an overview of the t-LAMP algorithm

the *t*-LAMP algorithm:
input: source domain descriptions $\{D_1, D_2, ..., D_n\}$, *plan traces* from the target domain,
action schemas of the target domain D_t.
output: *action model descriptions* of the target domain.

step 1. encode each plan trace as a formula in conjunctive form.
step 2. **for** each source domain D_i, **do**
step 3. encode all the action models of the domain D_i as a list of formulae $F(D_i)$.
step 4. find the best mapping MAP_i between D_i and D_t, the resulting formulae $F^*(D_t)$ and their weights.
step 5. **end**
step 6. generate *candidate formulae* to describe all the possible action models.
step 7. set the initial weights of all the *candidate formulae* as *zero*.
step 8. **for** each candidate formula f_j and its corresponding weight w_j **do**
step 9. **for** each MAP_i, **do**
step 10. if f_j is the same as f_k of the resulting $F^*(D_t)$ of MAP_i, then $w_j=w_j+w_k$.
step 11. **end**
step 12. **end**
step 13. learn weights of all the candidate formulae which are initially weighted by step 7-12.
step 14. select a subset of candidate formulae whose weights are larger than a threshold.
step 15. convert the selected candidate formulae to action models, and return.

which are highlighted.

4.1 Encoding Each Plan Trace as a Proposition Database

As is defined in the problem definition, each plan trace can be briefly stated as an action sequence with observed states, including initial state and goal state. We need to encode states and actions which are also called state transitions. We represent facts that hold in states using propositional formulae, e.g. consider the *briefcase* domain in Fig.1. We have an object *o1* and a location *l1*. We represent the state where the object *o1* is in the briefcase and the briefcase is at location *l1* with the propositional formula: *in(o1)* $\wedge$ *is-at(l1)*, where *in(o1)* and *is-at(l1)* can be viewed as propositional variables. A model of the propositional formula is the one that assigns true value to the propositional variables *in(o1)* and *is-at(l1)*. Every object in a state should be represented by the propositional formula, e.g. if we have one more location *l2*, the above propositional formula should be modified as: *in(o1)*$\wedge$ *is-at(l1)* $\wedge$ $\neg$*is-at(l2)*. The behavior of deterministic actions is described by a transition function γ. For instance, the action *move(l1,l2)* in Fig.1 is described by $\gamma(s_1, move(l1, l2)) = s_2$. In s_1, the briefcase is at location *l1*, while in s_2, it is at *l2*. The states s_1 and s_2 can be represented by: *is-at(l1)* $\wedge$ $\neg$*is-at(l2)* and $\neg$*is-at(l1)* $\wedge$ *is-at(l2)*. We need different propositional variables that hold in different states to specify that a fact holds in one state but does not hold in another state. We introduce a new parameter in predicates, and represent the transition from the state s_1 to the state s_2 by *is-at(l1,* s_1*)* $\wedge$ $\neg$*is-at(l2,* s_1*)* $\wedge$ $\neg$*is-at(l1,* s_2*)* $\wedge$ *is-at(l2,* s_2*)*. On the other hand, the fact that the action *move(l1, l2)* causes the transition can be represented by a propositional variable *move(l1, l2,* s_1*)*. Thus, the function $\gamma(s_1, move(l1, l2))$ can be represented as $move(l1, l2, s_1) \wedge$is-at$(l1, s_1) \wedge \neg$is-at$(l2, s_1) \wedge \neg$is-at$(l1, s_2) \wedge$is-at$(l2, s_2)$. As a result, a plan trace can be encoded correspondingly.

Thus, plan traces can be encoded as a set of propositional formulae, each of which is a conjunction of propositional variables. As a result, each plan trace can be represented by a set of propositional variables, whose elements are conjunctive. This set is recorded in a database called DB, i.e. each plan trace is corresponded to its own DB.

4.2 Encoding Action Models as Formulae

We consider an action model is a strips model plus conditional effects, i.e. a precondition of an action model is a positive atom, and an effect is either a positive/negative atom or a conditional effect. According to the semantic of an action model, we equally encode an action model with a list of formulae, as addressed in the following.
T1: If an atom p is a positive effect of an action a, then p must hold after a is executed. The idea can be formulated by: $\forall i.a(i) \rightarrow \neg p(i) \wedge p(i{+}1)$, where i corresponds to s_i.
T2: Similar to T1, the negation of an atom p is an effect of some action a, which means p will *never hold* (be deleted) after a is executed, which can be formulated by: $\forall i.a(i) \rightarrow \neg p(i{+}1) \wedge p(i)$.
T3: If an atom p is a precondition of a, then p should hold before a is executed. That is to say, the following formula should hold: $\forall i.a(i) \rightarrow p(i)$.
T4: A positive 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 hold after a is executed. Here, $f(\bar{x})$ is a formula in the conjunctive form of atoms. Thus a conditional effect can be encoded by: $\forall i.\bar{x}.a(\bar{x}, i) \wedge f(\bar{x}, i) \rightarrow q(\bar{x}, i{+}1)$.

Fig. 3. the algorithm to learn weights and the corresponding score WPLL

the algorithm to learn weights w and the corresponding score *WPLL*:
input: a list of *DB*s, a list of formulae $F^*(D_t)$.
output: a list of weights w for the formulae $F^*(D_t)$, and *WPLL*.

step 1. initiate $w^0 = (0, \ldots, 0)$.
step 2. $i = 0$.
step 3. **repeat**
step 4. calculate $WPLL(w^i)$ using *DB*s and $F^*(D_t)$.
step 5. $w^{i+1} = w^i + \lambda * \partial WPLL(w^i)/\partial w^i$, where λ is a mall enough constant.
step 6. $i = i+1$;
step 7. **until** i is larger than a maximal iterative number.
step 8. output w^i and $WPLL(w^i)$.

T5: Similarly, a negative conditional effect of the form like "*forall* $\bar{x}$ *(when* $f(\bar{x})$ $\neg q(\bar{x})$*)*", can be encoded by: $\forall i.\bar{x}.a(\bar{x}, i) \wedge f(\bar{x}, i) \rightarrow \neg q(\bar{x}, i{+}1)$.
By T1-T5, we can encode an action model by requiring its corresponding formulas to be always true. Furthermore, for each source domain D_i, we can encode the action models in D_i with a list of formulae $F(D_i)$.

4.3 Building the Best Mapping

In step 4, we find the best mapping between the source domain and the target domain, to bridge these two domains. To map two domains, firstly, we need to map the predicates between the source domain D_i and the target domain D_t; secondly, map the action schemas between D_i and D_t. The mapping process of these two steps is the same, which is: *for each predicate p_i in D_i and a predicate p_t in D_t, we build a **unifier** by mapping their corresponding names and arguments (we require that the number of arguments are the same in p_i and p_t, otherwise, we find next p_t to be mapped with p_i); and then substitute all the predicates in D_t by this unifier; for each p_i and p_t, we repeat the process of **unifier-building** and **substitution** until the unifier-building process stops.*

By applying a mapping to the list of formulae $F(D_i)$, we can generate a new list of formulae $F^*(D_t)$, which encodes action models of D_t. We manage to calculate a score function on $F^*(D_t)$ to measure the similarity between D_i and D_t. We exploit the idea of [4, 8] to calculate the score WPLL (which will be defined soon) when learning weights of formulae. The calculate process is given in Fig.3 In the highlighted step (step 4) of Fig.3, WPLL, the Weighted Pseudo-Log-Likelihood [4], is defined as $WPLL(w) = \sum_{l=1}^{n} \log P_w(X_l = x_l | MB_x(X_l))$ where, $P_w(X_l = x_l | MB_x(X_l)) = \frac{C_{(X_l = x_l)}}{C_{(X_l=0)} + C_{(X_l=1)}}$ and $C_{(X_l = x_l)} = \exp \sum_{f_i \in F_l} w_i f_i(X_l = x_l, MB_x(X_l))$. x is a possible world (a database DB). n is the number of all the possible *groundings* of atoms appearing in all the formulae $F^*(D_t)$, and X_l is the lth *groundings* of the all. $MB_x(X_l)$ is the state of the Markov blanket of X_l in x. The more detail description is presented by [4].

Using the algorithm, we will attain one score WPLL for each mapping. We keep the mapping (which is mentioned as the best mapping) with the highest score WPLL, the resulting $F^*(D_t)$ and their weights.

4.4 Generating Candidate Formulae and Action models

In steps 6 and 7, using the predicates and action schemas from D_t, we will generate all the possible action models by doing a combination between them. We initially associate each candidate formulae with a weight of zero to indicate that no contribution is provided initially.

From the definition of WPLL, we can see that the larger the WPLL is, the more probable the formulae $F^*(D_t)$ are satisfied by DBs, i.e. the more similar the source domain and the target domain (from which DBs are attained) are. Thus, we use WPLL to measure the similarity between source/target domains, and the weights of the resulting formulae $F^*(D_t)$ to transfer information of the "similarity". We exploit the idea that the "similarity" information is strengthened (weakened) when other domains strengthen (weaken) it, by simply adding up the weights "$w_j = w_j + w_k$" in step 10. With the weights attained by steps 7-12, in step 13 we learn weights of the candidate formulas by the algorithm of Fig.3.

From the learning process of WPLL, we can see that 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. In other words, the larger the weight of a candidate formula is, the more likely to be true that formula will be. When generating the final action models from these formulae in step 14, 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 converted to action models in step 15.

5 Experiments

5.1 Data Set and Evaluation Criteria

We collect plan traces from the following planning domains: *briefcase*[3], *elevator*[4], *depots*[5], *driverlog*[3], the plan traces numbers of which are 150, 150, 200 and 200 respectively. These plan 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[6]. Each of the domains will be used as the target domain in our experiment. The source domains are: *briefcase, elevator, depots, driverlog, zenotravel*[3].

We define error rates of our learning algorithm as the difference between our learned action models and the hand-written action models that are considered as the "ground truth". If a precondition appears in the preconditions of our learned action models but not in the ones of hand-written action models, the error count of preconditions, denoted by $E(pre)$, increases by one. If a precondition appears in hand-written action models but not in our learned action models, $E(pre)$ increases by one. Likewise, error count of effects are denoted by $E(eff)$. Furthermore, we denote the total

[3] http://www.informatik.uni-freiburg.de/ koehler/ipp.html
[4] http://www.cs.toronto.edu/aips2000/
[5] http://planning.cis.strath.ac.uk/competition/
[6] http://members.deri.at/ joergh/ff.html

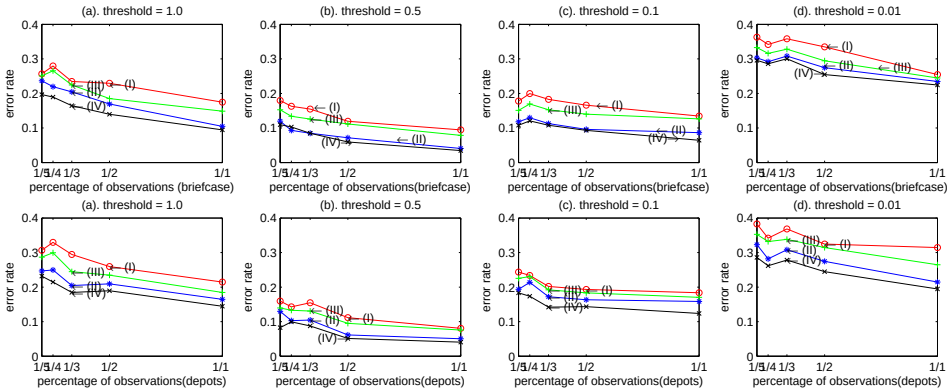

Fig. 4. Accuracy with different thresholds and percentage of observable intermediate states for learning action models of *briefcase* and *depots*

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 $R(a) = \frac{1}{2}(E(pre)/T(pre) + E(eff)/T(eff))$, 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.

5.2 Experimental Results

The evaluation results of t-LAMP in two domains are shown in Fig.4. The red curve (I) is the learning result without transferring any information from other domains; the blue curve (II) is the learning result with transferring information from the most similar domain based on WPLL; the green curve (III) is the result with transferring information from the least similar domain based on WPLL; the black curve (IV) is the result with transferring information from all the other source domains (when learning action models of *briefcase*, the source domains are *elevator, depots, driverlog, zenotravel*). From these two figures, we can see that, the result by transferring information from all the other source domains is the best. Furthermore, by comparing the results of (II) and (III), we can see that, when we choose the most similar domain for transferring, the result is generally better than choosing the least similar domain, i.e. the score function WPLL works well in measuring the similarity of two domains.

The first row of Fig.4 shows the result of learning the action models of *briefcase* with transferring the information from *depots, driverlog, zenotravel, elevator*, while the second row shows the result of learning the action models of *depots* with transferring the information from *briefcase, driverlog, zenotravel, elevator*. 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 choice of

the value. An intuitive explanation is that, a threshold that is too large may lose useful candidate formulae, and a threshold that is too small may contain too many noisy candidate formulae that will affect the overall accuracy of the algorithm. This intuition has been verified by our experiment. In our experiment, it can be shown that when we set the threshold as 0.5, the mean average accuracy is the best.

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 0.01, when there are only $1/4$ of states that are observable, the error rate is lower than the case when $1/3$ of states are observable.

From our experiment results, 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.

6 Conclusion

In this paper, we have presented a novel approach to learn action models through transfer learning and a set of observed plan traces. we propose a method to measure the similarity between domains and make use of the idea of *Markov Logic Networks* to learn action models by transferring information from other domains according to "similarity". Our empirical tests show that our method is both accurate and effective in learning the action models via information transfer. In the future, we wish 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. *IUI*, 13-20, 2001.
2. 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.
3. Scott Benson: Inductive Learning of Reactive Action Models. *In ICML*, 47-54, 1995.
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. Hankui Zhuo, Qiang Yang, Derek Hao Hu and Lei Li: Transferring Knowledge from Another Domain for Learning Action Models. *In PRICAI*, 2008.