Artificial Intelligence 212 (2014) 80–103

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Action-model acquisition for planning via transfer learning

Hankz Hankui Zhuo [a], Qiang Yang [b,*]

[a] School of Advanced Computing, Sun Yat-sen University, Guangzhou, China
[b] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 4 January 2013
Received in revised form 9 March 2014
Accepted 19 March 2014
Available online 25 March 2014

Keywords:
AI planning
Action model learning
Transfer learning
Markov logic networks
Web search

ABSTRACT

Applying learning techniques to acquire action models is an area of intense research interest. Most previous work in this area has assumed that there is a significant amount of training data available in a planning domain of interest. However, it is often difficult to acquire sufficient training data to ensure the learnt action models are of high quality. In this paper, we seek to explore a novel algorithm framework, called TRAMP, to learn action models with limited training data in a target domain, via transferring as much of the available information from other domains (called source domains) as possible to help the learning task, assuming action models in source domains can be transferred to the target domain. TRAMP transfers knowledge from source domains by first building structure mappings between source and target domains, and then exploiting extra knowledge from Web search to bridge and transfer knowledge from sources. Specifically, TRAMP first encodes training data with a set of propositions, and formulates the transferred knowledge as a set of weighted formulas. After that it learns action models for the target domain to best explain the set of propositions and the transferred knowledge. We empirically evaluate TRAMP in different settings to see their advantages and disadvantages in six planning domains, including four International Planning Competition (IPC) domains and two synthetic domains.

1. Introduction

AI planning techniques [21] 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 lots of interest from researchers in the past. For instance, McCluskey et al. [5,41] designed a system to interact with a human expert to generate action models. Amir [1] 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. [67] presented a framework for automatically discovering STRIPS [20] action models from a set of successfully observed plans; Zhuo et al. [71] proposed to learn method preconditions and action models simultaneously for HTN planning, assuming that a set of hierarchical structures was given; Xu and Laird [65] proposed to simultaneously learn discrete and continuous action model in Soar cognitive architecture; 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. However, in many real-world applications, collecting a large amount of training examples for learning high-quality action models is often both difficult and costly. For example, in military operation, it is both difficult and expensive to collect a large number of plan traces (i.e., training data),

* Corresponding author.
E-mail addresses: zhuohank@mail.sysu.edu.cn (H.H. Zhuo), qyang@cse.ust.hk (Q. Yang).

http://dx.doi.org/10.1016/j.artint.2014.03.004
0004-3702/

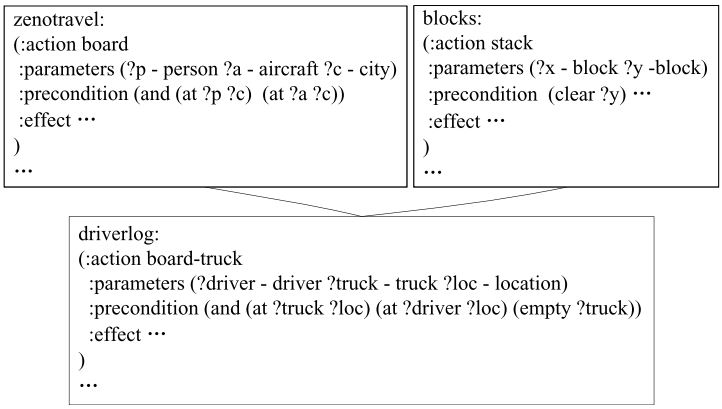

Fig. 1. Transfer knowledge from *board* and *stack* for learning *board-truck*.

since there is only a small amount of plan examples that can be collected in the real domain (e.g., a plan example may correspond to a war that does not happen frequently), and plan examples themselves may take lots of resources, such as ammunition. It is a difficult learning task when there is only a limited plan traces available [47].

By observation, we find that there is some knowledge available in other domains, which could be helpful for the learning. One motivating example for this problem can be illustrated in the domains *blocks*,[1] *zenotravel*[2] and *driverlog*[2], as shown in Fig. 1. Suppose that we wish to learn the logical models of the action "board-truck" in *driverlog* domain, which means that the driver "?driver" enters a truck "?truck" in a location "?loc". To execute this action, the current location of "?driver" and "?truck" should be in "?loc", indicating that the preconditions of "(at ?truck ?loc)" and "(at ?driver ?loc)" should be satisfied. This logical relationship is similar to the action of "board" from the *zenotravel* domain, which shows that if a person "?p" wants to enter an aircraft "?a" in a city "?c", the preconditions of "(at ?p ?c)" and "(at ?a ?c)" should be satisfied. Similarly, in the action "board-truck", if "?driver" wants to enter "?truck", "?truck" should be empty, which means the precondition of "(empty ?truck)" should be also satisfied. This relationship is similar to the action of "stack" from the domain *blocks*, which requires that the block "?y" should be clear (i.e. the precondition "(clear ?y)" is satisfied) if the block "?x" needs to be stacked on "?y". Thus, to learn the model of the action "board-truck", it is likely to be helpful to "borrow" the knowledge from the domains *zenotravel* and *blocks*.

Other motivation examples can also be found from many real world applications, such as *Coal mining* and *Urban search and rescue*. In the Coal mining domain, the goal of coal mining is to obtain coal from the ground. Coal mining has had a lot of developments over the recent years, from the early days of men tunneling, digging and manually extracting the coal on carts to large open cut and long wall mines. Mining at this scale requires the use of draglines, trucks, conveyor, jacks and shearers. Urban search and rescue (i.e., USAR) involves the location, extrication, and initial medical stabilization of victims trapped in confined spaces due to natural disasters, structural collapse, transportation accidents, and collapsed trenches. USAR services can be faced with complex rescue operations within hazardous environment. These two domains share some common knowledge that can be used to help each other. For example, in Fig. 2(a), the action *scoop-up* in the Coal mining domain, i.e., a robot scoops up coal using a coal shovel, is similar to the action *take-up* in the USAR domain (as is shown in Fig. 2(b)), i.e., a robot takes up a victim from hazardous environment. They both suggest that a robot is changing the location of some object (coal or victim). It would be greatly beneficial if we can transfer the knowledge from one domain where we have lots of knowledge on actions to another, where we have less domain knowledge, sine collecting a large number of plan traces from these domains is costly.

We propose a novel transfer learning algorithm framework to leverage knowledge from source domains, called TRAMP, which stands for **TR**ansfer learning **A**ction **M**odels for **P**lanning (the early version is presented in [70,73]). TRAMP seeks to learn action models in the target domain by transferring knowledge from source domains, assuming action models in source domains are already created by experts and can be transferred to the target domain. To transfer knowledge from source domains, TRAMP exploits two approaches to bridging source and target domains. The first approach is to autonomously map predicates and actions in source domains to the target domain via testing whether the mapped predicates and actions can best explain the plan traces in the target domain (the original idea was presented by our previous work [70]). Previous work [19] has demonstrated the success of structure-mapping in both theoretical and experimental aspects. The second approach is to bridge source and target domains via searching keywords of predicates and actions from the Web and calculating similarities of Web pages [10] (the original idea was presented by our previous work [73]). TRAMP is built based on Markov Logic Network (MLN) [53]. Specifically, TRAMP takes as inputs a set of action models from the source domains and a set of plan traces from the *target* domain, and outputs action models for the *target* domain (in this paper we focus

[1] http://www.cs.toronto.edu/aips2000/.
[2] http://planning.cis.strath.ac.uk/competition/.

(a). **Scoop up coal** in the coal mining domain

(b). **Take up victims** in the urban search and rescue domain

Fig. 2. A motivation example of TRAMP (note that Fig. 2(a) is from "http://www.clipartillustration.com/clipart-illustration-rescue-robot-mine-sweeper-or-miner/", and (b) is from "http://www.army.mil/article/48456/robots-to-rescue-wounded-on-battlefield/").

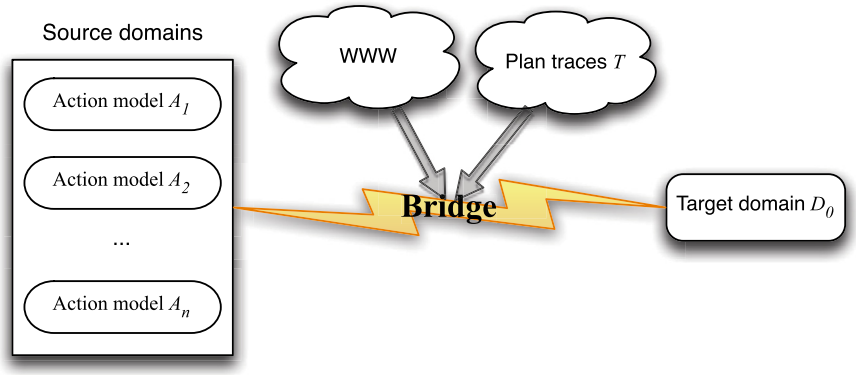

Fig. 3. Bridging source and target domains based on Web and plan traces $\mathcal{T}$.

on the STRIPS action models). TRAMP functions in the following four main steps. First, TRAMP encodes the inputted plan traces, including observed states and actions that are viewed as state transitions, into propositional formulas, and records them as databases DBs. Second, TRAMP encodes the action models from the source domains to a set of formulas. Third, TRAMP generates a set of *candidate formulas* to formulate the target action models and builds mappings between source and target domains. Finally, TRAMP learns action models from the transferred knowledge with the help of Markov Logic Networks.

In TRAMP, we assume that when we search keywords from the Web, search engines are capable of providing Web pages describing the semantic information related to keywords [69,64]. We employ the assumption that Web documents we sample are related to keywords anytime when we do Web search; the higher the rank of documents are, the more they are relevant to the keywords. We can thus sample documents based on the rank of documents to measure the similarity between keywords, as we did in the experiment. Intuitively, we show the framework of TRAMP in bridging source and target domains based on the Web and the plan traces $\mathcal{T}$ in Fig. 3. TRAMP takes as inputs sets of action models from multiple source domains $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, bridges source and target domains based the Web and the plan traces, and outputs action models $\mathcal{A}$ for the target domain. Note that our *knowledge transfer* framework is both different from transfer learning (for classification, regression and clustering problems) in data mining area, which aims to transfer *instances*, *feature spaces*, or *model parameters* [47], and different from *transfer learning* in reinforcement learning, which aims to transfer previous experience gained in learning to perform one task, to a related but different task to improve its learning performance [57]. Our transfer framework aims to transfer *second order relations* between domains, where a *second order relation* suggests the relation between two relations (e.g., if $pre(r_1, r_2)$ indicates relation r_1 is a precondition of relation r_2, *pre* is a *second order relation*), and a *relation* can be seen as a predicate with zero or more parameters, e.g., *(scoop-up robot1 coal1)*, where *scoop-up* is a predicate, and *robot1* and *coal1* are two parameters. Compared to previous transfer learning in data mining area (which can be viewed as transferring relations, rather than second order relations), our transfer learning task aims to transfer second order relations, which is much more difficult, since transferring relations is only a subtask of transferring second order relations and the number of second order relations is often very large.

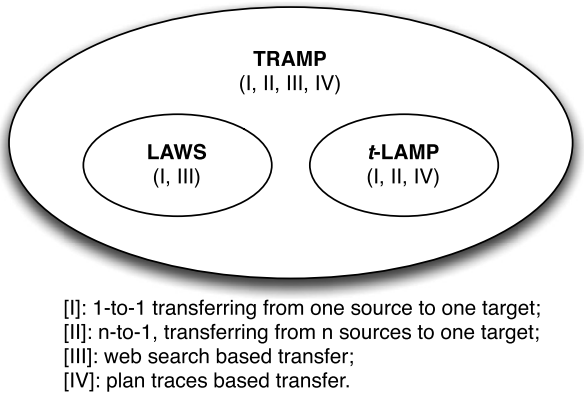

Fig. 4. The relation among TRAMP [this paper], LAWS [73] and t-LAMP [70].

In summary, we have the following contributions in this paper:

- Compared to our previous transfer learning approaches LAWS [73] and t-LAMP [70], TRAMP is a general framework which spans both LAWS and t-LAMP. The relation among TRAMP, LAWS and t-LAMP can be seen from Fig. 4. In LAWS, we explored the knowledge available from the Web to bridge source and target domains, which can transfer from one source domain to one target domain; in t-LAMP, we described an early version to transfer knowledge from multiple source domains by building mappings between source and target domains with the help of plan traces from the target domain. In this paper, we extend both LAWS and t-LAMP with more details, and present a general transfer learning framework TRAMP. We add more experiment results to evaluate TRAMP in different aspects.
- We explore a novel transfer learning framework to learn action models in the planning community under the condition that there are only a few plan traces (i.e., training data) available in the target domain. We explore a novel transfer learning approach which is composed of two phases. The first phase is to transfer knowledge from source domains based on plan traces from the target domain. Specifically, this phase automatically builds structure mappings between source and target domains that can best explain the plan traces from the target domain, and transfer knowledge to the target domain based on the built mappings. The second is to bridge source and target domains using Web search, and transfer knowledge to target domains based on the searched Web pages.
- In order to capture the transferability between source and target domains, we developed a score function to measure the similarity between source and target domains, which can be exploited to determine which source domains are transferable for the target domain. We demonstrated the effectiveness of the score function on determining the transferability of source and target domains in the experiment (as described in Section 4.2.5).
- In the experiment we compared TRAMP to LAWS [73] and t-LAMP [70], to see the advantage or disadvantage of transferring knowledge based on plan traces or the Web or both. We also compared TRAMP to a state-of-the-art non-transfer learning approach ARMS, showing that TRAMP functions well even though there is no source domain available.

In the following sections, we organize the paper as follows. We first give the definition of our learning problem and present our transfer learning framework TRAMP in the next two sections. After that, we empirically evaluate TRAMP in different settings to see their advantages and disadvantages. Finally, we review previous work related to this paper, and conclude the paper together with the future work.

2. Problem formulation

We consider the STRIPS model in this paper to illustrate our idea, leaving more elaborate PDDL models to future work. A planning domain is defined in this paper as $\Sigma = (S, A, \gamma)$, where S is the set of states, A is the set of action models, γ is the deterministic transition function $S \times \mathcal{A} \to S$. Each action model in $\mathcal{A}$ is composed of three parts: an action name with zero or more arguments, a set of preconditions which should be satisfied before the action is executed, and a set of effects which are the results of executing the action. A planning problem can be defined as $\mathcal{P} = (\Sigma, s_0, g)$, where s_0 is an initial state, and g is a goal state. A solution to a planning problem is an action sequence $(a_0, a_1, \ldots, a_n)$ called a plan, which makes a projection from s_0 to g.

We define an *action schema* as an action name with zero or more arguments, e.g., (*board-truck ?d – driver ?t – truck ?loc – location*). Furthermore, a *plan trace* is defined as $T = (s_0, a_0, s_1, a_1, \ldots, s_n, a_n, g)$, where $s_1, \ldots, s_n$ are *partially* observed intermediate states. A state is called "partial" because some propositions in the state are missing. For example, a partial state in the *driverlog* domain is composed of a set of propositions {*(at t1 l1)*, *(at d1 l1)*, *(empty t1)*, *(link l1 l2)*}, where the parts in gray indicate they are missing. "partial" suggests "empty", when all propositions in a state are missing. We assume that the action execution leaves a trace (or plan trace) which is recorded in a data file.

input:	source domains:	zenotravel, blocks, ...
	Target domain: driverlog	**predicates:** (at ?obj - locatable ?loc - location) (empty ?t - truck) ... **action schemas:** (board-truck ?d – driver ?t - truck ?loc - location) **…**
		plan trace 1: (at t1 l1) (at d1 l1) (empty t1)(link l1 l2), (board-truck d1 t1 l1) (drive-truck t1 l1 l2 d1), (at t1 l2) (driving d1 t1) **plan trace 2:** ⋯ **…**
output:	board-truck(?d – driver ?t - truck ?loc - location) preconditions: (and (at ?truck ?loc) (at ?driver ?loc) (empty ?truck)) effects: (and (not (at ?driver ?loc)) (driving ?driver ?truck) (not (empty ?truck))))	
	… ...	

Fig. 5. An input/output example of the TRAMP algorithm.

Our learning problem is stated as follows. We are given as input: (1) the completely known action models $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ from source domains $D_1, \ldots, D_n$; (2) a set of action schemas A and a set of predicates P from the *target domain*; and (3) a set of plan traces $\mathcal{T}$ with partial intermediate states collected from the target domain D_0; these traces will help decide which actions from the other domains are the correct ones to be transferred. Our TRAMP algorithm will output a set of action models, including preconditions and effects of all the actions in the target domain. Our aim is to transfer a high-quality action model set for the target domain that 'fits' the observed plan traces as well as possible, to do this efficiently and to keep the required plan traces in the target domain at a minimal. An example of input and output can be found in Fig. 5.

3. The transfer learning approach: TRAMP

In this section, we present our transfer learning approach TRAMP based on Markov Logic Networks. We first give an overview of TRAMP as shown in Algorithm 1. The detailed description for the main steps will be given in Sections 3.1–3.4.

Algorithm 1 Algorithm overview of TRAMP.

Input: (1) sets of action models from domain $D_1, \ldots, D_n$: $\mathcal{A}_1, \ldots, \mathcal{A}_n$; (2) a set of plan traces $\mathcal{T}$ from target domain D_0.
Output: action models $\mathcal{A}$ for D_0.

1: set a threshold δ;
2: encode each plan trace as a proposition set (or database) DB;
3: $F_0 = all\_possible\_formulas(D_0)$;
4: transfer knowledge from source domains:
  (4a) transferring based on plan traces:
    w = plan_traces_based_transfer(F_0, $\mathcal{A}_1, \ldots, \mathcal{A}_n$, DBs);
  (4b) transferring based on Web search:
    w = web_search_based_transfer(w, F_0, $\mathcal{A}_1, \ldots, \mathcal{A}_n$);
5: $w^* = learnweights(w, F_0, \text{DBs})$;
6: select formulas from F_0 with weights larger than threshold δ, and convert these formulas to action models $\mathcal{A}$;
7: **return** $\mathcal{A}$;

3.1. [Step 2:] Encoding plan traces

We aim to encode the input plan traces as a set of "big conjunctions" in a way similar to situation calculus [38]. A big conjunction is stored in a database called DB which is one of the input of Markov Logic Networks. Plan traces are the "training data" one can collect in a target domain. As defined above, each plan trace can be briefly stated as an action sequence with partially observed states, including initial state and goal state. Thus, to encode a plan trace, we need to encode both the partial states and the actions that serve state transitions.

Encode states: We represent the facts that hold in states as propositional formulas; e.g., consider the *driverlog* domain in Fig. 5. We have a truck *t1*, a driver *d1*, and two locations *l1* and *l2*. We represent the state where the truck *t1* and the driver *d1* are both at *l1*, and *t1* is empty, with the propositional formula:

$$(at\ t1\ l1) \wedge (at\ d1\ l1) \wedge (empty\ t1),$$

where the items such as *(at t1 l1)* can be viewed as propositional variables. A model of the propositional formula assigns truth value to these propositional variables.

Encode state transitions: The behavior of deterministic actions is described by a transition function γ. For example, the action *(board-truck d1 t1 l1)* in Fig. 5 is described by $\gamma(s_1, (\textit{board-truck d1 t1 l1})) = s_2$. In s_1, the driver is at location *l1*; and while in s_2, he is in the truck *t1* and driving *t1*. The states s_1 and s_2 can be represented by:

$$(\textit{at d1 l1}) \wedge \neg(\textit{driving d1 t1})$$

and

$$\neg(\textit{at d1 l1}) \wedge (\textit{driving d1 t1}).$$

These formulas, however, cannot be used to represent the fact that the system evolves from state s_1 to state s_2. Therefore, 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. This formalism is about transferring models from source domains and learning action models in the target domain. We want to reason about different situations/context in which certain statements are true. That way we can differentiate those context and learn models. To do this, we introduce a new parameter in predicates, similar to situation calculus [38], so that the transition from the state s_1 to the state s_2 can be represented by

$$(\textit{at d1 l1 } s_1) \wedge \neg(\textit{driving d1 t1 } s_1) \wedge \neg(\textit{at d1 l1 } s_2) \wedge (\textit{driving d1 t1 } s_2). \tag{1}$$

Conversely, the fact that the action (*board-truck d1 t1 l1*) causes the transition can be represented by a propositional variable (*board-truck d1 t1 l1* s_1). Thus, the function $\gamma(s_1, (\textit{board-truck d1 t1 l1}))$ can be represented as

$$(\textit{board-truck d1 t1 l1 } s_1) \wedge (\textit{at d1 l1 } s_1) \wedge \neg(\textit{driving d1 t1 } s_1) \wedge \neg(\textit{at d1 l1 } s_2) \wedge (\textit{driving d1 t1 } s_2).$$

In this way, a plan trace can be fully encoded. Notice that, for simplicity, we only consider one proposition (*at d1 t1*) in s_1, although there are other propositions such as (*empty t1*) that can be considered.

So far, we have encoded the plan traces as a set of propositional formulas, each of which is a conjunction of propositional variables. These propositional variables are recorded in a proposition database called DB, i.e., Step 2 of Algorithm 1, and there is one DB for each plan trace. In the subsequent sections, we denote all the plan traces as DBs.

3.2. [Step 3:] Encoding action models

Consider that an action model is in the form of STRIPS, i.e. a precondition of an action model is a positive literal, and an effect is either a positive/negative literal. According to the semantics of an action model, we equally encode an action model with a list of formulas, as addressed in the following.

F1: ***(positive effects)*** If an atom p is a positive effect of an action a, then p must hold after a is executed. Considering the consistent requirement in real applications, we require that an action cannot add an existent atom, i.e. p should not exist before 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 the state s_i. $a(i)$ with *true* value means that the action a is executed in the state s_i. $p(i+1)$ with *true* value means that p holds in the state s_{i+1}.

F2: ***(negative effects)*** Similar to **F1**, 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. Also, we require that an action cannot delete a nonexisting atom, i.e. p exists before a is executed. The idea can be formulated by:

$$\forall i. a(i) \rightarrow \neg p(i+1) \wedge p(i).$$

Notice that this constraint is placed to ensure the learned action models are succinct, and most existing planning domains satisfy them. Nevertheless, our algorithm works perfectly without it, i.e., using the formula $\forall i. a(i) \rightarrow \neg p(i+1)$ instead.

F3: ***(preconditions)*** If an atom p is a precondition of a, then p should hold before a is executed. That is, the following formula should hold:

$$\forall i. a(i) \rightarrow p(i).$$

By **F1–F3** and predicates and action schemas from D_0, we generate all the possible action models via their combinations, which is what procedure *all_possible_formulas* do in Step 3 of Algorithm 1. For instance, in Fig. 5, the predicate *at* can be a precondition of *board-truck* and *drive-truck*, or a positive/negative effect of *board-truck* and *drive-truck*. Thus, we need to combine all the actions and predicates by considering that a predicate could be a precondition, a positive or negative effect of an action. We exploit a *parameter constraint*, which requires that the parameters in a precondition or an effect appear in the parameters of the action, to reduce the size of the combinations. This constraint is consistent with the definition of a STRIPS model.

After all the possible action models are generated, we will further generate a set of formulas, called *candidate formulas* and denoted as F_0, to encode these action models using **F1–F3**. The process is the same as step 3. Finally, we initialize the learning process by associating each candidate formula with a weight of zero, so that these formulas make no contribution before the transfer.

Note that we can also encode each action model in source domains as a set of formulas with **F1–F3** by requiring its corresponding formulas to be always true. For example, we can encode the precondition of the action model *board-truck* by the formula according to **F3**:

$$\forall i.(\textit{board-truck}\ ?d\ ?t\ ?l\ i) \rightarrow (\textit{at}\ ?d\ ?l\ i).$$

Likewise, we can encode the effects according to **F1–F2**. Thus, for each source domain D_i, we can encode the action models $\mathcal{A}_i$ with a list of formulas F_i.

The above formulas **F1–F3** are responsible for generating the space of all possible candidate literals and formulas for action models, in terms of their preconditions and effects, but constraints are then needed for a learning system to select a set of consistent formulas that are consistent with the training examples, which are the plan traces. We impose the following three constraints below, which were introduced by our previous work [72].

Action-consistency constraint. First, we wish the model learned do not conflict with themselves. Thus, if an action a has an effect p, then this same action a cannot have an effect $\neg p$ in the same state after a is executed. We formulate this constraint as follows. For each action a and literal p, if in its effect there are two formulas "$f_1 = L1 \rightarrow p$" and "$f_2 = L2 \rightarrow \neg p$", where both $L1$ and $L2$ are a conjunction of literals, then we require that $L1$ and $L2$ are mutually exclusive. Notice that either $L1$ or $L2$ can be empty, i.e., "$f_1 = p$" or "$f_2 = \neg p$". $L1$ and $L2$ are not mutually exclusive when either of them is empty. For example, "(in ?x)" and "(not (in ?x))" should not be chosen as the effects of the action "(take-out ?x)" at the same time.

Plan-consistency constraint. We require that the action models learned are consistent with the training plan traces. This constraint is imposed on the relationship between ordered actions in plan traces, and it ensures that the causal links in the plan traces are not broken. That is, for each precondition p of an action a_j in a plan trace, either p is in the initial state, or there is an action a_i $(i < j)$ prior to a_j that adds p and there is no action a_k $(i < k < j)$ between a_i and a_j that deletes p. For each atom q in a state s_j, either q is in the initial state s_0, or there is an action a_i before s_j that adds q while no action a_k deletes q. We formulate the constraint below:

$$p \in \text{PRE}(a_j) \wedge p \in \text{EFF}(a_i) \wedge \neg p \notin \text{EFF}(a_k)$$

and

$$q \in s_j \wedge \big(q \in s_0 \vee \big(q \in \text{EFF}(a_i) \wedge \neg q \notin \text{EFF}(a_k)\big)\big),$$

where $i < k < j$, $\text{PRE}(a_j)$ is a set of preconditions of the action a_j and the state s_j is composed of a set of propositions (or predicates).

For example, for the action "put-in", since "is-at" is a precondition of "put-in", if it is not in the initial state, there should be an action "move" that adds it, and it is never deleted by the actions (if any) between "move" and "put-in". Otherwise, the action "put-in" cannot be executed after the action "move".

Non-empty constraint. We wish to avoid the extreme situation where in a plan trace, the action models are learned such that all except one of the actions in the trace have non-empty preconditions and effects. Although such an action model is not incorrect, it is rather undesirable. To avoid such cases, we require that the preconditions and effects of the actions we learn should be non-empty. In other words, for each action model a, the following formula should hold:

$$\text{PRE}(a) \neq \emptyset \wedge \text{EFF}(a) \neq \emptyset.$$

These constraints can be used in the learning phase for building the action models in an MLN, or it can be used as post-processing constraints for selecting the effects after they are learned. In our experiment, these constraints are enforced.

3.3. [Step 4:] Transfer knowledge from source domains

In this step, we aim to transfer knowledge from source domains to the target domain. We explore two phases to achieve the task. In the first phase, i.e., Step 4a of Algorithm 1, we require that the transferred knowledge should best explain the training data from the target domain. After that, we explore extra knowledge from the Web to improve the quality of transferred information. In the following two subsections we present these two phases in detail.

3.3.1. Transferring based on plan traces

In Step 4a of Algorithm 1, the procedure "w = plan_traces_based_transfer(F_0, $\mathcal{A}_1, \ldots, \mathcal{A}_n$, DBs)" aims to transfer knowledge from source domains $D_0, \ldots, D_n$ to target domain D_0, which is represented as a set of weighted formulas F_0 built from target domain D_0. Specifically, we build all the possible mappings of formulas between source and target domains, and accumulate the weights of formulas mapped to F_0. The detailed description of the procedure "plan_traces_based_transfer" is shown in Algorithm 2.

Algorithm 2 Transferring based on plan traces: w = plan_traces_based_transfer(F_0, $\mathcal{A}_1, \ldots, \mathcal{A}_n$, DBs).

Input: (1) a set of possible formulas in D_0: F_0; (2) sets of action models $\mathcal{A}_1, \ldots, \mathcal{A}_n$ from $D_1, \ldots, D_n$; (3) a set of proposition databases DBs.
Output: the weights w of formulas in F_0.

1: initialize the weights w of formulas in F_0 to be zero;
2: **for** each source domain $\mathcal{A}_i$ **do**
3: encode action models in $\mathcal{A}_i$ as formulas F_i;
4: build a set of mappings MAP_i between $\mathcal{A}_i$ and D_0;
5: **end for**
6: let $\mathcal{F} = \{F_i\}_{i=1}^n$ and $\mathcal{M} = \{MAP_i\}_{i=1}^n$, and learn the best mapping: $\langle w^*, MF_0^* \rangle$ = build_transferred_formulas($\mathcal{M}$, $\mathcal{F}$, DBs);
7: **for** each $f_j \in F_0$ and its corresponding weight $w_j \in w$ **do**
8: if $\exists f_k \in MF_0^*$ that $f_k = f_j$, then $w_j = w_j + w_k^*$;
9: **end for**
10: **return** w;

In Step 3 of Algorithm 2, we encode each action model in $\mathcal{A}_i$ with formulas in the form of **F1–F3**. For example, "(*ontable* ?x)" is a precondition of action "(pickup ?x)" in the *blocks* domain (viewed as one of the sources). This relation can be encoded with formula (in the form of **F3**):

$$\forall i.(pickup\ ?x\ i) \rightarrow (ontable\ ?x\ i).$$

Thus, we can encode action models $\mathcal{A}_i$ with a set of formulas F_i. In Step 4 of Algorithm 2, we build all the possible mappings between action models $\mathcal{A}_i$ and actions/predicates in D_0. Specifically, we map both the predicates and the action schemas between $\mathcal{A}_i$ and D_0. We exploit the useful information a STRIPS action model provides: each parameter of a predicate or an action is associated with a *type*, e.g., the *type* of *driver* in the domain *driverlog*; all the parameters of a precondition (or an effect) of an action should be included by the action's. We do the mapping by the following two phases:

- First, we map predicates between D_0 and $\mathcal{A}_i$, and build a *unifier* between the *type*s of parameters of two predicates to *substitute* other predicates' *type*s. Notice that the number of parameters of two mapped predicates should be the same. For instance, in Fig. 1, when mapping "(clear ?y)" of the domain *blocks* and "(empty ?truck)" of the domain *driverlog*, the *type*s of "?y" and "?truck", *block* and *truck* respectively, are mapped and this mapping will be used to constrain the mappings of other predicates.
- Second, we map actions between D_0 and $\mathcal{A}_i$, constrained by the mappings built in the first step. For instance, in Fig. 1, assume that "(empty ?truck)" is an effect of the action "board-truck" (which needs to be learned), and "(empty ?truck)" and "(clear ?y)" have been mapped by the first step, the action "stack" in domain *blocks* should include a parameter with a type *block* if we want to map "stack" to "board-truck". That is because "clear" will be an effect of "stack" according to the mapping. This constraint can be addressed by the chain: "board-truck"⇒"empty"⇒"clear"⇒"stack"⇒"board-truck".

Formally, a mapping built by the above two steps is recorded by a set of predicate pairs or action pairs, i.e., $\{\langle s_i, t_0 \rangle\}$, where s_i is a predicate (or an action) in source domain $\mathcal{A}_i$ and t_i is a predicate (or an action) in target domain D_0. Note that we assume mappings are one-to-one and possibly partial. Since a predicate in $\mathcal{A}_i$ can be mapped to different predicates in D_0, there could be different mappings between $\mathcal{A}_i$ and D_0. We denote the set of mappings by MAP_i. It is possible that the size of mappings is exponentially large since each predicate (action) in the source domain can be mapped to any predicate (or action) in the target domain. We thus explore a greedy way to approximate the big set of mappings (although there could be large amount of approaches to prune the set of mappings). Specifically, when we build a predicate pair (or an action pair), we will look at the number of candidate predicate pairs (or action pairs) available to be built in the next step. Each time we choose the predicate pair (or action pair) with the largest number of candidate predicate pairs (or action pairs) left. In other words, we greedily build mappings with the largest number of predicate pairs and action pairs to maximally map source and target domains. Despite its greedy nature, experiments have indicated that it can efficiently match some of the world's largest knowledge bases with high accuracy [34].

In Step 6 of Algorithm 2, we put all the formula sets with respect to all the source domains together and denote the result by $\mathcal{F} = \{F_i\}_{i=1}^n$, put all the mapping sets together and denote the result by $\mathcal{M} = \{MAP_i\}_{i=1}^n$, and build a set of transferred formulas (denoted by MF_0^*) using DBs (which corresponds to a subset of formulas in target domain D_0) and the weights w^* of these formulas. We will describe the detailed procedure of "build_transferred_formulas" in Algorithm 3 in the next paragraph. After building the set of transferred formulas and weights, we transfer the weights of formulas in MF_0^* to formulas in F_0, as is stated by Steps 7 and 8 of Algorithm 2, and return the final weights of formulas F_0. A weight associated to a formula suggests the "degree" that the formula will hold. Thus, we use the weights of the resulting formulas

MF_0 as the bridge to transfer the information from multiple source domains to the target domain. We simply add all the weights "$w_j = w_j + w_k^*$" in Step 8, so that the "degree" is strengthened if $w_k^* > 0$, or weakened if $w_k^* < 0$. Readers may ask why not just build action models directly from MF_0, instead of F_0. The reason is we would like to guarantee that the conditions not being mapped by source domains can also be learned. MF_0 may only encode a subset of all the conditions of action models to be learned. Intuitively, in Algorithm 2 we first build a set of mappings between source and target domains, and then learn weights of the mapped knowledge or formulas based on the mappings. As we can see from Algorithm 1, the lower the weights are, the less important (for learning the target model) the mapped knowledge is. If the source is not "that" similar to the target, the weights of the mapped knowledge will be low according to the weight learning procedure (i.e., Algorithm 4), since the mapped knowledge will not "that" strongly satisfy the input plan traces.

Algorithm 3 Build a set of transferred formulas: build_transferred_formulas($\mathcal{M}$, $\mathcal{F}$, DBs).

Input: (1) sets of mappings: $\mathcal{M}$; (2) sets of formulas: $\mathcal{F}$; (3) DBs.
Output: MF_0^* and its corresponding weights w^*.

1: initiate WPLL* as a small enough value;
2: randomly select a mapping map_i from each $MAP_i \in \mathcal{M}$, resulting in $\langle map_1, \ldots, map_n \rangle$;
3: **repeat**
4: randomly select a set MAP_i from $\mathcal{M}$, and $\mathcal{M} = \mathcal{M} - \{MAP_i\}$;
5: **for** each $map' \in MAP_i$ **do**
6: replace map_i with map', i.e., $map_i = map'$;
7: for each map_i, do $MF_i = mapformulas(F_i, map_i)$, where $F_i \in \mathcal{F}$, resulting in $\langle MF_1, \ldots, MF_n \rangle$;
8: let $MF_0 = \bigcup_1^n MF_i$;
9: $\langle w, \text{WPLL} \rangle = learnweights(0, MF_0, \text{DBs})$;
10: if WPLL* < WPLL then WPLL$^* =$ WPLL, $MF_0^* = MF_0$, $w^* = w$;
11: **end for**
12: **until** $\mathcal{M}$ is empty;
13: **return** MF_0^* and w^*;

In Algorithm 3, we aim to transfer knowledge from all source domains simultaneously. We randomly select a mapping from each MAP_i in M, resulting in a mapping vector $\langle map_1, \ldots, map_n \rangle$. After that, we greedily replace map_i with other mappings in MAP_i and apply map_i to F_i to generate a new list of formulas, denoted by MF_i. Since different mapping selections will result in different MF_0, we provide a score function called WPLL to measure the quality of the mapping, which is used in the MLN system Alchemy [53,30]. We will give the definition of WPLL in the next paragraph. With the score function WPLL, we perform a procedure "learnweights" to learn weights of current transferred formulas MF_0, and keep the one with the highest WPLL. The whole procedure of Algorithm 3 stops when all the mappings from all source domains are selected (i.e., $\mathcal{M}$ is empty), and outputs the set of transferred formulas MF_0^* and their corresponding weights w^*. Note that in Step 7 of Algorithm 3 the procedure "mapformulas" is to apply the mapping map_i to F_i, resulting in a set of transferred formulas MF_i. The procedure "learnweights" in Step 9 of Algorithm 3 is to learn the weights for MF_0 and calculate the corresponding value of WPLL. The detailed description is given in Algorithm 4.

Algorithm 4 Learn the weights of formulas MF_0: $learnweights(w^{init}, MF_0, \text{DBs})$.

Input: (1) a set of formulas: MF_0; (2) DBs.
Output: the weights w of MF_0 and the score WPLL.

1: initiate $w^0 = w^{init}$, $i = 0$;
2: **repeat**
3: calculate WPLL(w^i) using DBs and MF_0;
4: $w^{i+1} = w^i + \lambda * \partial \text{WPLL}(w^i)/\partial w^i$;
5: $i = i + 1$;
6: **until** i is larger than a maximal iterative number N;
7: $w = w^N$ and WPLL $=$ WPLL(w^N);
8: **return** w and WPLL;

In Step 3 of Algorithm 4, the score function WPLL, short for Weighted Pseudo-Log-Likelihood [53], is defined by

$$\begin{aligned} \text{WPLL}(w) &= \sum_{l=1}^{n} \log P_w\big(X_l = x_l \big| MB_x(X_l)\big) \\ &= \sum_{l=1}^{n} \log \frac{C_{(X_l = x_l)}}{C_{(X_l=0)} + C_{(X_l=1)}}, \end{aligned}$$

where

$$C_{(X_l=x_l)} = \exp \sum_{f_i \in F_l} w_i f_i\big(X_l = x_l, MB_x(X_l)\big),$$

and $x = (x_0, x_1, \ldots, x_l, \ldots, x_n)$ is a possible world (a database DB), where $x_l = 1$ means the lth grounding appears in DB and $x_l = 0$ means the lth grounding does not appearing in DB. n is the number of all the possible *groundings* of atoms appearing in all the formulas MF_0, and X_l is the lth *grounding* of the all. $MB_x(X_l)$ is the state of the Markov blanket of X_l in x. A Markov blanket of a ground atom is a set of ground atoms that appear in some grounding of a formula with it. F_l is the set of ground formulas that X_l appears in, and $f_i(X_l = x_l, MB_x(X_l))$ is the value of the feature corresponding to the ith ground formula when $X_l = x_l$ and Markov blanket state $MB_x(X_l)$. The value is 1 when the ith ground formula is *true*, otherwise it is 0. We refer the readers to [53] for more details, including about how to calculate $\partial \mathtt{WPLL}(w^i)/\partial w^i$.

Note that $w_i > 0$ ($w_i < 0$) suggests that the ith ground formula should be *true* (*false*). The more the formulas MF_0 are satisfied in x, the larger the $C_{(X_l = x_l)}$ is, as a result, the larger the $\mathtt{WPLL}(w)$ is, since $C_{(X_l=0)} + C_{(X_l=1)}$ is a constant. That is to say WPLL is defined to describe the "degree" of all the formulas MF_0 being satisfied in x. That is to say, WPLL could be used as a metric to measure the "degree" of a set of formulas being satisfied in a possible world. Using this metric, we can filter sets of formulas whose WPLL is small, and finally filter the source domains that provide these formulas. In this way, we can avoid negative effects played on our learning result by these source domains. It is straightforward to integrate the filtering process into our algorithm TRAMP. We will show the idea in our experiment.

3.3.2. Transferring based on Web search

In this subsection, we present another transferring phase that exploits Web search to bridge source and target domains. Based on the weights attained by Step 4a of Algorithm 1, in this phase we transfer knowledge from source domains by further updating the weights w of formulas F_0. The procedure is shown in Algorithm 5. For each action model set $\mathcal{A}_i$ from source domain D_i, we encode $\mathcal{A}_i$ with a set of formulas F_i as is done by Step 3 of Algorithm 2. After that, we calculate the similarity between formulas in F_0 and F_i, and accumulate the weights w of F_0, as is described by Steps 4–9 of Algorithm 5. Note that in Step 5 of Algorithm 5, the procedure "can_map" returns *true* if the number of parameters of each literal in formulas s and t is the same; otherwise, it returns *false*. In the following, we will focus on Step 6 of Algorithm 5, about how to build the similarity function from across domains using Web search. This similarity function is used to transfer knowledge from source domains to the target domain.

Algorithm 5 Update weights of formulas via Web search: w = transfer_web_searching(w, F_0, $\mathcal{A}_1, \ldots, \mathcal{A}_n$).

Inputs: weights w, formulas F_0, source domains $\mathcal{A}_1, \ldots, \mathcal{A}_n$;
Output: updated w of formulas F_0.

```
1: for each set of action models A_i from source domain D_i do
2:     encode A_i as formulas F_i;
3:     for each formula s ∈ F_0 and each formula t ∈ F_i do
4:         if can_map(s, t) == true then
5:             sim = similarity(s, t);
6:             w_s = w_s + sim, where w_s ∈ w;
7:         end if
8:     end for
9: end for
10: return w;
```

There are emerging Web pages that describe semantics of actions with the proliferation of the Web services. These Web pages encode the human understanding to the action semantics, such as under what conditions an action can be executed, what effects an action will produce, etc. This semantics can greatly help in measuring the similarities between the actions.

For each formula s and t, we can exploit Web search to extract Web pages related to them. For example, let s be "$\forall i.(stack\ ?x\ ?y\ i) \rightarrow (clear\ ?y\ i)$" from domain *blocks*, where *clear* is a predicate and *stack* is an action. We can search with query "stack and clear". 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, let t be "$\forall i.(board\text{-}truck\ ?x\ ?y\ ?z\ i) \rightarrow (empty\ ?y\ i)$" from domain *driverlog*. We can extract Web pages by searching a query "board-truck and empty". Note that we ignore the difference of the searching results from different predicate-action orders, such as "clear and stack" and "stack and clear". 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. Note that the plain text is composed of text from table items, labels, captions, other plain text from an *html* file. Other text embedded in other objects such as pictures in the *html* file is ignored. In the experiment we will sample top 20 Web pages for extracting documents when searching a keyword. 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) [29] of each word w of d_i:

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

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 is a measure of whether the term w is common or rare across all documents. 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, since the term "the" is so common, this will tend to incorrectly emphasize documents that happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms. The term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms, unlike other less common words. An inverse document frequency factor is thus incorporated to diminish the weight of terms that occur very frequently in the document set and increase the weight of terms that occur rarely.

Therefore, for a formula s, we can search it and get a set of documents

$$\mathcal{D}_1 = \{x_i \mid i = 1, \ldots, m_s\},$$

with each x_i as a tf–idf vector. Similarly, for another formula t, we can get another set of documents

$$\mathcal{D}_2 = \{y_i \mid i = 1, \ldots, m_t\},$$

with each y_i as a tf–idf vector.

After extracting the Web data $\mathcal{D}_1$ and $\mathcal{D}_2$, we measure the similarity between formulas s and t. Note that a possible choice to calculate the similarity between two data distributions is using the Kullback–Leibler (KL) divergence [32]. 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) [8] to calculate the similarity, which can directly measure the distribution distance without the density estimation. Let $\mathcal{F}$ be a class of functions $f: X \to \mathbb{R}$. 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 Borel probability distributions p and q respectively. The empirical estimate of distance between p and q defined by MMD is

$$MMD[\mathcal{F}, X, Y] = \sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} f(x_i) - \frac{1}{n} \sum_{i=1}^{n} f(y_i) \right). \tag{2}$$

Considering the universal reproducing kernel Hilbert spaces (RKHS) $\mathcal{H}$, we interpret the function f as the feature mapping function $\phi(\cdot)$ of a Gaussian kernel [8], i.e., $\phi(\cdot)$ satisfies

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

where σ is the kernel width for the Gaussian kernel function. Then we have the following result given by [8]:

$$\begin{aligned} MMD^2[\mathcal{F}, X, Y] &= \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) - \frac{1}{n} \sum_{j=1}^{n} \phi(y_j) \right\|_{\mathcal{H}}^2 \\ &= \frac{1}{m(m-1)} \sum_{i \neq j}^{m} k(x_i, x_j) + \frac{1}{n(n-1)} \sum_{i \neq j}^{n} k(y_i, y_j) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k(x_i, y_j). \end{aligned}$$

Given the Web data $\mathcal{D}_1$ and $\mathcal{D}_2$, we can finally have the similarity function between s and t defined by

$$similarity(r_1, r_2) := MMD^2[\mathcal{F}, \mathcal{D}_1, \mathcal{D}_2].$$

We can see that using the function "similarity" we can calculate the similarity between two formulas s and t in Step 6 of Algorithm 5.

Note that we do not extract structural information from documents. Given a rule $x \to y$ from a source domain, we check similarity between keywords $\langle x, y \rangle$ and $\langle a, b \rangle$, if $\langle a, b \rangle$ is highly similar to $\langle x, y \rangle$, we will transfer the relation $x \to y$ from the source to the target by saying $a \to b$. If both keywords are existent in a common document, the document will indeed enhance the similarity of the keywords. This negative effect is, however, expected to be restrained by looking at multiple documents and calculating the similarity statistically.

3.4. [Steps 5–7:] Generating action models

In Steps 5–7 of Algorithm 1, we perform the learning procedure "learnweights" (Algorithm 4) with the initial weights w attained by Step 4 of Algorithm 1. From the definition of WPLL, we can see 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 final weight of a candidate formula is a confidence measure of that formula. Intuitively speaking, the larger the weight of a candidate formula is, the more likely that formula is true. We can thus perform the following phases to generate the final action models $\mathcal{A}$ based on the weights of candidate formulas.

- First, we choose a weight threshold δ based on the error estimates in the evaluation criteria using the training plan traces. We will test different values of δ in the experiment section.
- Second, we select all the formulas from F_0, whose weights are larger than the threshold δ.
- Third, we convert the selected formulas to action models according to **F1–F3**. For instance, if the weight of a formula generated by **F1** is larger than the threshold, then the formula is selected and the predicate p in it will be transformed to an effect of the action a in it.

Note that converting the selected formulas is straightforward by scanning each formula and collecting preconditions and effects of actions (i.e., with linear time cost). For example, let $\textit{board-truck}(i) \rightarrow \neg \textit{driving}(i) \wedge \textit{driving}(i+1)$ be a formula selected. Predicate *driving* will be directly viewed as an adding effect of action *board-truck* according to **F1**.

3.5. Discussion

Considering the two phases of transferring in Step 4 of Algorithm 1, we can see that there can be three different transfer instantiations spanned by TRAMP, as are given below.

(I1) As the first instantiation, we just exploit the transferring procedure based on plan traces. This instantiation can be developed by simply ignoring Step 4b and keeping Step 4a in Algorithm 1.
(I2) As the second instantiation, we just exploit the transferring procedure based on Web search. This instantiation can be simply developed by deleting Step 4a, and initializing weights w to be zero in Step 4b in Algorithm 1 (keeping Step 4b).
(I3) As the third instantiation, we use both transfer phases, as is described by Algorithm 1. Note that it means instantiation **(I3)** when we say TRAMP in the following sections.

Note that each instantiation functions well with respect to different number of inputted source domains, including no source domain inputted, which corresponds to a non-transfer learning approach (in this case, Step 4 is ignored in Algorithm 1).

4. Experiments

In this section we would like to evaluate TRAMP in different aspects. We first present the plan traces we used in the experiment and then define the evaluation criteria of the learning quality. After that, we present the experimental results.

4.1. Training data and evaluation criteria

We evaluate TRAMP in domains from the second and third International Planning Competitions (IPC-2[1] and IPC-3[2]), i.e., *blocks*[1], *depots*[2], *driverlog*[2], and *zenotravel*[2], as well as four synthetic domains *laundry*, *dishwashing*, *coal-mining* and *urban-rescue*, which are about how to do laundering, dishwashing, mining coal and rescuing in urban environments. The numbers of predicates and actions, are shown in Table 1, where the first four domains have the same semantics defined by IPC-2 or IPC-3, the last four synthetic domains are addressed as follows.

laundry: The domain *laundry* is created based on the simplified process of laundering, which is composed of 6 actions, i.e., *turn-on*, *turn-off*, *put-detergent*, *put-in*, *get-out* and *wash*. Action *turn-on* suggests turning on the tap to get water for the washer when the washer has not enough water. Action *turn-off* indicates turning off the tap when the washer has enough water. Action *put-detergent* suggests putting detergent to the washer when there is not enough detergent in the washer. Actions *put-in* and *get-out* suggest putting clothing into the washer and getting out clothing from the washer, respectively. Finally, action *wash* indicates the washer is started to wash clothing when clothing, water and detergent are ready.

dishwashing: The domain *dishwashing* is extracted from the process of dishwashing. *Dishwashing* has 5 actions, i.e., *put-dish*, *get-water*, *put-cleanser*, *clean* and *dry*. Action *get-dish* indicates putting dishes to the sink. Action *get-water* suggests putting water to the sink if there is not enough water in the sink. Action *put-cleanser* suggests putting cleanser to the sink if there is no cleanser in the sink. Action *clean* means starting to wash dishes until they are clean when both water and cleanser are ready. Action *dry* indicates drying the dishes when they are washed clean.

Table 1
Features of the testing domains.

Domains	Number of predicates	Number of action models
blocks	5	4
depots	6	5
driverlog	5	6
zenotravel	4	5
laundry	13	6
dishwashing	8	5
coal-mining	7	5
urban-rescuing	16	8

coal-mining: The domain *coal-mining* describes the process of coal mining, which is assumed to be with 5 actions, i.e., *drill*, *fill-explosive*, *blast*, *remove*, *load* and *move*. Action *fill-explosive* suggests a mining robot fills a drill hole with explosives when explosives and the robot are in the same location of the drill hole; *blast* suggests a mining robot blasts the overburden of the mining area when explosives are filled in the drill hole; *remove* indicates a robot removes the overburden by draglines; *load* suggests the coal is loaded on to large trucks or conveyors; *transport* specifies trucks or conveyors transport the coal from a location to a coal preparation plant.

urban-rescuing: The domain *urban-rescuing* addresses the simplified scenario of urban search and rescue (USAR[3]), which involves the location, extrication, and initial medical stabilization of victims trapped in confined spaces due to natural disasters, structural collapse, transportation accidents and collapsed trenches. We describe the domain with 8 actions, i.e., *walk*, *leverage*, *crib*, *lift*, *drag*, *carry*, *mark*, *aid*. Action *walk* specifies a robot walking from one location to another; *leverage* suggests a robot removes an object near a victim using leveraging when the object blocks the robot's path to the victim; *crib* indicates that a robot constructs a rectangular wooden framework known as a box crib underneath the object to stabilize the object to avoid further injury by the object; *lift* specifies a robot lifts an object close to a victim; *drag* specifies a robot drags a victim from one location to another; *carry* suggests a robot carries a victim in its arms; *aid* indicates a robot renders medical aid to victims if they are in danger; *mark* suggests a robot marks buildings to indicate they have been searched.

We collected 150 plan traces as the training data from each planning domain. These plan traces were generated by generating plans from the given initial and goal states in these planning domains using the human encoded action models and a planning system, FF planner.[4] We randomly selected a percentage of propositions to create partial states between actions in plan traces. We use 30% as the default value of partialness of states, which suggests 30 propositions will be randomly selected if a full state is described by 100 propositions. We also randomly selected a percentage of partial states in plan traces. We use $\frac{2}{3}$ as the default percentage which suggests there two partial states are randomly selected among three continuous states in plan traces. We assume action models that are viewed as target domains are *unknown*, while action models that are viewed as source domains are *known*.

We define error rates of our learning algorithm as the difference between our learned action models and the action models (or the "ground truth") created by domain experts (assuming domain models created by domain experts are reliable or executable). The difference of the learned models can be used to measure the manual effort needed to correct the learnt model to be the perfect model (or the ground truth). The lower the error rate is, the less manual effort is needed. 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. 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 counts 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

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

where we assume the error rates of preconditions and effects as 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

$$R(A) = \frac{1}{|A|}\sum_{a \in A} R(a),$$

[3] The abbreviation USAR has been adopted by the UN's International Search and Rescue Advisory Group (INSARAG) to signify an Urban Search and Rescue team all over the world.

[4] http://fai.cs.uni-saarland.de/hoffmann/ff.html.

where $|A|$ is the number of A's elements, and the accuracy of all the action models is defined by $Acc = 1 - R(A)$. From our definition of the evaluation criteria, the smaller of the error rate of the learned action model, the better quality of the learned action model is.

Note that evaluation criteria define the objective functions to be optimized for. They should truly reflect how important a piece of learned object is in the overall process of planning. The error bounds we propose here correspond to the 'static' evaluation of the action models, when we compare the models' syntactic structure to the ground truth model's corresponding structure. What we can do in the future, is to explore further the 'semantic' nature of errors, not just in terms of missing conditions and effects, but also includes costs that will be incurred when a model is learned inaccurately. Some initial ideas towards this cost model are to conduct cross validation experiments with the learned model, and compare the plans obtained from the learned models with those in the ground truth model. However, this dynamic evaluation is highly dependent on the distribution of future problems the learned models are designed to solve, since for different groups of future problems, the semantic evaluation model will give different results. The syntactic model we use in this paper, while being simple, can in fact be considered as a distribution independent model, which provides a baseline for more sophisticated model evaluation.

4.2. Experimental results

In the next subsection, we test TRAMP in the following different aspects:

- We compare TRAMP with the state-of-the-art transfer learning system LAWS [73] that transfers knowledge based on Web search, to demonstrate that exploiting transfer based on both Web search and plan traces is indeed better than only exploiting Web search.
- Likewise, we compare TRAMP with t-LAMP [70] that transfers knowledge based on plan traces from the target domain, to show the advantage of exploiting both Web search and plan traces.
- We compare TRAMP with state-of-the-art non-transfer learning system ARMS [67] by setting the number of source domains to be zero, to show that TRAMP functions well even though there is no source domain available.
- We run TRAMP with three different settings, i.e., transferring knowledge from five source domains, three source domains, and one source domain, respectively, and compare the results of these cases to see the impact of the number of source domains.
- We test the relation between the accuracy of TRAMP and the score function WPLL (defined in Section 3.3.1) to see which source domains are transferable to the target domain according the score function, i.e., which source domains can be transferred to help learn target domain models.
- Finally, we show the running time of TRAMP to test its efficiency.

4.2.1. Comparison between TRAMP and LAWS

We first compare the TRAMP algorithm to another transfer learning approach LAWS [73] which is based on Web search, to see the advantage of exploiting transfer based on both plan traces and Web search, as is done by TRAMP. Since LAWS aims at transferring from one source domain, we set the number of source domains in TRAMP to be one as well to make the comparison fair. We tested both TRAMP and LAWS in eight transfer cases, i.e., *driverlog*→*depots*, *depots*→*driverlog*, *blocks*→*zenotravel*, *zenotravel*→*blocks*, *laundry*→*dishwashing*, *dishwashing*→*laundry*, *coal-mining*→*urban-rescuing* and *urban-rescuing*→*coal-mining*. Note that *driverlog*→*depots* indicates learning action models of target domain *depots* via transferring knowledge from source domain *driverlog*, likewise for other cases. In each case, we set the threshold δ in Algorithm 1 to be 0.5, sampled top 20 Web documents from search engines "google"[5] and "wikihow",[6] respectively, and ran TRAMP and LAWS five times to calculate an average of accuracy. We exploit these settings in all experiments unless otherwise specified. The results are shown in Fig. 6.

From Fig. 6, we can see that accuracies of TRAMP are generally higher than LAWS in all transfer cases, which suggests exploiting transfer based on both plan traces and Web search to bridge source and target domains is better than exploiting transfer just based on Web search. This is because plan traces exploited by TRAMP provide useful information for helping bridging source and target domains, which results in better transfer than LAWS. We can also observe that the advantage of TRAMP becomes smaller when the number of plan traces becomes larger. This is because knowledge from the target domain is large enough for acquiring high-quality action models when the number of plan traces becomes larger, which correspondingly reduces the impact of transferred knowledge; the better the transfer ability is, the faster the impact is reduced, which suggests the impact of TRAMP is reduced faster than LAWS. We also observe that accuracies of both TRAMP and LAWS increase when the number of plan traces becomes larger. This is consistent with our intuition since the larger the plan traces are, the more information can be exploited to acquire high-quality action models.

[5] http://www.google.com.
[6] http://www.wikihow.com/.

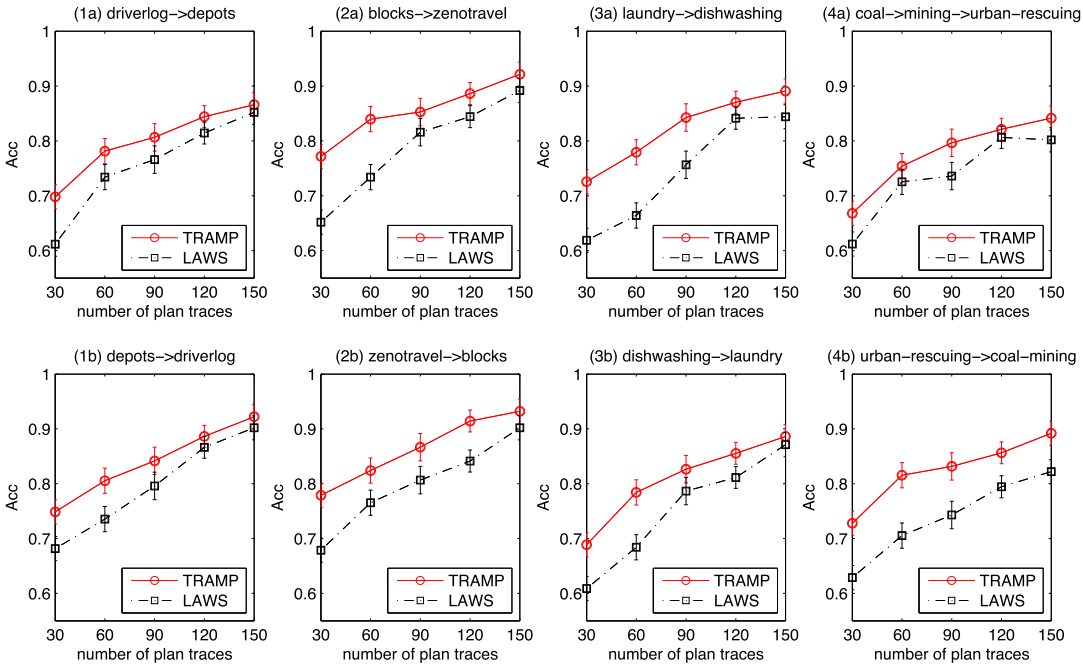

Fig. 6. The comparison between TRAMP and LAWS in eight transfer cases: (1a) *driverlog*→*depots*, (1b) *depots*→*driverlog*; (2a) *blocks*→*zenotravel*, (2b) *zenotravel*→*blocks*; (3a) *laundry*→*dishwashing*, (3b) *dishwashing*→*laundry*; (4a) *coal-mining*→*urban-rescuing*, (4b) *urban-rescuing*→*coal-mining*.

4.2.2. Comparison between TRAMP and t-LAMP

We would like to further compare TRAMP to another state-of-the-art transfer learning algorithm *t*-LAMP which explores transferring between source and target domains based on plan traces. Since *t*-LAMP is capable of transferring knowledge from multiple source domains, we set both TRAMP and *t*-LAMP with seven source domains to make the comparison fair. We tested both TRAMP and *t*-LAMP in eight transfer cases which are denoted by *depots*, *blocks*, *zenotravel*, *driverlog*, *dishwashing*, *laundry*, *coal-mining* and *urban-rescuing*, respectively, where *depots* suggests *depots* is the target domain while others are source domains. Similar to the comparison with LAWS in Section 4.2.1, we set the threshold δ in Algorithm 1 to be 0.5 and ran both TRAMP and *t*-LAMP five times to calculate an average of accuracy. We varied the number of plan traces to see the change of accuracies. The results are shown in Fig. 7.

From Fig. 7, we can see that accuracies of both TRAMP and *t*-LAMP increase when the number of plan traces becomes larger. This is consistent with our intuition since the larger the number of plan traces is, the more information can be used to help improve the learning accuracies. We can also observe that in all eight transfer cases accuracies of TRAMP are better than *t*-LAMP, which verifies that exploiting transfer based on both plan traces and Web search is indeed better than just exploiting transfer based on plan traces (as done by *t*-LAMP). The rationale is TRAMP takes advantage of additional knowledge from the Web to help calculating the similarity between source and target domains, which thus improves the transfer learning result. Besides, by observation, we can see that the difference between TRAMP and *t*-LAMP becomes smaller when the number of plan traces becomes larger. This is because when the number of plan traces becomes large, the information in the target domain becomes rich enough to learn high-quality action models, which weakens the impact of information transferred from source domains, particularly the one transferred via Web search. This makes the advantage of exploiting transfer based on both plan traces and Web search smaller.

4.2.3. Comparison between TRAMP and ARMS

We also would like to evaluate if TRAMP functions well when there is no source domain available. We thus compared TRAMP with a state-of-the-art learning system ARMS, which was capable of learning action models in non-transfer style. To compare with ARMS, we fed TRAMP with zero source domain (i.e., no source domain is provided), resulting in a non-transfer version of TRAMP. We varied the number of plan traces from 30 to 150, to see the change of accuracies of both TRAMP and ARMS. Each time we ran TRAMP and ARMS, we randomly selected plan traces to be fed to the systems and ran both TRAMP and ARMS five times to calculate an average of accuracy. We set δ to be three different values, i.e., 0.1, 0.5 and 1.0, to test the impact of the threshold δ in Algorithm 1. We calculated their corresponding accuracies with respect to different number of plan traces. The results of TRAMP and ARMS are shown in Figs. 8–10.

In Fig. 8, we show the learning accuracy of the *depots* domain with different values of δ. From the figure we can see that accuracies of both TRAMP and ARMS generally increase when the number of plan traces becomes larger in all three values of δ. This is consistent with our intuition, since the larger the number of plan traces is, the more information can

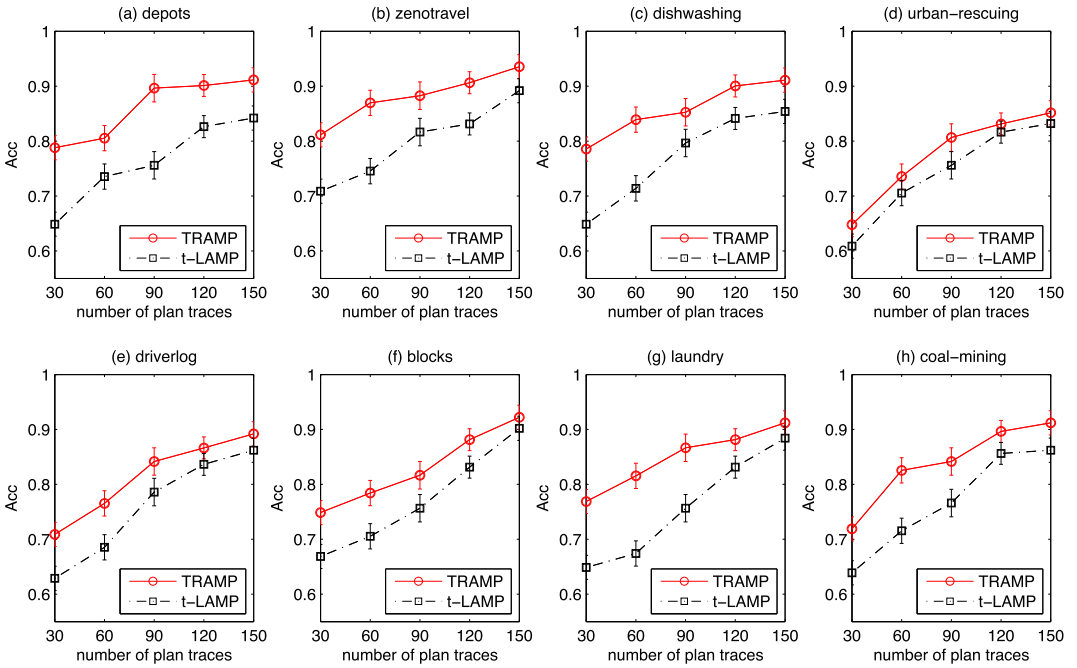

Fig. 7. The comparison between TRAMP and *t*-LAMP in eight transfer scenarios with target domains *depots*, *blocks*, *zenotravel*, *driverlog*, *dishwashing*, *laundry*, *coal-mining* and *urban-rescuing*, respectively.

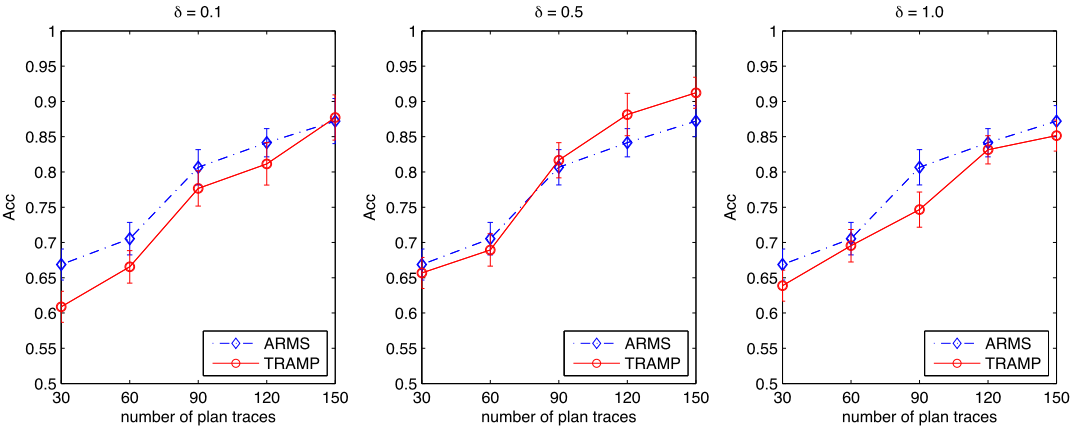

Fig. 8. Comparison between TRAMP and ARMS in the *depots* domain in three cases by setting δ to be 0.1, 0.5 and 1.0, respectively.

be used to help improve the learning accuracy. By comparing different cases of δ, we observe that the accuracy of TRAMP is almost the same as ARMS when $\delta = 0.5$ (although it is worse than ARMS when $\delta = 0.1$ or $\delta = 1.0$). This suggests that we should not set the value of δ to be too large (e.g., 1.0) or too small (e.g., 0.1). $\delta = 0.5$ would be a good choice for TRAMP to perform well (compared to ARMS) when there are no source domains provided as input. Indeed, the larger δ is, the more extra preconditions or effects of actions will be learnt in Algorithm 1; the smaller δ is, the more correct preconditions or effects of actions will be filtered (or will not be learnt); both scenarios will result in low learning accuracy. To further verify this fact, we tested TRAMP and ARMS in another domain *driverlog*. We can see that the result as shown in Fig. 9 is similar to Fig. 8, i.e., the accuracy of TRAMP is almost the same as ARMS when setting δ to be 0.5. In other six domains *blocks*, *zenotravel*, *dishwashing*, *laundry*, *urban-rescuing* and *coal-mining*, we fix $\delta = 0.5$ and see that the accuracies of TRAMP and ARMS are very close to each other (as shown in Fig. 10). This is because TRAMP and ARMS share the same idea of combining constraints and statistics. Specifically, both TRAMP and ARMS exploit the same sets of constraints when encoding action models in Step 3 of Algorithm 1; TRAMP calculates weights of formulas (i.e., **F1–F3**, which will be filtered based on threshold δ) to statistically best explain plan traces, while ARMS exploited frequent set mining (using a presented threshold) to learn weights of constraints; it is reasonable that the accuracy of TRAMP is close to ARMS when tuning a proper threshold, i.e., $\delta = 0.5$ in the experiment.

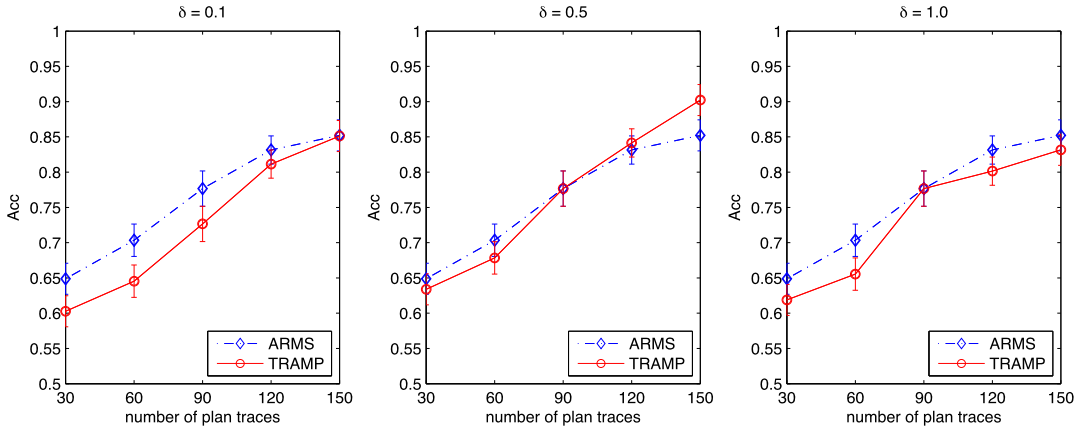

Fig. 9. Comparison between TRAMP and ARMS in the *driverlog* domain in three cases by setting δ to be 0.1, 0.5 and 1.0, respectively.

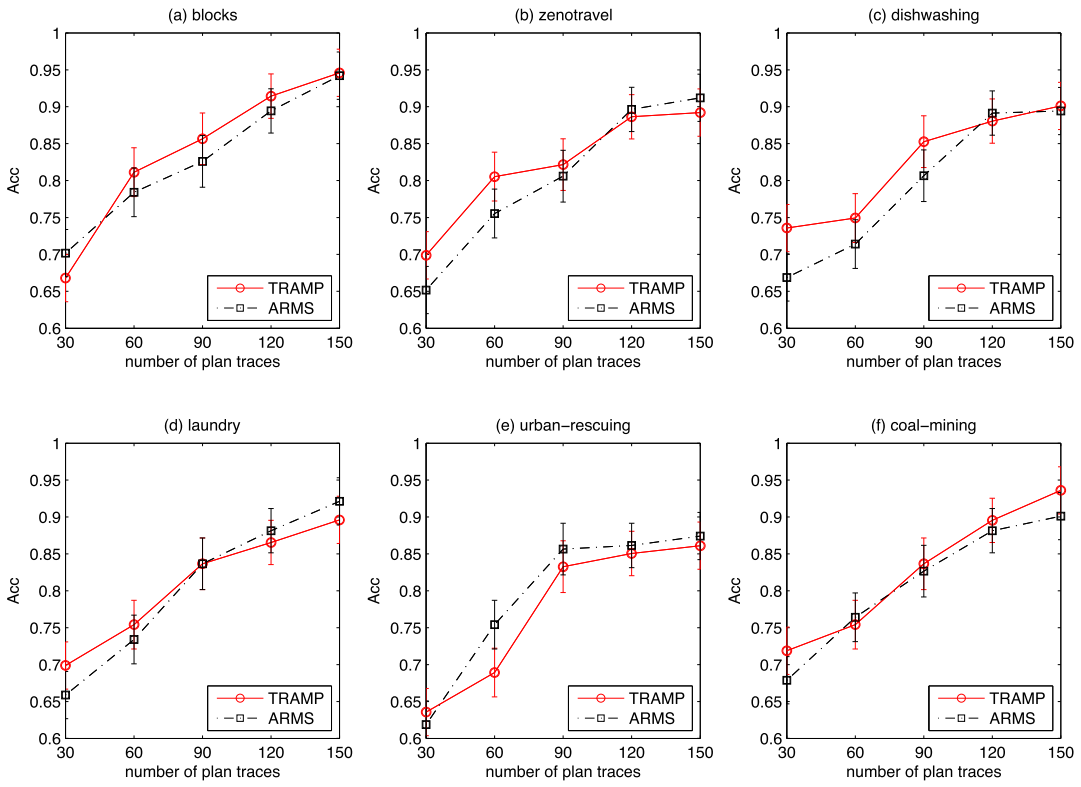

Fig. 10. Comparison between TRAMP and ARMS in domains *blocks*, *zenotravel*, *laundry*, *dishwashing*, *urban-rescuing* and *coal-mining*, fixing δ to 0.5.

4.2.4. Varying the number of source domains

We would like to see the change of accuracies when the number of source domains increases. We set the number of source domains to be 5, 3 and 1, and ran TRAMP to calculate their accuracies. The results are shown in Figs. 11 and 12, where *5-depots*, *3-depots* and *1-depots* suggest the curves are the results of learning action models from the target domain *depots* by transferring knowledge from five, three and one source domain, respectively, likewise for *5-driverlog*, *3-driverlog* and *1-driverlog*. Note that when learning action models for *depots*, we randomly select the source domains from *driverlog*, *blocks*, *zenotravel*, *laundry* and *dishwashing* and calculate an average accuracy for five selections; and randomly select the source domains from *depots*, *blocks*, *zenotravel*, *laundry* and *dishwashing* when learning action models for *driverlog*. When *depots* is used as the target domain, its action models are assumed to be unknown; when it is used as a source domain, its action models are assumed to be known in advance; likewise for the domain *driverlog*.

In Figs. 11 and 12 we can see that the more source domains are used for transfer, the higher the accuracies are. This observation suggests that the knowledge transferred indeed helps learn the action models in the target domain. However,

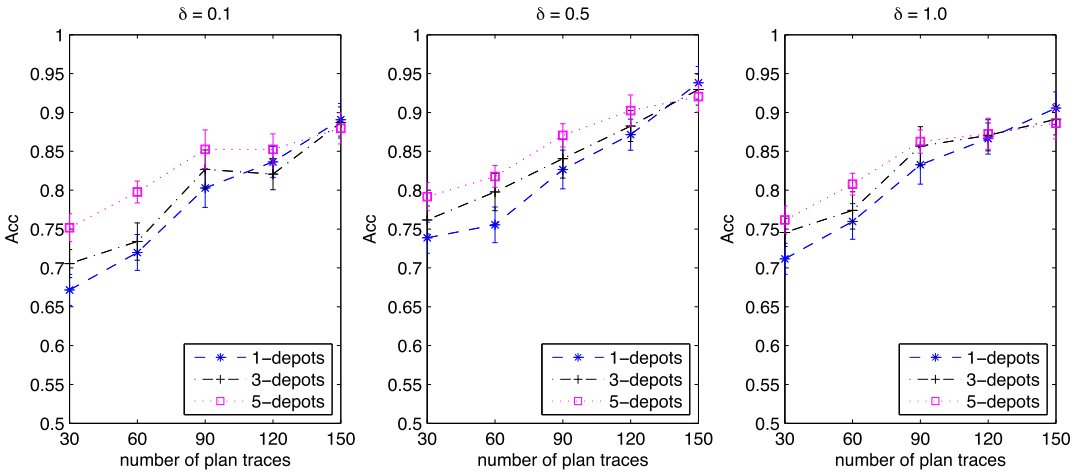

Fig. 11. Transfer learning action models of *depots*.

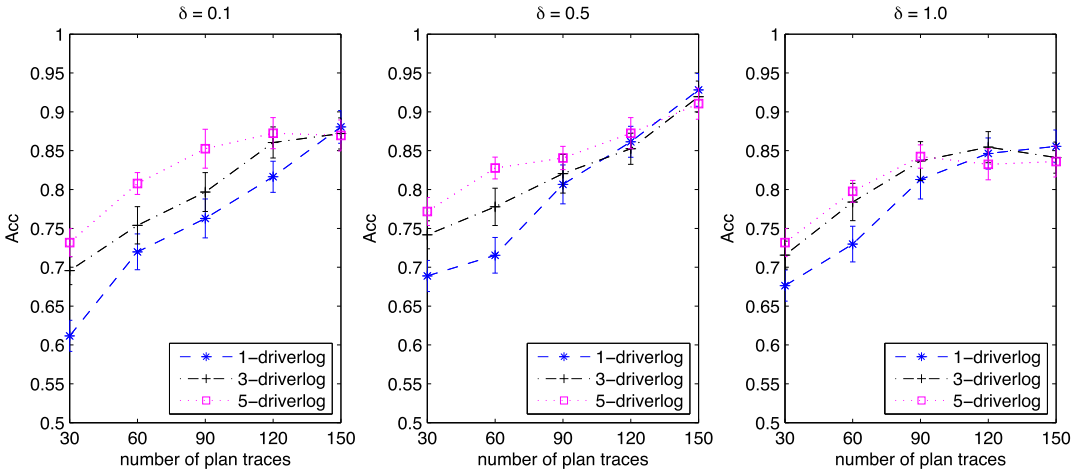

Fig. 12. Transfer learning action models of *driverlog*.

as the number of plan traces increases, the benefit of transfer learning is getting small. This is because, when the number of plan traces is large enough, the training information from the target domain is rich enough to learn action models well, and the impact of knowledge transferred from other domains are not as important as before. This is consistent with our intuition. By comparing different δ, we find that the accuracies are generally higher than other cases when δ is 0.5. This fact can be found from both Figs. 11 and 12, and it suggests that 0.5 could be chosen as the threshold in our algorithm TRAMP.

4.2.5. Relations between source and target domains

It is possible that a source domain plays a negative effect on the learning result. As is mentioned in the end of our algorithm description, WPLL could be used as a metric to measure the "degree" of a set of formulas being satisfied. We run our algorithm TRAMP to learn action models of *driverlog* with/without transferring knowledge from *blocks*, *depots* and *zeno-travel* respectively, by setting $\delta = 0.5$. Note that we only show the result on three source domains. Similar results can also be attained when we test on all the source domains including *laundry* and *dishwashing*. The results are shown in the first row of Fig. 13, where the curve denoted by "without transfer" is the learning result without transfer from any source domain, and the curve denoted by "from blocks" is the learning result with transfer from the domain *blocks*, likewise for the curves denoted by "from depots" and "from zeno-travel". We also record all the WPLL* calculated by Algorithm 3 when transfer learning from *blocks*, *depots* and *zeno-travel*. The results are shown in the second row of Fig. 13. From the first and third columns, we find that the learning results are better with transfer from *blocks* or *zeno-travel* than without transfer, meanwhile, their corresponding WPLL*s are all larger than -1. Furthermore, the larger the WPLL* is, the more improvement will be taken by transfer (can be seen from the third column of Fig. 13 when the number of plan traces is 40). This is consistent with our claim that WPLL could be used as a metric. On the other hand, in the second column, we find that transfer learning from *depots* will cause larger error rate than without transfer, and their WPLL*s are smaller than -1. This

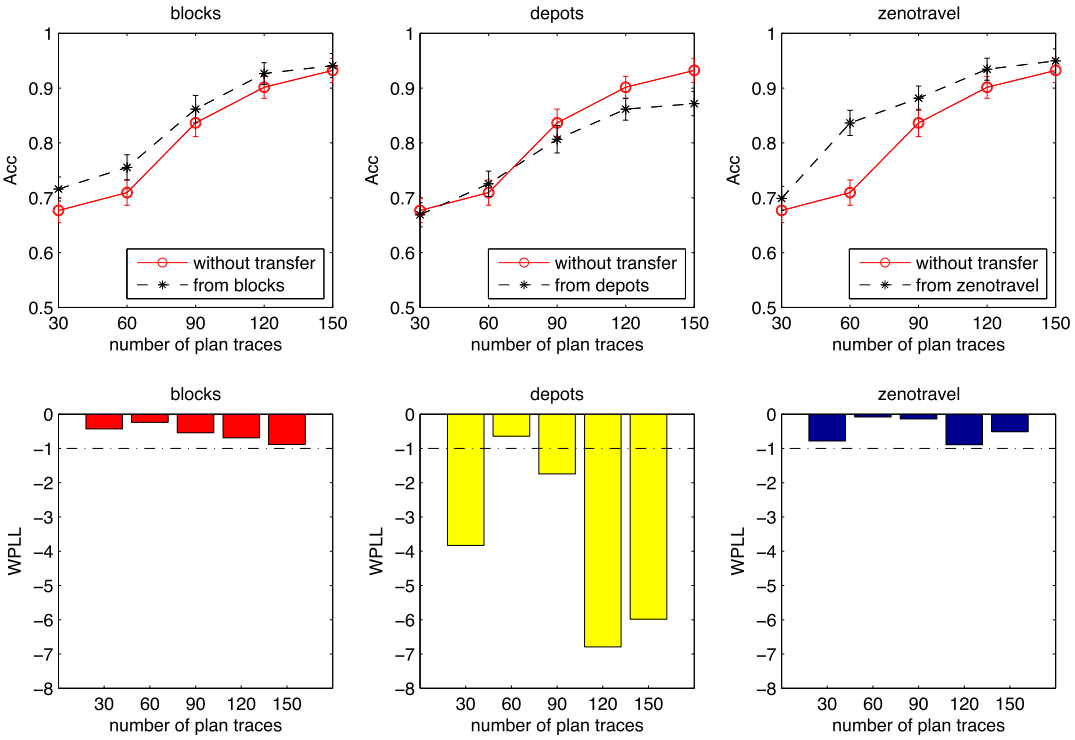

Fig. 13. Learning action models of *driverlog* with/without transfer.

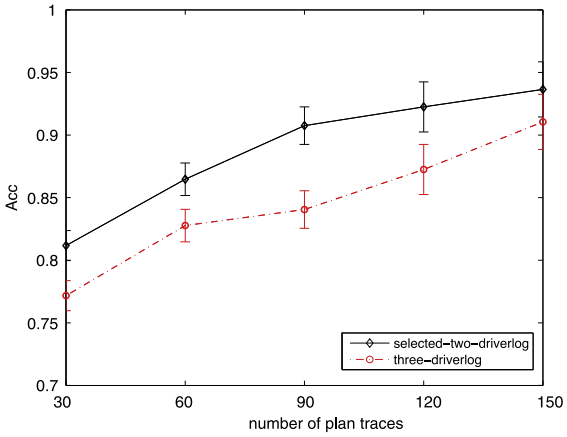

Fig. 14. Learning action models of *driverlog* by transferring knowledge from two and three other domains, respectively.

fact suggests that a domain, e.g. *depots*, provides negative effects on the learning result when its corresponding WPLL* is small; while a domain, e.g., *blocks*, provides positive effects when its corresponding WPLL* reaches some threshold, e.g., −1.

We would like to see what benefit we can get by using WPLL* as a metric to filter source domains. When we exploit −1 as the threshold, *blocks* and *zeno-travel* will be selected based on Fig. 13 as source domains to learn action models of *driverlog*. The learning result is shown in Fig. 14, where the curve denoted by "three-driverlog" is the learning result with transfer from *blocks*, *depots* and *zeno-travel*, and the curve denoted by "selected-two-driverlog" is the learning result with transfer from *blocks* and *zeno-travel*. From Fig. 14, we can see that the learning result using filtering is much better than not using it. Note that the curve denoted by "2-driverlog" in Fig. 12 is the average result by transferring knowledge from randomly selected two source domains from among *blocks*, *depots* and *zeno-travel*. Thus, it is different from the curve denoted by "selected-two-driverlog".

4.2.6. Running time

In Table 2, we show the running time of TRAMP for learning action models of the *driverlog* domain. We find that the running time (seconds) generally increases polynomially with respect to the number of plan traces, which can be seen from

Table 2
CPU times (seconds) of TRAMP for learning action models in the *driverlog* domain, where column 1 is the number of plan traces.

# plan traces	1-driverlog	3-driverlog	5-driverlog
30	133	245	318
60	882	963	1092
90	1345	1432	1549
120	1532	1615	1698
150	1613	1674	1833

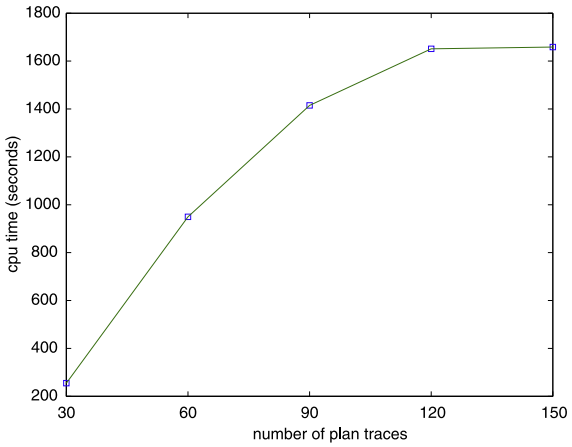

Fig. 15. The CPU times of TRAMP for learning action models of *driverlog* by transferring from three source domains.

the second column of Table 2. This fact can be found in Fig. 15, which is a fitting curve of the running time of *3-driverlog*. Notice that we did not show other curves which can be fitted similarly, since they suggest the property similar to Fig. 15. Specifically, the curve in Fig. 15 is the result that can fit a polynomial as $-0.1273 * x^2 + 34.6143 * x - 669.2000$, where x is the number of plan traces.

5. Related work

Our approach is related to previous work in domain model acquisition, transfer learning, Markov Logic Networks, and Web search, which will be reviewed in the following subsections, respectively.

5.1. Domain model acquisition

Our work is first related to domain model acquisition in planning communities. There have been many researches in acquisition domain models, which can be categorized by three classes, i.e., *deterministic action-model acquisition*, *probabilistic action-model acquisition*, and *Hierarchical Task Network (HTN) [45] domain acquisition*.

In *deterministic action-model acquisition*, Gil described a system called EXPO, bootstrapped by an incomplete STRIPS-like domain description which was augmented through experience [22]. Wang proposed an approach to automatically learn planning operators by observing expert solution traces and refining the operators through practice in a learning-by-doing paradigm [61]. Holmes and Isbell, Jr. modeled synthetic items based on experience to construct action models [26]. Yang et al. presented the ARMS system (action-relation modeling system) [67] for automatically discovering STRIPS [20] 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 [5,41]. Amir [1] presented a tractable and exact technique for learning action models known as Simultaneous Learning and Filtering, where the state observations were needed for learning. Walsh and Littman presented an efficient algorithm for learning action schemas for describing Web services [59]. Cresswell et al. (2009) [13] established a system called *LOCM* designed to carry out automated induction of action models from sets of example plans. Compared with previous systems, *LOCM* can learn action models with action sequences as input, and is shown to work well under the assumption that the output domain model can be represented in an object-centered representation [13]. Zhuo et al. [72] proposed an algorithm called LAMP to learn complex action models with quantifiers and implications using Markov Logic Networks (MLNs) [53]. Mourao et al. [44] presented an approach to learn STRIPS action models from noisy, incomplete observations.

In *probabilistic action-model acquisition*, Zettlemoyer et al. [68] investigated into learning a model of the effects of actions in noisy stochastic worlds. Chrisman [12] demonstrated how to learn stochastic actions with no conditional effects. Benson

[4] designed a TRAIL system to learn action models from domains that contained continuous variables, nondeterministic action effects. Pasula et al. [48] developed an algorithm to learn probabilistic relational planning operators, but not allowing noises in environments. Furthermore, in [49], a probabilistic, relational planning rule representation can be learned which could compactly model noisy, nondeterministic action effects. Kwiatkowski et al. [33] proposed to incrementally learn probabilistic models of structure and semantics form a corpus of child-directed utterances paired with possible representations of their meanings.

In *HTN domain acquisition*, Zhuo et al. [71] presented to learn action models and Hierarchical Task Network method preconditions simultaneously. Ilghami et al. and Xu and Munoz-Avila [27,66] proposed eager and lazy learning algorithms respectively to learn the preconditions of HTN methods, given as input the hierarchical relationships between tasks, the action models, and a complete description of the intermediate states. Nejati et al. [46] used means-end analysis to learn structures and preconditions of the input plans, assuming that a model of the tasks in the form of Horn clauses was given. Hogg et al. [25] developed an algorithm, called HTN-MAKER, to learn structures by assuming that the model of a task was given in the form of preconditions and effects. All the algorithms developed did not address the problem of learning method preconditions, method structures and action models altogether. There are also some algorithms designed to learn HTNs in nondeterministic planning domains where actions might have multiple outcomes [23], or learn probabilistic HTNs that capture user preferences by examining user-produced plans [39]. Hogg et al. [24] presented an algorithm that integrated HTN learning with Reinforcement Learning to learn HTN methods for planning.

Despite the success of these previous learning systems, they all learnt action models in one domain with the assumption that there are large enough plan traces available in the task domain, while in this paper, we aim to acquire action models across domains.

5.2. Transfer learning

Transfer learning has gained more and more attention in the last two decades due to the development of algorithms that can successfully generalize information across multiple tasks for classification, regression, clustering, reinforcement learning (RL), statistical relational learning (SRL) [47,11,57,52,56]. In the following, we introduce transfer learning in the following three aspects: *data mining based transfer learning*, *RL based transfer learning*, and *SRL based transfer learning*.

In *data mining based transfer learning*, there are mainly three different subsettings, i.e., transferring instances, transferring feature representations and transferring parameters. *Transferring instances* assumes there are certain parts of the data in the source domain can be reused together with a few labeled data in the target domain. There have been lots of work on this type of transferring. For example, Dai et al. [14] presented a boosting algorithm, called *TrAdaBoost*, to address the instance-transferring problems. Jiang and Zhai [28] proposed a heuristic method to remove "misleading" training examples from the source domain based on the difference between conditional probabilities. Liao et al. [40] presented a new active learning algorithm to select the unlabeled data in a target domain to be labeled with the help of the source domain data. Wu and Dietterich [62] integrated the source domain data in an SVM framework for improving the classification performance. *Transferring feature representations* aims to find better feature representations to minimize domain divergence and classification or regression model error. In this setting, Argyriou et al. [2] described a sparse feature learning method for multi-task learning, and proposed a spectral regularization framework on matrices for multi-task structure learning [3]. Lee et al. [37] presented an optimization algorithm for learning meta-priors and feature weights from related prediction tasks. *Transferring parameters* assume that individual models for related tasks should share some parameters or prior distributions of hyper-parameters. For example, Lawrence et al. [36,7] presented an efficient algorithm MT-IVM based on Gaussian Processes to handle the multi-task learning problems. Schwaighofer et al. [54] proposed to exploit hierarchical Bayesian framework and Gaussian Processes for multi-task learning.

In *reinforcement learning based transfer learning*, Taylor and Stone introduced the problem of reusing agents' knowledge so that it can be used to solve a different task, i.e., to transfer knowledge between agents solving different tasks in RL domains [52,56,58,9]. They introduced what was possible with transfer in RL domains, e.g., there are a number of different possible goals for transfer in RL; transfer methods can differ in terms of what RL methods they are compatible with; different transfer learning methods transfer different types of knowledge, i.e., what is transferred between the source and target tasks.

In *statistical relational learning based transfer learning*, which deals with transfer learning problems in relational domains, where the data are not i.i.d. and can be represented by multiple relations, such as first order logic formulae, Mihalkova et al. [42] presented a TAMAR algorithm to transfer relational knowledge across relational domains based on Markov Logic Networks (MLNs) [53]. MLNs, which combines the compact expressiveness of first order logic with flexibility of probability, is a powerful formalism. Furthermore, Mihalkova and Mooney [43] assumed the amount of target data was minimal and consisted of information about just a handful of entities. They proposed an algorithm to handle this case by transferring models in source domains to the target domains. Davis and Domingos [15] proposed an approach to discover structural regularities in the source domain based on second order Markov logic. Deshpande et al. [16] proposed to leverage data from multiple source tasks and a limited set of examples from a target task to learn the rule set for the target task with the greatest posterior probability. Different from *data mining based*, *reinforcement learning based* and *statistical relational learning based transfer learning*, we aim to explore transfer learning for action model acquisition for planning.

Könik et al. proposed an algorithm called GAMA [31] to transfer skills of solving problems from a source domain to solving new problems from the target domain in the context of ICARUS [35]. They assume the source and target domain

theories are both provided as input and the mappings between source and target domains are evaluated by both domain theories, which is different from our approach TRAMP which aims to learn the target domain theory. Adapting GAMA to learning domain theories is difficult since it requires both source and target domain theories as input for building transfer between the source and target, while in our setting we do not have such target domain theory to build transfer that way. To the best of our knowledge, our previous approaches, t-LAMP [73] (which transfers knowledge from source domains based on plan traces) and LAWS [70] (which transfers knowledge from source domains based on Web search), are the first transfer learning approaches in learning action models in planning community. We will compare the proposed TRAMP approach with t-LAMP and LAWS in the experiment.

5.3. Markov Logic Networks

Markov Logic Network (MLN) [53] is a powerful framework that combines probability and first-order logic with statistical learning. We will exploit MLN in our algorithm to help transfer learning. A Markov logic network (or MLN) is a probabilistic logic that can conduct both learning and inference based on a Markov network. It is expressive enough to allow first order logic. It includes a set of weighted first-order logic formulas that forms a network. In this network, the vertices are atomic formulas, and the edges are the logical connectives used to construct the formula. Taking the facts and constraints as input, it exploits an optimization algorithm to compute the weights of the output formulas.

An advantage of MLNs is in its ability to "soften" the constraints; e.g., when a world violates a formula in a knowledge base, it is less probable, but not impossible. MLNs have been applied to several real world applications with great success. In this paper, we propose to use Weighted Pseudo-Log-Likelihood (WPLL) [53] of MLN as our transfer learning metric to be optimized (we give a formal definition in Algorithms 1), and calculate this metric by the Alchemy system [30]. For instance, Domingos [17] proposes to apply Markov logic to model real social networks, which evolve in time with multiple types of arcs and nodes and are affected by the actions of multiple players; Poon and Domingos [51] propose a joint approach to perform information extraction using Markov logic and existing algorithms, where segmentation of all records and entity resolution are performed together in a single integrated inference process; etc.

5.4. Web search

Web search engines try to understand user needs by retrieving information from the WWW, whose results are generally referred to as search engine results pages (SERPs), which are presented in a line of results. The information may be described by Web pages, images, and other types of files. Web search also mines data available in databases or open directories, and maintain real-time information by running an algorithm on a Web crawler. When a user enters a query into a search engine (typically by using keywords), the engine examines its index and provides a listing of best-matching Web pages according to its criteria. Andrei Broder classified Web search according to their intent into three classes [10]:

- *Navigational*: the immediate intent is to reach a particular site. The purpose of this query is to reach a particular site that the user has in mind, e.g., searching "compaq" probably targets at "http://www.compaq.com".
- *Informational*: the intent is to acquire some information assumed to be present on one or more Web pages. The purpose of such queries is to find information assumed to be available on the Web in a static form, i.e., no further interaction is predicted, except reading.
- *Transactional*: the intent is to perform some Web-mediated activity. The purpose of such queries is to reach a site where further interaction will happen, e.g., shopping, downloading images, etc.

Our Web search exploited in this paper is related to the second class, i.e., *informational Web search*. In the past, there have been some previous research work that had considered learning common sense knowledge from Web search, such as ehow and KnowItAll [18,55]. Specifically, Perkowitz et al. proposed to mine the natural language descriptions of activities, e.g., making-tea, from ehow.com as labeled data, and translated them into probabilistic collections of object terms, e.g., teacup, teabag, etc. [50]. After that, they used these probabilistic collections as input data to train a dynamic Bayesian network model for prediction of activities. Wyatt et al. also proposed to mine recipes of the activities from the Web as the labeled data, but they only used this knowledge as a prior and trained a Hidden Markov Model from the unlabeled RFID sensor data [63]. Wang et al. further improved this work by utilizing personal activity data from wearable sensors [60]. They first extracted the actions from the wearable sensors, and then incorporated the actions with the object usage to finally predict the activities. Bollegala et al. [6] proposed a relational similarity measure, using a Web search engine, to compute the similarity between semantic relations implied by two pairs of words. Zheng et al. [69] developed a mapping between activities in two domains by learning a similarity function via Web search. Xiang et al. [64] proposed BIG (Bridging Information Gap) approach to extract useful knowledge in a worldwide knowledge base and use this knowledge to link the source and target domains for improving the classification performance. BIG works when the source and target domains share the same feature space but different underlying data distributions. Inspired by the success of exploiting Web search for measuring *keywords* similarities, we aim to acquire action models across domains with the help of Web search in planning communities.

6. Final remarks

We explore a novel transfer learning approach TRAMP to learning action models for planning in the STRIPS representation, when there are only a few limited plan traces or training data available in target domains. TRAMP exploits two phases to transfer knowledge from source domains to target domains. The first phase is to transfer knowledge from source based on plan traces in target domains. The second phase is to explore extra knowledge from the Web and as thus transfer knowledge from source domains based on Web search. We compare TRAMP to LAWS and t-LAMP, to see the advantage or disadvantage of using the first or second or both phases in the experiments. We also compare our transfer learning framework TRAMP to a state-of-the-art non-transfer learning approach ARMS, showing that TRAMP functions well even though there is no source domain available. We also empirically show that the similarity between source and target domains can be captured by the metric WPLL and using WPLL to filter source domains can indeed improve the transfer learning accuracies.

In this paper, we assume that source domains have complete action models and the task is to learn action models for target domains. In the future, we would like to consider the case that when source domains are incomplete and target domains are annotated with a few incomplete model descriptions, the task is to learn action models for both source and target domains. This is significant since it is easier to collect incomplete domain models for transfer in real world applications, compared to collecting complete models. In addition, we would like to consider more elaborate representations in PDDL language and test the system in more real world planning domains.

Acknowledgements

We thank the support of China National 973 project 2014CB340304, National Natural Science Foundation of China (No. 61309011), Fundamental Research Funds for the Central Universities (No. 14lgzd06), Research Fund for the Doctoral Program of Higher Education of China (No. 20110171120054), Guangzhou Science and Technology Project (No. 2011J4300039), and Hong Kong RGC Projects 621013, 620812, and 621211.

References

[1] E. Amir, Learning partially observable deterministic action models, in: Proceedings of IJCAI, 2005, pp. 1433–1439.
[2] A. Argyriou, T. Evgeniou, M. Pontil, Multi-task feature learning, in: Proceedings of NIPS, 2007, pp. 41–48.
[3] A. Argyriou, C.A. Micchelli, M. Pontil, Y. Ying, A spectral regularization framework for multi-task structure learning, in: Proceedings of NIPS, 2008, pp. 25–32.
[4] S.S. Benson, Learning STRIPS operators from noisy and incomplete observations, Ph.D. thesis, Stanford, 1996.
[5] J. Blythe, J. Kim, S. Ramachandran, Y. Gil, An integrated environment for knowledge acquisition, in: Proceedings of IUI, 2001, pp. 13–20.
[6] D. Bollegala, Y. Matsuo, M. Ishizuka, Measuring the similarity between implicit semantic relations from the web, in: Proceedings of WWW, 2009.
[7] E.V. Bonilla, K.M.A. Chai, C.K.I. Williams, Multi-task Gaussian process prediction, in: Proceedings of NIPS, 2008, pp. 153–160.
[8] K. Borgwardt, A. Gretton, M. Rasch, H. Kriegel, B. Scholkopf, A. Smola, Integrating structured biological data by kernel maximum mean discrepancy, in: Proceedings of ISMB, 2006, pp. 49–57.
[9] H. Bou-Ammar, K. Tuyls, M.E. Taylor, K. Driessens, G. Weiss, Reinforcement learning transfer via sparse coding, in: AAMAS, 2012, pp. 383–390.
[10] A. Broder, A taxonomy of web search, SIGIR Forum 36 (2) (2002) 3–10.
[11] R. Caruana, Multitask learning, Mach. Learn. 28 (1) (1997) 41–75.
[12] L. Chrisman, Abstract probabilistic modeling of action, in: Proceedings of the First International Conference on Artificial Intelligence Planning Systems (AIPS-92), 1992, pp. 28–36.
[13] S. Cresswell, T.L. McCluskey, M.M. West, Acquisition of object-centred domain models from planning examples, in: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS'09), 2009.
[14] W. Dai, Q. Yang, G.-R. Xue, Y. Yu, Boosting for transfer learning, in: Proceedings of ICML, 2007, pp. 193–200.
[15] J. Davis, P. Domingos, Deep transfer via second-order Markov logic, in: Proceedings of ICML, 2009, pp. 217–224.
[16] A. Deshpande, B. Milch, L.S. Zettlemoyer, L.P. Kaelbling, Learning probabilistic relational dynamics for multiple tasks, in: Proceedings of UAI, 2007, pp. 83–92.
[17] P. Domingos, Mining social networks for viral marketing, IEEE Intell. Syst. 20 (1) (2005) 80–82.
[18] O. Etzioni, M.J. Cafarella, D. Downey, A.M. Popescu, T. Shaked, S. Soderland, D.S. Weld, A. Yates, Methods for domain-independent information extraction from the web: an experimental comparison, in: Proceedings of AAAI, 2004, pp. 391–398.
[19] B. Falkenhainer, K.D. Forbus, D. Gentner, The structure-mapping engine: algorithm and examples, Artif. Intell. 41 (1) (1989) 1–63.
[20] R. Fikes, N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artif. Intell. J. (1971) 189–208.
[21] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.
[22] Y. Gil, Learning by experimentation: incremental refinement of incomplete planning domains, in: Proceedings of the Eleventh International Conference on Machine Learning (ICML-94), 1994, pp. 87–95.
[23] C. Hogg, U. Kuter, H. Muñoz-Avila, Learning hierarchical task networks for nondeterministic planning domains, in: Proceedings of IJCAI, 2009, pp. 1708–1714.
[24] C. Hogg, U. Kuter, H. Munoz-Avila, Learning methods to generate good plans: integrating HTN learning and reinforcement learning, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), 2010.
[25] C. Hogg, H. Muñoz-Avila, U. Kuter, HTN-MAKER: learning HTNs with minimal additional knowledge engineering required, in: Proceedings of AAAI, 2008, pp. 950–956.
[26] M.P. Holmes, C.L. Isbell Jr., Schema learning: experience-based construction of predictive action models, in: Advances in Neural Information Processing Systems 17 (NIPS-04), 2004.
[27] O. Ilghami, H. Muñoz-Avila, D.S. Nau, D.W. Aha, Learning approximate preconditions for methods in hierarchical plans, in: Proceedings of ICML, 2005, pp. 337–344.
[28] J. Jiang, C. Zhai, Instance weighting for domain adaptation in NLP, in: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Association for Computational Linguistics, Prague, Czech Republic, June 2007, pp. 264–271.
[29] K.S. Jones, A statistical interpretation of term specificity and its application in retrieval, J. Doc. 28 (1972) 11–21.

[30] S. Kok, P. Singla, M. Richardson, P. Domingos, The alchemy system for statistical relational AI, 2005.
[31] T. Könik, P. O'Rorke, D.G. Shapiro, D. Choi, N. Nejati, P. Langley, Skill transfer through goal-driven representation mapping, Cogn. Syst. Res. 10 (3) (2009) 270–285.
[32] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1) (1951) 79–86.
[33] T. Kwiatkowski, S. Goldwater, L.S. Zettlemoyer, M. Steedman, A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings, in: Proceedings of EACL, 2012, pp. 234–244.
[34] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, Z. Ghahramani, Sigma: simple greedy matching for aligning large knowledge bases, in: Proceedings of KDD, 2013, pp. 572–580.
[35] P. Langley, D. Choi, Learning recursive control programs from problem solving, J. Mach. Learn. Res. 7 (2006) 493–518.
[36] N.D. Lawrence, J.C. Platt, Learning to learn with the informative vector machine, in: Proceedings of ICML, 2004.
[37] S.-I. Lee, V. Chatalbashev, D. Vickrey, D. Koller, Learning a meta-level prior for feature relevance from multiple related tasks, in: Proceedings of ICML, 2007, pp. 489–496.
[38] H.J. Levesque, F. Pirri, R. Reiter, Foundations for the situation calculus, Electron. Trans. on Artif. Intell. 2 (1998) 159–178.
[39] N. Li, S. Kambhampati, S.W. Yoon, Learning probabilistic hierarchical task networks to capture user preferences, in: Proceedings of IJCAI, 2009, pp. 754–759.
[40] X. Liao, Y. Xue, L. Carin, Logistic regression with an auxiliary data source, in: Proceedings of ICML, 2005, pp. 505–512.
[41] T.L. McCluskey, D. Liu, R.M. Simpson, GIPO II: HTN planning in a tool-supported knowledge engineering environment, in: Proceedings of ICAPS, 2003, pp. 92–101.
[42] L. Mihalkova, T. Huynh, R.J. Mooney, Mapping and revising Markov logic networks for transfer learning, in: Proceedings of AAAI, 2007, pp. 608–614.
[43] L. Mihalkova, R.J. Mooney, Transfer learning from minimal target data by mapping across relational domains, in: Proceedings of IJCAI, 2009, pp. 1163–1168.
[44] K. Mourao, L.S. Zettlemoyer, R.P.A. Petrick, M. Steedman, Learning strips operators from noisy and incomplete observations, in: Proceedings of UAI, 2012.
[45] D.S. Nau, Y. Cao, A. Lotem, H. Muñoz-Avila, SHOP: simple hierarchical ordered planner, in: Proceedings of IJCAI, 1999, pp. 968–973.
[46] N. Nejati, P. Langley, T. Konik, Learning hierarchical task networks by observation, in: Proceedings of ICML, 2006, pp. 665–672.
[47] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[48] H.M. Pasula, L.S. Zettlemoyer, L.P. Kaelbling, Learning probabilistic relational planning rules, in: Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04), 2004, pp. 73–82.
[49] H.M. Pasula, L.S. Zettlemoyer, L.P. Kaelbling, Learning symbolic models of stochastic domains, J. Artif. Intell. Res. 29 (2007) 309–352.
[50] M. Perkowitz, M. Philipose, K. Fishkin, D. Patterson, Mining models of human activities from the web, in: Proceedings of WWW, 2004, pp. 573–582.
[51] H. Poon, P. Domingos, Joint inference in information extraction, in: Proceedings of AAAI'07, 2007, pp. 913–918.
[52] J. Ramon, K. Driessens, T. Croonenborghs, Transfer learning in reinforcement learning problems through partial policy recycling, in: Proceedings of ECML, 2007, pp. 699–707.
[53] M. Richardson, P. Domingos, Markov logic networks, Mach. Learn. 1–2 (62) (2006) 107–136.
[54] A. Schwaighofer, V. Tresp, K. Yu, Learning Gaussian process kernels via hierarchical bayes, in: Proceedings of NIPS, 2005, pp. 1209–1216.
[55] E.M. Tapia, T. Choudhury, M. Philipose, Building reliable activity models using hierarchical shrinkage and mined ontology, in: Proceedings of Pervasive, 2006, pp. 17–32.
[56] M.E. Taylor, P. Stone, Cross-domain transfer for reinforcement learning, in: Proceedings of ICML, June 2007.
[57] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains, J. Mach. Learn. Res. 10 (1) (2009) 1633–1685.
[58] M.E. Taylor, P. Stone, An introduction to inter-task transfer for reinforcement learning, AI Mag. 32 (1) (2011) 15–34.
[59] T.J. Walsh, M.L. Littman, Efficient learning of action schemas and web-service descriptions, in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08), 2008, pp. 714–719.
[60] S. Wang, W. Pentney, A.-M. Popescu, T. Choudhury, M. Philipose, Common sense based joint training of human activity recognizers, in: Proceedings of IJCAI, 2007, pp. 2237–2242.
[61] X. Wang, Learning by observation and practice: an incremental approach for planning operator acquisition, in: Proceedings of the Twelfth International Conference on Machine Learning (ICML-95), 1995, pp. 549–557.
[62] P. Wu, T.G. Dietterich, Improving SVM accuracy by training on auxiliary data sources, in: Proceedings of ICML, 2004, pp. 1–8.
[63] D. Wyatt, M. Philipose, T. Choudhury, Unsupervised activity recognition using automatically mined common sense, in: Proceedings of AAAI, 2005, pp. 21–27.
[64] E.W. Xiang, B. Cao, D.H. Hu, Q. Yang, Bridging domains using world wide knowledge for transfer learning, IEEE Trans. Knowl. Data Eng. 22 (6) (2010) 770–783.
[65] J.Z. Xu, J.E. Laird, Combining learned discrete and continuous action models, in: Proceedings of AAAI, 2011, pp. 1449–1454.
[66] K. Xu, H. Muñoz-Avila, A domain-independent system for case-based task decomposition without domain theories, in: Proceedings of AAAI, 2005, pp. 234–240.
[67] Q. Yang, K. Wu, Y. Jiang, Learning action models from plan examples using weighted MAX-SAT, Artif. Intell. J. 171 (February 2007) 107–143.
[68] L.S. Zettlemoyer, H.M. Pasula, L.P. Kaelbling, Learning planning rules in noisy stochastic worlds, in: Proceedings of AAAI, 2005.
[69] V.W. Zheng, D.H. Hu, Q. Yang, Cross-domain activity recognition, in: Proceedings of UbiComp, 2009, pp. 61–70.
[70] H. Zhuo, Q. Yang, L. Li, Transfer learning action models by measuring the similarity of different domains, in: Proceedings of PAKDD, 2009.
[71] H.H. Zhuo, D.H. Hu, C. Hogg, Q. Yang, H. Munoz-Avila, Learning HTN method preconditions and action models from partial observations, in: Proceedings of IJCAI, 2009, pp. 1804–1810.
[72] H.H. Zhuo, Q. Yang, D.H. Hu, L. Li, Learning complex action models with quantifiers and logical implications, Artif. Intell. 174 (18) (2010) 1540–1569.
[73] H.H. Zhuo, Q. Yang, R. Pan, L. Li, Cross-domain action-model acquisition for planning via web search, in: Proceedings of ICAPS, 2011, pp. 298–305.