Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

# Crowdsourced Action-Model Acquisition for Planning

**Hankz Hankui Zhuo**
School of Advanced Computing,
Sun Yat-sen University, Guangzhou, China. 510006
zhuohank@mail.sysu.edu.cn

## Abstract

AI planning techniques often require a given set of action models provided as input. Creating action models is, however, a difficult task that costs much manual effort. The problem of action-model acquisition has drawn a lot of interest from researchers in the past. Despite the success of the previous systems, they are all based on the assumption that there are enough training examples for learning high-quality action models. In many real-world applications, e.g., military operation, collecting a large amount of training examples is often both difficult and costly. Instead of collecting training examples, we assume there are abundant annotators, i.e., the crowd, available to provide information learning action models. Specifically, we first build a set of soft constraints based on the labels (true or false) given by the crowd or annotators. We then builds a set of soft constraints based on the input plan traces. After that we put all the constraints together and solve them using a weighted MAX-SAT solver, and convert the solution of the solver to action models. We finally exhibit that our approach is effective in the experiment.

## Introduction

AI planning techniques often require a given set of action models as input. Creating action models, however, is a difficult task that costs much manual effort. The problem of action-model acquisition has drawn a lot of interest from researchers in the past. For instance, McCluskey et al. (McCluskey, Liu, and Simpson 2003) designed a system to interact with human experts to generate action models. Amir (Amir 2005) introduced a tractable and exact technique for learning action models known as Simultaneous Learning and Filtering, where the state observations were needed for learning. Yang et al. (Yang, Wu, and Jiang 2007) presented a framework for discovering STRIPS (Fikes and Nilsson 1971) action models from a set of successfully observed plans, just to name a few. Despite the success of 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 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), since there are 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 challenging when there is only a limited number of plan traces available (Zhuo and Yang 2014).

Inspired by crowdsourcing (Howe 2008), we are interested in considering integrating intelligence of crowds into action-model acquisition for planning, assuming the crowd are able to provide knowledge to help learn action models. There have been successful general-purpose platforms such as Amazons Mechanical Turk[1], task-specific sites for call centers[2] and programming jobs[3] to do crowdsourcing. Crowdsourcing has in fact been heavily used to solve various tasks, such as audio transcription (Liem, Zhang, and Chen 2011), hand-written recognition (Ouyang and Li 2012), story generation (Li et al. 2013), labeling training data for machine learning (Raykar et al. 2010), and crowdsourcing itinerary planning (Zhang et al. 2012). Rare work, however, is proposed to explore crowdsourcing to help learn action models in planning community.

In this paper, we propose an algorithm called `CAMA` (which stands for **C**rowdsourced **A**ction-**M**odel **A**cquisition), to explore knowledge from crowds to learn action models for planning. Specifically, `CAMA` functions by two phases. `CAMA` first builds a set of soft constraints based on the labels (true or false) given by the crowd or annotators. `CAMA` then builds a set of soft constraints based on the input plan traces. After that `CAMA` puts all the constraints together and solves them using a weighted MAX-SAT solver (Borchers and Furman 1998; Davies and Bacchus 2011), and converts the solution of the solver to action models.

Our approach explores knowledge from both crowdsourcing and plan traces. This is superior to solely harness crowdsourcing since the crowd might give erroneous or outright malicious responses, and these responses may be "filtered" by plan traces.



[1] http://mturk.com
[2] http://liveops.com
[3] http://topcoder.com

In the following, we first review previous work related to our paper in the next section, and formulate the planning problem. After that we present our CAMA algorithm in detail and evaluate CAMA in three planning domains. In the end, we conclude our paper and address future work.

## Related Work

Our work is first related to action model learning. Except the related work addressed in the introduction section, Holmes and Isbell, Jr. modeled synthetic items based on experience to construct action models (Holmes and Isbell, Jr. 2004). Cresswell et al. (Cresswell, McCluskey, and West 2009) established a system called *LOCM* designed to carry out automated induction of action models from sets of example plans. Zhuo et al. (Zhuo et al. 2010; Zhuo, Muñoz-Avila, and Yang 2014) proposed algorithms to learn complex action models and hierarchical task network domains from plan traces. Hoffmann et al. (Hoffmann, Weber, and Kraft 2010) proposed to derive domain models from software engineering.

Crowdsourcing is first used to improve label quality of training data in data mining community (Dawid and Skene 1979; Romney, Weller, and Batchelder 1986). For example, Raykar et al. (Raykar et al. 2010) proposed a model in which the parameters for worker accuracy depend on the true answer. Whitehill et al. (Whitehill et al. 2009) address the concern that worker labels should not be modeled as independent of each other. Welinder et al. (Welinder et al. 2010) design a multidimensional model for workers that takes into account competence, expertise, and annotator bias. Kamar et al. (Kamar, Hacker, and Horvitz 2012) extract features from the task at hand and use Bayesian Structure Learning to learn the worker response model. Kajino et al. (Kajino, Tsuboi, and Kashima 2012) formulate the repeated labeling problem as a convex optimization problem. Dai et al. (Dai et al. 2013) designed AI agents to obtain cost-quality tradeoff based on Bayesian learning and inference. Talamadupula et al. (Talamadupula et al. 2013; Manikonda et al. 2014) presented a general framework that foregrounds the potential roles of automated planners in crowd-planning. Cenamor et al. (Cenamor, de la Rosa, and Borrajo 2013) proposed an initial idea of attaining information from traveling social network to build tourist plans. Though there has been a large amount of work proposed to explore crowdsourcing in different areas, there is rare work conducted on learning action models for planning.

## 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 work as $\Sigma = (S, A, \Gamma)$, where $S$ is the set of states, $A$ is the set of action models, $\Gamma$ is the deterministic transition function $S \times \mathcal{A} \rightarrow S$. 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 $g$. Each $a_i$ is an *action schema* composed of an action name and 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 $s_1, ..., s_n$ are *partially* observed intermediate states. A state is called "partial" because some propositions in the state are missing. "partial" suggests "empty", when all propositions are missing.

We define our learning problem as: given a set of action schemas, a set of predicates, a small set of plan traces $\mathcal{T}$, how do we construct a set of action models? An example input of our learning problem in the *blocks* domain[4] is shown in Table 1. The output of our problem is a set of action models as shown below.

**(pickup ?x - block)**
(:precondition (ontable ?x) (clear ?x) (handempty))
(:effect (handempty) (not (clear ?x)) (not (ontable ?x))
(holding ?x) (not (handempty)))

The gray parts of the output action models are missing preconditions or effects of the learnt action models, e.g., *(ontable ?x)* is a precondition that should be learnt but is missing in the actual learnt model of *pickup*. The underlined parts are preconditions or effects that should not be learnt.

Table 1: Example input of our learning problem in *blocks*, where $s_0^1$={(ontable C) (on B C) (on A B) (clear A) (handempty)}, $g^1$={(on B A) (on C B)}, $s_0^2$={(ontable E) (on D E) (ontable F) (clear D) (clear F) (handempty)}, $g^2$={(on F E) (on D F)}.

**Action schemas:**
(pickup ?x - block) (putdown ?x - block)
(unstack ?x ?y - block) (stack ?x ?y - block)
**Predicates:**
(ontable ?x - block) (on ?x ?y - block)
(clear ?x - block) (holding ?x) (handempty)
**Plan traces:**
#1: $s_0^1$ (unstack A B) (putdown A) (unstack B C)
(stack B A) (pickup C) (stack C B) $g^1$
#2: $s_0^2$ (unstack D E) (putdown D) (pickup F)
(stack F E) (pickup D) (stack D F) $g^2$
...

## Our CAMA Framework

In this section, we present our algorithm CAMA to learn action models. We show an overview of CAMA in Algorithm 1. We will give the detailed description of each step of Algorithm 1 in the following subsections.

### Build Constraints Based on Crowdsourcing

We first build a set of constraints to capture the knowledge from the crowd (i.e., Step 1 of Algorithm 1). To do this, we

[4] http://www.cs.toronto.edu/aips2000/aips-2000datafiles.tgz

**Algorithm 1** high-level CAMA algorithm

**input:** (1) an action schema set $A$, a predicate set $P$, a set of plan traces $\mathcal{T}$

**output:** A set of action models $\mathcal{A}$

1: build constraints based on crowdsourcing
2: build constraints based on plan traces $\mathcal{T}$
3: solve all the constraints
4: convert the solving result to $\mathcal{A}$
5: **return** $\mathcal{A}$

enumerate all possible preconditions and effects for each action. Specifically, we generate actions' preconditions and effects as follows. If the parameters of predicate $p$, denoted by Para($p$), are included by the parameters of action $a$, denoted by Para($a$), i.e., Para($p$) $\subseteq$ Para($a$), $p$ is likely a precondition, or an adding effect, or a deleting effect of $a$. We therefore generate three new proposition variables "$p \in$ Pre($a$)", "$p \in$ Add($a$)" and "$p \in$ Del($a$)", the set of which is denoted by $H_{pre} = \{p \in \text{Pre}(a) | \forall p \in P \text{ and } \forall a \in A\}$, $H_{add} = \{p \in \text{Add}(a) | \forall p \in P \text{ and } \forall a \in A\}$, $H_{del} = \{p \in \text{Del}(a) | \forall p \in P \text{ and } \forall a \in A\}$, respectively. We put all the proposition variables together, i.e., $H = H_{pre} \cup H_{add} \cup H_{del}$, and estimate the label (*true* or *false*) of each variable by querying the crowd or annotators.

For each proposition in $H$, we build a Human Intelligence Task (HIT) in the form of a short survey. For example, for the proposition "(ontable $?x$) $\in$ Pre(pickup)", we generate a survey as shown below:

*Is the fact that "x is on table" a precondition of picking up the block x?*

There are three possible responses, i.e., "Yes, No, Cannot tell", out of which an annotator has to choose exactly one. Each annotator is allowed to annotate a given survey once. Note that the set of proposition variables $H$ will not be large, since all predicates and actions are in the "variable" form rather than instantiated form, and we only consider predicates whose parameters are included by actions. For example, for the *blocks* domain shown in Table 1, there are only 78 proposition variables in $H$.

Assume there are $R$ annotators and $N$ tasks ($N = |H|$) with binary labels $\{1, 0\}$. Denote by $\mathcal{Z} = \{z_i \in \{1, 0\}, i \in [N]\}$ the true label of task $i$, where $[N]$ represents the set of first $N$ integers. Let $\mathcal{N}_j$ is the set of tasks labeled by annotator $j$, and $\mathcal{R}_i$ is the set of annotators labeling task $i$. The task assignment scheme can be represented by a bipartite graph where an edge $(i, j)$ denotes that the task $i$ is labeled by the worker $j$. The labeling results form a matrix $y \in \{1, 0\}^{N \times R}$. The goal is to find an optimal estimator $\hat{\mathcal{Z}}$ of the true labels $\mathcal{Z}$ given the observation $\mathcal{Y}$, minimizing the average bit-wise error rate $\frac{1}{N} \sum_{i \in [N]} prob[\hat{z}_i \neq z_i]$.

We model the accuracy of annotators separately on the positive and the negative examples (Raykar et al. 2010; Raykar and Yu 2011). If the true label is one, the true positive rate $TP^j$ for the $j$th annotator is defined as the probability that the annotator labels it as one, i.e.,

$$TP^j = p(y_i^j = 1 | z_i = 1).$$

On the other hand if the true label is zero, the true negative rate $TN^j$ is defined as the probability that annotator labels it as zero, i.e.,

$$TN^j = p(y_i^j = 0 | z_i = 0).$$

Considering the family of linear discriminating functions, the probability for the positive class is modeled as a logistic sigmoid, i.e.,

$$p(y = 1 | x, w) = \sigma(w^T x),$$

where $\sigma(z) = \frac{1}{(1+e^{-z})}$.

Let $x_i$ be a value assigned to variable $X_i \in \mathcal{X}$ and $y_i^j$ be the label (1 or 0) given by the $j$th annotator. We have the training data $D = \{x_i, y_i^1, \ldots, y_i^R\}_{i=1}^N$ with N instances from $R$ annotators.

The task is to estimate the parameter $w$ as well as the true positive $\mathcal{P} = \langle TP^1, \ldots, TP^R \rangle$ and the true negative $\mathcal{N} = \langle TN^1, \ldots, TN^R \rangle$. Let $\theta = \{w, \mathcal{P}, \mathcal{N}\}$, the probability of training data $D$ can be defined by

$$p(D|\theta) = \prod_{i=1}^{N} p(y_i^1, \ldots, y_i^R | x_i, \theta).$$

Given the true label $y_i$, we assume $y_i^1, \ldots, y_i^R$ are independent. We thus have

$$p(D|\theta) = \prod_{i=1}^{N} [a_i p_i + b_i (1 - p_i)],$$

where $p_i = \sigma(w^T x_i)$, $a_i = \prod_{j=1}^{R} [TP^j]^{y_i^j} [1 - TP^j]^{1 - y_i^j}$, and $b_i = \prod_{j=1}^{R} [TN^j]^{1 - y_i^j} [1 - TN^j]^{y_i^j}$.

The EM algorithm can be exploited to estimate the parameter $\theta$ by maximizing the log-likelihood of $p(D|\theta)$ (Raykar et al. 2010), i.e.,

$$\hat{\theta} = \arg\max_{\theta} \{\ln p(D|\theta)\},$$

where $\hat{\theta} = \{\hat{\mathcal{P}}, \hat{\mathcal{N}}, \hat{w}\}$. With $\hat{\theta}$ we can estimate the final label of $x_i$ according the probability $p(y_i = 1 | y_i^1, \ldots, y_i^R, x_i, \hat{\theta})$. We use the unknown hidden true label $z_i$ as the missing data, and denote $\mathcal{Z} = [z_1, \ldots, z_N]$. We have

$$\ln p(D, \mathcal{Z} | \theta) = \sum_{i=1}^{N} z_i \ln p_i a_i + (1 - z_i) \ln (1 - p_i) b_i.$$

As a result, the expectation can be state as

$$E\{\ln p(D, \mathcal{Z} | \theta)\} = \sum_{i=1}^{N} \mu_i \ln p_i a_i + (1 - \mu_i) \ln (1 - p_i) b_i,$$

where $\mu_i = p(z_i = 1 | y_i^1, \ldots, y_i^R, x_i, \theta)$. In this paper, however, we would like to trust a particular expert more than the others. One way to achieve this is to impose priors on $\mathcal{P}$ and $\mathcal{N}$. We choose to use the beta distribution as the priors of $\mathcal{P}$ and $\mathcal{N}$, i.e., $\mathcal{B}(TP^j | a_1, a_2)$ and $\mathcal{B}(TN^j | b_1, b_2)$. Note that we specify the priors in terms of the mean $\mu$ and variance $\sigma^2$. The mean and the variance for a beta prior are given by $\mu =$

$a_1/(a_1+a_2)$ and $\sigma^2 = a_1a_2/((a_1+a_2)^2(a_1+a_2+1))$, resulting in $a_1 = (-\mu^3+\mu^2-\mu\sigma^2)/\sigma^2$ and $a_2 = a_1(1-\mu)/\mu$, likewise for $b_1$ and $b_2$.

In addition, since we do not have features $x_i$ and we wish to obtain an estimate of the actual ground truth based only on the labels from multiple annotators. We thus estimate $p = prob[z = 1]$ indicating the prevalence of the positive class. We assume a beta prior for the prevalence, i.e., $\mathcal{B}(p|c_1, c_2)$. The EM algorithm is simplified as follows (Raykar et al. 2010).

1. Let $\mu_i = \frac{1}{N}\sum_{j=1}^{R} y_i^j$ based on majority voting.
2. Given $\mu_i$, the true positive and true negative can be estimated by:

$$TP^j = \frac{a_1^j - 1 + \sum_{i=1}^{N} \mu_i y_i^j}{a_1^j + a_2^j - 2 + \sum_{i=1}^{N} \mu_i},$$

and

$$TN^j = \frac{b_1^j - 1 + \sum_{i=1}^{N} (1-\mu_i)(1-y_i^j)}{b_1^j + b_2^j - 2 + \sum_{i=1}^{N} (1-\mu_i)},$$

the prevalence of the positive class is estimated by:

$$p = \frac{c_1 - 1 + \sum_{i=1}^{N} \mu_i}{c_2 + c_2 - 2 + N}.$$

3. Update $\mu_i$ by

$$\mu_i = \frac{a_i p}{a_i p + b_i(1-p)},$$

where $a_i = \prod_{j=1}^{R} [TP^j]^{y_i^j} [1 - TP^j]^{1-y_i^j}$ and $b_i = \prod_{j=1}^{R} [TN^j]^{1-y_i^j} [1 - TN^j]^{y_i^j}$.

Steps 2 and 3 are iterated until convergence is reached. Once the EM algorithm stops, we can use $\mu_i$ to estimate the true labels $z_i$ by setting a threshold. We simply set the threshold as 0.5, i.e., if $\mu_i > 0.5$, the value of $\hat{z}_i$ is assigned with 1, otherwise 0. We view $\mu_i$ as the weight of the constraint that "$z_i$ is $true$". As a result, we can build a set of weighted constraints with respect to each task in $H$.

## Build Constraints Based on Plan Traces

Since annotators might give erroneous or outright malicious responses, we explore additional information from plan traces to reduce the negative effect from erroneous annotators. We explore this information in the form of constraints. Specifically, we build three types of constraints, i.e., *state constraints*, *action constraints* and *plan constraints*, to capture the knowledge helpful for acquiring action models. These three types of constraints were also explored by (Yang, Wu, and Jiang 2007).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Solving All Constraints

In Step 4, we solve all the weighted constraints built by Steps 2 and 3 using a weighted MAX-SAT solver (Borchers and Furman 1998; Davies and Bacchus 2011). Before feeding the weighted constraints to the weighted MAX-SAT solver, we adjust the weights of crowdsourcing constraints by replacing the original weights (denoted as $w_o$ ($0 \leq w_o \leq 1$), which is calculated by the similarity function), with $\frac{\gamma}{1-\gamma} \times w_m \times w_o$, where $w_m$, as the scale factor for $w_o$, is

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

## Experiments

We test our learning algorithm CAMA in three domains *blocks*[4], *driverlog*[5] and *laundry*. *laundry* is a domain created based on the description in the MIT PLIA1 dataset[6]. We collect 75 plan traces as training data in each domain. To collect labels of variables, we design the crowdsourcing process by the following two ways. The first way, i.e., the simulated way, is to build 100 virtual annotators to simulate the crowd, and randomly generate labels with the parameter $\theta$. Each virtual annotator is controlled by $\theta$ when labeling variables. The other way, i.e., the manual way, to invite 30 people (including 20 students and 10 computer engineers) to provide labels for variables. As mentioned in the section of Step 1 of Algorithm 1, we change propositions to short surveys in natural language, which means annotators can answer questions solely based on their background knowledge, with no need to additionally learn AI planning background.

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

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

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

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

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

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

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

### Comparison CAMA and ARMS

We first evaluated CAMA in the **simulated** crowdsourcing data, i.e., the labels were collected in the simulated way. We repeated our CAMA five times to calculate an average of accuracies. Each time we randomly selected one of each five sequential intermediate partial states being observed in plan traces leaving other states empty, and each partial state was selected by 50% of propositions in the corresponding full state (a state was assumed to be represented by a set of propositions). We compared our CAMA algorithm to the previous learning system ARMS (Yang, Wu, and Jiang 2007), where ARMS learnt action models without any information from the crowd. We show the experimental results in Figure 1. The parameter $\gamma$, which is introduced in the previous section, is set as $0.5$ when running our algorithm CAMA.

From Figure 1, we can see that the accuracy of our algorithm CAMA is higher than ARMS, which indicates that exploiting the knowledge from the crowd can indeed perform better than learning without this knowledge, as ARMS does. We can also observe that when the number of plan traces is small, the difference between our algorithm CAMA and ARMS is larger. However, when the number of plan traces becomes large, the gap shrinks. This phenomenon indicates that our algorithm CAMA provides better effect on learning the action models when we do not have enough plan traces, since when the number of plan traces becomes larger, there will be more knowledge available from the plan traces themselves, which can be enough to learn the action models. The result also reveals that even when the number of plan traces is very small (e.g., 30), the learning accuracy of our algorithm CAMA will be no less than $80\%$, which means that exploiting the knowledge the crowd can really help learning action models.

Likewise, we compared our CAMA algorithm to ARMS in the **manual** crowdsourcing data. The results are shown in Figure 2. From the results, we can observe that CAMA performs much better than ARMS in the manual crowdsourcing data. We can also see that the difference between CAMA and ARMS becomes smaller as the number of plan traces becomes larger. The reason is the same as the one provided in the previous paragraph, i.e., the knowledge becomes rich enough for both systems to learn high-quality action models when the number becomes large, and weakens the impact of knowledge from crowdsourcing on the other hand.

### Varying the $\gamma$ Parameter

We varied the value of the parameter $\gamma$ from 0 to 1 to see the trend of the accuracy, by fixing the number of plan traces to 60. We show the results in Figure 3. From Figure 3, we can see that when $\gamma$ increases from 0 to 0.5, the accuracy increases, which exhibits that when the effect of the knowledge from crowdsourcing enlarges, the learning accuracy gets higher. However, when $\gamma$ is larger than 0.5 (note that when $\gamma = 0.5$, $\frac{\gamma}{1-\gamma}$ will be equal to 1, which means the weights of crowdsourcing constraints remain unchanged), the accuracy becomes lower when $\gamma$ increases. This is because the impact of the knowledge from the plan traces is relatively reduced when $\gamma$ becomes very large, and implies

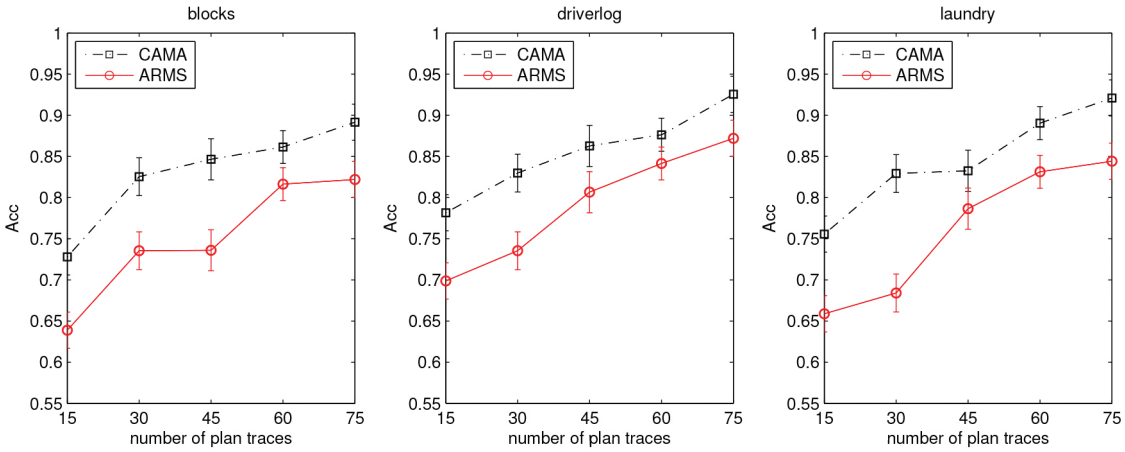


Figure 1: The comparison between CAMA and ARMS in the **simulated** crowdsourcing data.

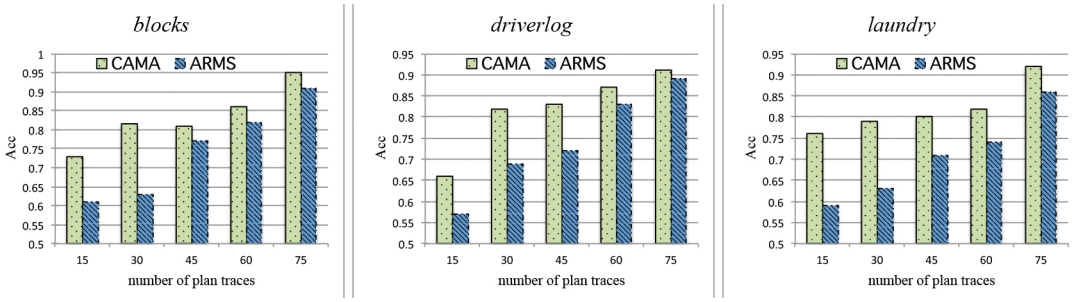


Figure 2: Comparison between CAMA and ARMS in the **manual** crowdsourcing data.

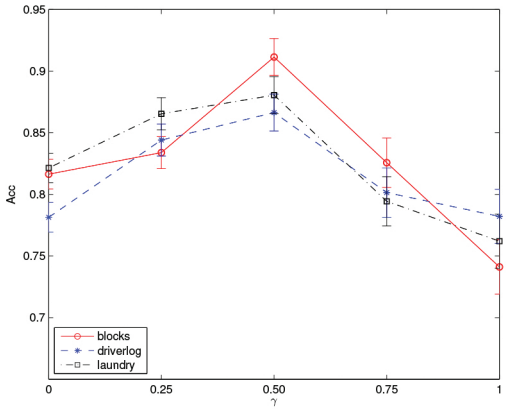


Figure 3: The accuracy with respect to different $\gamma$

that the knowledge from the plan traces is also important for learning high-quality action models. In summary, with the knowledge of the current limited plan traces, exploiting knowledge from crowdsourcing does help improve the learning accuracy.

## Conclusion

We propose a novel crowdsourced action-models acquisition algorithm CAMA to learn action models by employing the knowledge from crowdsourcing. We first build a a set of constraints to capture the information provided by the annotators. We then build constraints from plan traces and solve all the constraints using a weighted MAX-SAT solver. From our experiments, we can see that our CAMA algorithm can learn more accurate action models with the help of the crowd.

In this work, we consider the quality of information acquired from annotators, rather than the cost of completing Human Intelligence Tasks. In the future, we would like to study the feasibility of considering the crowdsourcing cost in our CAMA framework.

## Acknowledgement

Hankz Hankui Zhuo's research is supported by the 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).

## References


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

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

Cenamor, I.; de la Rosa, T.; and Borrajo, D. 2013. Ondroad planner: Building tourist plans using traveling social network information. In *HCOMP*.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of object-centred domain models from planning examples. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Dai, P.; Lin, C. H.; Mausam; and Weld, D. S. 2013. Pomdp-based control of workflows for crowdsourcing. *Artif. Intell.* 202:52–85.

Davies, J., and Bacchus, F. 2011. Solving maxsat by solving a sequence of simpler sat instances. In *17th International Conference on Principles and Practice of Constraint Programming (CP-2011)*, 225–239.

Dawid, A. P., and Skene, A. M. 1979. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28(1):20–28.

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

Hoffmann, J.; Weber, I.; and Kraft, F. M. 2010. SAP speaks PDDL. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Holmes, M. P., and Isbell, Jr., C. L. 2004. Schema learning: Experience-based construction of predictive action models. In *In Advances in Neural Information Processing Systems 17 (NIPS-04)*.

Howe, J. 2008. Crowd sourcing: Why the power of the crowd is driving the future of business.

Kajino, H.; Tsuboi, Y.; and Kashima, H. 2012. A convex formulation for learning from crowds. In *AAAI*.

Kamar, E.; Hacker, S.; and Horvitz, E. 2012. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*, 467–474.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story generation with crowdsourced plot graphs. In *AAAI*.

Liem, B.; Zhang, H.; and Chen, Y. 2011. An iterative dual pathway structure for speech-to-text transcription. In *Human Computation*.

Manikonda, L.; Chakraborti, T.; De, S.; Talamadupula, K.; and Kambhampati, S. 2014. AI-MIX: using automated planning to steer human workers towards better crowdsourced plans. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 3004–3009.

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

Ouyang, T., and Li, Y. 2012. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *CHI*, 2895–2904.

Raykar, V. C., and Yu, S. 2011. Ranking annotators for crowdsourced labeling tasks. In *NIPS*, 1809–1817.

Raykar, V. C.; Yu, S.; Zhao, L. H.; Valadez, G. H.; Florin, C.; Bogoni, L.; and Moy, L. 2010. Learning from crowds. *Journal of Machine Learning Research* 11:1297–1322.

Romney, A. K.; Weller, S. C.; and Batchelder, W. H. 1986. Culture as consensus: A theory of culture and informant accurac. *American Anthropologist, New Series* 88(2):313–338.

Talamadupula, K.; Kambhampati, S.; Hu, Y.; Nguyen, T. A.; and Zhuo, H. H. 2013. Herding the crowd: Automated planning for crowdsourced planning. In *HCOMP*.

Welinder, P.; Branson, S.; Belongie, S.; and Perona, P. 2010. The multidimensional wisdom of crowds. In *NIPS*, 2424–2432.

Whitehill, J.; Ruvolo, P.; Wu, T.; Bergsma, J.; and Movellan, J. R. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, 2035–2043.

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

Zhang, H.; Law, E.; Miller, R.; Gajos, K.; Parkes, D. C.; and Horvitz, E. 2012. Human computation tasks with global constraints. In *CHI*, 217–226.

Zhuo, H. H., and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artif. Intell.* 212:80–103.

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

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artif. Intell.* 212:134–157.