Multi-Agent Plan Recognition with Partial Team Traces and Plan Libraries

Hankz Hankui Zhuo and **Lei Li**
Department of Computer Science,
Sun Yat-sen University, Guangzhou, China
{zhuohank,lnslilei}@mail.sysu.edu.cn

Abstract

Multi-Agent Plan Recognition (MAPR) seeks to identify the dynamic team structures and team behaviors from the observed activity sequences (team traces) of a set of intelligent agents, based on a library of known team activity sequences (team plans). Previous MAPR systems require that team traces and team plans are fully observed. In this paper we relax this constraint, i.e., team traces and team plans are allowed to be partial. This is an important task in applying MAPR to real-world domains, since in many applications it is often difficult to collect full team traces or team plans due to environment limitations, e.g., military operation. This is also a hard problem since the information available is limited. We propose a novel approach to recognizing team plans from partial team traces and team plans. We encode the MAPR problem as a satisfaction problem and solve the problem using a state-of-the-art weighted MAX-SAT solver. We empirically show that our algorithm is both effective and efficient.

1 Introduction

Multi-Agent Plan Recognition (MAPR) explores an explanation of the observed team trace, i.e., activity sequences of a set of agents, by identifying the dynamic team structures and team behaviors of agents based on a library of team plans. It has important applications in analyzing data from automated monitoring, surveillance and intelligence analysis in general [Banerjee *et al.*, 2010]. It is also a difficult task since an observed team trace is often composed of many possible team plans in the library, and team-mates may dynamically change in the observing process.

There have been many techniques designed to automatically recognize team plans given an observed team trace as input. For instance, Avrahami-Zilberbrand and Kaminkaa presented a Dynamic Hierarchical Group Model (DHGM), which indicated the connection between agents, to track the dynamically changed structures of groups of agents [Avrahami-Zilberbrand and Kaminka, 2007]; Sukthankar and Sycara proposed another recognizing algorithm that encoded the dynamic team membership to prune the size of the plan library, assuming that agents in the same team execute a common activity [Sukthankar and Sycara, 2008]; Banerjee et al. proposed to formalize MAPR with a new model, revealing the distinction between the hardness of single and multi-agent plan recognition, and solve MAPR problems in the model using a first-cut approach, provided that a *fully* observed team trace and a library of *full* team plans were given as input [Banerjee *et al.*, 2010]; etc.

Despite the success of previous approaches, they either assume that agents in the same team can only execute a common activity, i.e., coordinated activities of agents are not allowed in a team, or require that the team trace and team plans are *complete*, i.e., missing values (activities that are missing) are not allowed. In many real-world applications, however, it is often difficult to observe full team traces or collect full team plans due to environment or resource limitations. For example, in military operation, it may be difficult to observe every activity of team-mates, since team-mates sometimes need to hide when their enemies are attacking. As another example, in team work of a company, there may be no sufficient sensors to be set in each possible place to observe each activity of team-mates. Thus, it is important to design a novel approach to solving the problem that team traces and team plans are not fully observed.

In this paper, we seek to develop a novel algorithm framework to recognize multi-agent plans provided there are *missing values* in team traces and team plans. We call our algorithm `MARS`, which stands for **M**ulti-**A**gent plan **R**ecognition **S**ystem. `MARS` first builds a set of *candidate occurrences*, i.e., a possible case that a team plan occurs in the team trace. After that, it generates sets of *soft* constraints and *hard* constraints based on candidate occurrences. Finally, it solves all these constraints using a state-of-the-art weighted MAX-SAT solver, such as MaxSatz [LI *et al.*, 2009], and converts the solving result to the solution of our MAPR problem.

We organize the paper as follows. We first introduce the related work in the next section, and then give the formulation of our MAPR problem. After that, we present our `MARS` algorithm and discuss properties of `MARS`. Finally, we evaluate `MARS` in the experiment section and conclude the paper.

2 Related work

The plan recognition problem has been addressed by many algorithms, e.g., Kautz proposed an approach to recognize plans based on parsing view actions as sequences of subactions and essentially model this knowledge as a context-free rule in an "action grammar" [Kautz and Allen, 1986]; Bui et al. presented approaches to probabilistic plan recognition problems [Bui, 2003; Geib and Goldman, 2009]; instead of using a library of plans, Ramrez and Geffner proposed an approach to solving the plan recognition problem using slightly modified planning algorithms [Ramrez and Geffner, 2009]; etc. Despite the success of the above systems, they all focus on single agent plan recognition.

For multi-agent plan recognition, Sukthankar et al. presented an approach for multi-agent plan recognition that leveraged several types of agent resource dependencies and temporal ordering constraints in the plan library to prune the size of the plan library considered for each observation trace [Sukthankar and Sycara, 2008]. Avrahami-Zilberbrand and Kaminka preferred a library of single agent plans to team plans, but identified dynamic teams based on the assumption that all agents in a team executing the same plan under the temporal constraints of that plan [Avrahami-Zilberbrand and Kaminka, 2007]. The constraint on activities of the agents that can form a team can be severely limiting when teammates can execute coordinated but different behaviors.

Instead of using the assumption that agents in the same team should execute a common activity, Sadilek and Kautz provided a unified framework to model and recognize activities that involved multiple related individuals playing a variety of roles [Sadilek and Kautz, 2010]; Banerjee et al. formalized multi-agent plan recognition using a basic model that explicated the cost of abduction in single agent plan recognition by "flatting" or decompressing the plan library [Banerjee *et al.*, 2010]. In these systems, although coordinated activities can be recognized, they either assume there is a set of real-world GPS data available, or assume that team traces and team plans can be fully observed. In this paper, we allow that: (1) agents can execute coordinated different activities in a team, (2) team traces and team plans can be partial, and (3) GPS data is not needed.

3 Problem Definition

Considering a set of agents $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, a library of team plans $\mathcal{P}$ of agents $\mathcal{A}$ is defined as a set of matrices. Specifically, each team plan $p \in \mathcal{P}$ ($p = [p_{ij}]$) is an $r \times c$ matrix, where $0 < r \leq T$, $1 < c \leq n$, and T is the number of total time steps. p_{ij} is an activity that is expected to be executed at time i by agent j, where $i = \{1, 2, \dots, c\}$ and $j = \{1, 2, \dots, r\}$. The value of each team plan p is associated with an utility function $\mu(p)$.

An observed team trace $\mathcal{O}$ executed by agents $\mathcal{A}$ is a matrix $\mathcal{O} = [o_{tj}]$. o_{tj} is the observed activity executed by agent j at time step t, where $0 < t \leq T$ and $0 < j \leq n$. A team trace $\mathcal{O}$ is *partial*, if some elements in $\mathcal{O}$ are empty (denoted by *null*), i.e., there are missing values in $\mathcal{O}$. We define *occurrence* (we slightly revise the original definition given by [Banerjee *et al.*, 2010]) as follows:

Definition 1 (Occurrence): *A team plan (sub-matrix)* $p = [p_{ij}]_{r \times c}$ *is said to occur in a matrix* $\mathcal{O}$ *if* r *contiguous rows* $(t_1, \dots, t_r, t_i = t_{i-1} + 1)$*, and* c *columns (say* $k_1, \dots, k_c$*, a* *c-selection in any order from* n *agent indices) can be found in* $\mathcal{O}$ *such that*

$$o_{t_i k_j} = null \vee p_{ij} = null \vee p_{ij} = o_{t_i k_j},$$

where $i = 1, \dots, r$*,* $j = 1, \dots, c$*, and* $o_{t_i k_j}$ *is the observed activity of row* t_i *and column* k_j *in* $\mathcal{O}$*.*

We denote an occurrence by $(t_1, p, \langle \alpha_{k_1}, \alpha_{k_2}, \dots, \alpha_{k_c} \rangle)$, where t_1 is the start position of the occurrence in $\mathcal{O}$, p is a team plan id, $\langle \alpha_{k_1}, \alpha_{k_2}, \dots, \alpha_{k_c} \rangle$ is the agent sequence as is described in definition 1.

Our multi-agent plan recognition problem can be defined by: given as input a team trace and a library of team plans, both of which may have missing values, our algorithm outputs a set of occurrences $\mathcal{C}^{sol}$ that satisfies the following conditions **C1-C3**:

C1: All occurrences in $\mathcal{C}^{sol}$ occur in the team trace.

C2: For each activity o_{ij} in $\mathcal{O}$, o_{ij} occurs in *exactly* one occurrence in $\mathcal{C}^{sol}$.

C3: The total utility of $\mathcal{C}^{sol}$ is optimal, i.e., $\sum_{p \in \mathcal{C}^{sol}} \mu(p)$ is the maximal utility that can be obtained.

We show an example of our recognition problem in Figure 1. In the team trace, α_1, α_2, α_3, and α_4 are agents. t is a time step ($(1 \leq t \leq 4)$). a, b, c, d, and e are activities. $null$ indicates the activity that is missing. p_1, p_2, p_3, and p_4 are four team plans that compose a library. In the output, there are five occurrences that exactly cover the inputted team trace.

input: a team trace

t	**α1**	**α2**	**α3**	**α4**
1	*a*	*d*	*null*	*a*
2	*a*	*b*	*c*	*null*
3	*null*	*null*	*c*	*e*
4	*b*	*e*	*b*	*null*

input: a library of team plans

null	*null*
a	*c*
b	*b*

e	*a*

a	*d*
b	*b*

b	*c*
a	*null*

output: {(2, *p1*, <α1, α3>), (1, *p2*, <α4, α2>), (1, *p* , <α3, α1>),
(3, *p* , <α4, α2>), (4, *p* , <α2, α4>)}

Figure 1: An example of our recognition problem.

4 Our MARS algorithm

An overview of our MARS algorithm is shown in algorithm 1. We will present each step of Algorithm 1 in detail in Sections 4.1-4.4.

4.1 Creating candidate occurrences

In step 1 of Algorithm 1, we first create a set of *candidate* occurrences, denoted by $\mathcal{C}^{cand}$, by scanning all the team plans in $\mathcal{P}$. Each candidate occurrence $c \in \mathcal{C}^{cand}$ is probably an occurrence that composes the final solution to the MAPR problem, i.e., $c \in \mathcal{C}^{sol}$ probably holds. We call a candidate occurrence c a *solution occurrence* if $c \in \mathcal{C}^{sol}$. We describe the creating process in Algorithm 2.

Algorithm 1 An overview of our `MARS` algorithm

input: a library of team plans $\mathcal{P}$ and a team trace $\mathcal{O}$.
outputs: a set of occurrences $\mathcal{C}^{sol}$ that cover $\mathcal{O}$.

```
create a set of candidate occurrences in O:
    C^cand=create-candidate(P, O);
generate a set of soft constraints SC;
generate a set of hard constraints HC;
solving all the constraints using a weighted MAX-SAT solver;
convert the solving result to C^sol;
return C^sol;
```

Algorithm 2 $\mathcal{C}^{cand}$=*create_candidate*($\mathcal{P}$, $\mathcal{O}$)

input: a library of team plans $\mathcal{P}$ and a team trace $\mathcal{O}$
output: a set of candidate occurrences $\mathcal{C}^{cand}$

```
1: C^cand = ∅;
2: for t = 1 to T do
3:    for each p ∈ P do
4:       for each c-selection ⟨α_{k_1}, α_{k_2}, ..., α_{k_c}⟩, such that
         p occurs in O starting at position t in O do
5:          C^cand = {(t, p, ⟨α_{k_1}, α_{k_2}, ..., α_{k_c}⟩)} ∪ C^cand;
6:       end for
7:    end for
8: end for
9: return C^cand;
```

Table 1: An example of candidate occurrences $\mathcal{C}^{cand}$ that can be created by Algorithm 2.

$\mathcal{C}^{cand}$=$\{(1,p_2,\langle\alpha_4,\alpha_2\rangle)$,$(1,p_3,\langle\alpha_3,\alpha_1\rangle)$,
$(1,p_3,\langle\alpha_3,\alpha_4\rangle)$, $(2,p_1,\langle\alpha_1,\alpha_3\rangle)$,$(2,p_3,\langle\alpha_4,\alpha_1\rangle)$,
$(2,p_4,\langle\alpha_2,\alpha_3\rangle)$, $(3,p_3,\langle\alpha_1,\alpha_2\rangle)$,$(3,p_3,\langle\alpha_2,\alpha_1\rangle)$,
$(3,p_3,\langle\alpha_4,\alpha_1\rangle)$, $(3,p_3,\langle\alpha_4,\alpha_2\rangle)$,$(4,p_3,\langle\alpha_2,\alpha_4\rangle)\}$

In step 4 of Algorithm 2, $\langle\alpha_{k_1},\alpha_{k_2},\ldots,\alpha_{k_c}\rangle$ is a c-selection in any order from n agent indices (c is the number of columns in p), as is described in definition 1. Note that since there may be different c-selections such that p occurs in $\mathcal{O}$ (because there may be different columns with the same values), we need to search all the possible c-selections to create all possible candidate occurrences. *For instance, in Figure 1, there are two possible c-selections $\langle 3,1\rangle$ and $\langle 3,4\rangle$ (or equivalently, $\langle\alpha_3,\alpha_1\rangle$ and $\langle\alpha_3,\alpha_4\rangle$) for p_3 starting at $t=1$ position in $\mathcal{O}$. As a result, we can build all the candidate occurrences $\mathcal{C}^{cand}$, as is shown in Table 1, with inputs given by Figure 1:*

4.2 Generate *Soft* Constraints

For each candidate occurrence $c_i \in \mathcal{C}^{cand}$, we conjecture that it could be possibly one of the final set of solution occurrences $\mathcal{C}^{sol}$. In other words, for each candidate occurrence c_i, the following constraint could possibly hold: $c_i \in \mathcal{C}^{sol}$. We associate this constraint with weight w_i to specify that it is not 100% to be *true*. We call this kind of constraints *soft constraints* (denoted by SC). We calculate weight w_i of candidate occurrence c_i with the following equation:

$$w_i = \lambda(c_i) \times \mu(p_i), \quad (1)$$

where the first term $\lambda(c_i)$ is the observing rate (defined later), describing the degree of the *confidence* that candidate occurrence c_i happens. Generally, we assume the more activities being observed, the more confidence we have on the happening of c_i. The second term $\mu(p_i)$ is the utility of p_i that needs to be considered as an impact factor in order to maximize the total utility. Note that p_i is the team plan id in c_i. The observing rate $\lambda(c_i)$ is defined by

$$\lambda(c_i) = \frac{2|p_i| - |\{null_{p_i}\}| - |\{null_{\mathcal{O}}\}|}{2|p_i|},$$

where $|p_i|$ is the total number of actions of team plan p_i, $|\{null_{\mathcal{O}}\}|$ is the number of $null$ in the scope of $\mathcal{O}$ restricted by c_i, and $|\{null_{p_i}\}|$ is the number of $null$ in p_i.

For example, consider the occurrence $(2,p_1,\langle\alpha_1,\alpha_3\rangle)$ that is shown in Figure 1. The number of p_1 is 6, i.e., $|p_1|=6$. The number of null is 1 corresponding to the occurrence in the observed team trace, i.e., $|\{null_{\mathcal{O}}\}|=1$. The number of null in p_1 is 2, i.e., $|\{null_{p_1}\}|=2$ Thus, we have $\lambda(p_1)=\frac{2\times 6-2-1}{2\times 6}=\frac{3}{4}$.

It is easy to find that $\lambda(c_i)$ is equivalent to zero when $|\{null\}|=|p_i|$, resulting in $w_i=0$. This suggests that the occurrence c_i cannot be a solution occurrence if none of its actions are observed. This is not true according to our MAPR problem definition. Thus, we relax this constraint by revising $\lambda(c_i)$ as follows

$$\lambda(c_i) = \frac{2|p_i| - |\{null_{p_i}\}| - |\{null_{\mathcal{O}}\}| + 1}{2|p_i| + 1}, \quad (2)$$

where "1" can be interpreted as there is a *virtual* action (that makes the occurrence c_i happen) which can always be observed.

4.3 Generating *Hard* Constraints

According to condition **C2**, each element of $\mathcal{O}$ should be covered by *exactly* one solution occurrence. In this step, we seek to build *hard* constraints to satisfy this condition. To do this, we first collect an occurrence subset of $\mathcal{C}^{cand}$ for each element $o_{ij} \in \mathcal{O}$, such that o_{ij} is covered by all the occurrences in the subset. We use S_{ij} to denote this subset, and $\mathbb{S}$ to denote the collection of all the subsets with respect to different elements of $\mathcal{O}$, i.e., $\mathbb{S}=\{S_{ij}|o_{ij}\in\mathcal{O}\}$. The detailed description can be found from Algorithm 3. Note that the collection $\mathbb{S}$ has different elements, which is guaranteed by the union operator in step 10 of Algorithm 3.

With $\mathbb{S}$, we generate hard constraints to guarantee condition **C2** as follows. For each subset $S \in \mathbb{S}$, there is only one occurrence $c \in S$ that belongs to $\mathcal{C}^{sol}$, i.e., the proposition variable "$c \in \mathcal{C}^{sol}$" is assigned to be *true*. Formally, we have the following constraints

$$\bigvee_{c\in S}(c \in \mathcal{C}^{sol} \wedge \bigwedge_{c'\in S-\{c\}} c' \notin \mathcal{C}^{sol}),$$

Algorithm 3 Build a collection of subsets of candidate occurrences in $\mathcal{C}^{cand}$

input: The team trace $\mathcal{O}$ and candidate occurrences $\mathcal{C}^{cand}$.
output: A collection $\mathbb{S}$ of subsets of occurrences in $\mathcal{C}^{cand}$.

$\mathbb{S} = \emptyset$;
for each element $o_{ij} \in \mathcal{O}$ **do**
  $S = \emptyset$;
  for each candidate occurrence c in $\mathcal{C}^{cand}$ **do**
    if o_{ij} is covered by c **then**
      $S = S \cup \{c\}$;
    end if
  end for
  if $S \neq \emptyset$ **then**
    $\mathbb{S} = \mathbb{S} \cup \{S\}$;
  end if
end for
return $\mathbb{S}$;

where the term $\bigwedge_{c' \in S - \{c\}} c' \notin \mathcal{C}^{sol}$ indicates all occurrences, which are different from c, do not belong to $\mathcal{C}^{sol}$. Furthermore, we have the following constraints with respect to $\mathbb{S}$,

$$\bigwedge_{S \in \mathbb{S}} \{ \bigvee_{c \in S} (c \in \mathcal{C}^{sol} \wedge \bigwedge_{c' \in S - \{c\}} c' \notin \mathcal{C}^{sol})\}. \qquad (3)$$

We set the weights of this kind of constraints, denoted by HC, with "high" enough values to guarantee these constraints are *hard*. We empirically choose the sum of the weights of soft constraints as this "high" value.

4.4 Solving constraints

With steps 2 and 3 of Algorithm 1, we have two kinds of weighted constraints, i.e., soft constraints SC and hard constraints HC. In this step, we put SC and HC together and solve them using a weighted MAX-SAT solver. In the experiment, we would like to test two different cases using or not using the observing rate function $\lambda(c_i)$. We introduce a new parameter $\rho \in \{0, 1\}$ and revise Equation (1) to a new equation as shown below.

$$w_i = \lambda^{\rho}(c_i) \times \mu(p_i). \qquad (4)$$

If ρ is 1, Equation (4) is reduced to Equation (1); otherwise, Equation (4) is reduced to $w_i = \mu(p_i)$. We will evaluate that $\lambda(c_i)$ is helpful in improving the recognizing accuracy in the experiment section.

The solving result of the weighted MAX-SAT solver is a set of assignments (*true* or *false*) to proposition variables $\{c_i \in \mathcal{C}^{sol} | \text{for all } c_i \in \mathcal{C}^{cand}\}$. If a proposition variable "$c_i \in \mathcal{C}^{sol}$" is assigned to be *true*, c_i is one of the solution occurrences outputted by MARS; otherwise, c_i is not outputted by MARS.

4.5 Discussion

In this section, we discuss the properties of our MARS algorithm related to *completeness* and *soundness*.

Property 1 (Completeness): *The completeness of* MARS *depends only on the completeness of the weighted MAX-SAT solver, i.e., given an MAPR problem that is solvable,* MARS *can output a solution to this problem if the weighted MAX-SAT solver is complete.*

The sketch of the proof can be presented as follows. For each solvable MAPR problem, we can encode the problem with constraints SC and HC in polynomial time by steps 1-3 of Algorithm 1. Furthermore, if the weighted MAX-SAT solver is complete, it can successfully solve these constraints (by step 4) and output a solving result, which can be converted to the solution to the MAPR problem in polynomial time (by step 5). □

Property 2 (Soundness): *Given an MAPR problem, if the ρ is set to be 0 in Equation (4), the output of* MARS *is the solution to the MAPR problem.*

The sketch of the proof can be described as follows. From step 1 of Algorithm 1, we can see that the candidate occurrences $\mathcal{C}^{cand}$, covered by the observed team trace, are all from the library of team plans, which satisfies the first condition C1 in Section 3. From step 2, if ρ is set to be 0, the weights of soft constraints are determined by the utility function μ. Furthermore, the solution outputted by the weighted MAX-SAT solver maximizes the total weights (which is done by step 4), which suggests the second condition C3 is also satisfied. Finally, the third condition C3 is satisfied by the hard constraints established by step 3. Thus, the output of MARS satisfies C1-C3, which means it is the solution to the MAPR problem. □

5 Experiment

5.1 Dataset and Criterion

We follow the experimental method prescribed by [Banerjee *et al.*, 2010] to generate a set of MAPR problems. For generating an MAPR problem, we first generate a random team trace with dimensions 100×50, i.e., 100 time steps and 50 agents. Each element of the team trace belongs to a set of activities A with $|A| = 20$. We randomly partition the team trace into a set of team plans which initiate the members of the library of team plans $\mathcal{P}$. This guarantees there is a solution to the MAPR problem. We generate a set of such MAPR problems $\mathbb{R}$, where $|\mathbb{R}| = 3000$. After that we add M random team plans to library $\mathcal{P}$ of each MAPR problem to enlarge the library. We will test different values of M from $\{20, 40, 60, 80\}$ to vary the size of the library. We will also test different percentages ξ of random missing values from $\{0\%, 10\%, 20\%, 30\%, 40\%, 50\%\}$ for each team trace and team plan. Note that "$\xi = 10\%$" indicates that there are randomly 10% of values that are missing in each team trace and team plan, likewise for other ξ. To define the utility function of team plans, we associate each team plan with a random utility value.

To evaluate MARS, we define a recognizing accuracy $Acc(\xi, M)$ as follows. For each MAPR problem $R \in \mathbb{R}$ with a specific M value and $\xi = 0\%$ (without any missing value), we solve the problem using MARS and denote the solution by $\mathcal{C}$. After that, we revise the problem R by setting ξ with another percentage to get a new problem R'. We solve R' and get a solution $\mathcal{C}'$. If $\mathcal{C}'$ is the same as $\mathcal{C}$, the function $\theta_R(\xi, M)$ with respect to R is set to be 1. Otherwise, $\theta_R(\xi, M)$ is set to be 0. The recognizing accuracy with respect to ξ and M can be defined by

$$Acc(\xi, M) = \frac{\sum_{R \in \mathbb{R}} \theta_R(\xi, M)}{|\mathbb{R}|}. \tag{5}$$

As a special case, $Acc(\xi, M) = 1$ when ξ is 0%.

5.2 Experimental Results

We would like to test the following four aspects of MARS: (1) the recognizing accuracy with respect to different percentages of missing values; (2) the recognizing accuracy with respect to different numbers of randomly added team plans (referred to as "team plans" for simplicity); (3) the number of generated clauses with respect to each percentage; (4) the running time with respect to different percentages of missing values. We present the experiment results in these aspects below.

Varying the percentage of missing values

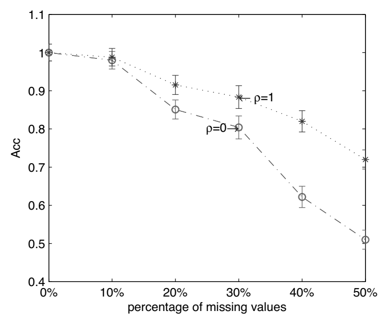

Figure 2: The recognizing accuracy with respect to different percentages of missing values (M is set to be 80)

To evaluate that MARS functions well in missing-value problems, we set the number of team plans M to be 80, and vary the percentage of missing values ξ from $\{0\%, 10\%, 20\%, 30\%, 40\%, 50\%\}$ to see the recognizing accuracy Acc defined by Equation (5). For each ξ, we run five random selections to calculate an average accuracy. The results are shown in Figure 2, where the curve denoted by "$\rho = 1$ indicates the result obtained by setting $\rho = 1$ in Equation (4), likewise for the curve denoted by "$\rho = 0$".

From Figure 2, we can see that the accuracy Acc generally decreases when the percentage increases, no matter whether ρ is 1 or not. This is expected, because missing values may provide information that may be exploited to find accurate occurrences. The more values are missing, the more information is lost. Considering the difference between two curves, we can find that the accuracy is generally larger when $\rho = 1$ than that when $\rho = 0$. The results are statistically significant; we performed the Student's t-test and the result is 0.0482, which indicates the two curves are significantly different at the 5% significance level. This suggests that the observing rate function λ is helpful in improving the accuracy. We can also find that the difference becomes larger when more values are missing, which indicates that λ is more significant when more values are missing. By observation, MARS generally functions well in handling missing values when there are less than 30% of missing values, where there is no less than the accuracy of about 0.9 by setting $\rho = 1$.

Varying the number of team plans

We now analyze the relation between the recognizing accuracy and the size of team plans. We set the percentage ξ to be 30% and vary the size of team plans M to see the change of accuracies. We show the results in Figure 3. From the curves, we can see that the accuracy is generally reduced with the number of team plans increasing. This is consistent with our intuition, since more "uncertain" information is introduced when more team plans are added (each of which has 30% of missing values that introduce uncertain information).

We also observe that the curve denoted by "$\rho = 1$" is generally better than the one denoted by "$\rho = 0$", and the difference between two curves becomes sharp as the size of team plans increasing. This indicates that MARS that exploits the observing rate function (corresponding to "$\rho = 1$"), performs better than the one that does not exploit it, especially when the size of team plans is large.

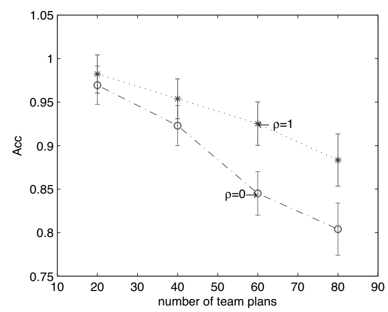

Figure 3: The recognizing accuracy with respect to different numbers of team plans (ξ is set to be 30%)

From Figures 2 and 3, we can conclude that in order to improve the recognizing accuracy, we should better exploit the observing rate function to control weights of constraints as is given in Equation (1), especially when the percentage of missing values and the size of team plans are large.

The generated clauses

We would like to verify that the generated constraints SC and HC would not increase too fast when the percentage of missing values increases. We record the total number of clauses, which correspond to SC and HC, with respect to each percentage of missing values. Note that the clauses obtained from SC and HC are disjunctions of propositions that can be solved directly by a weighted MAX-SAT solver. The results are shown in Table 2, where the first column is the number of team plans, and the other columns are numbers of clauses corresponding to each percentage of missing values. For instance, in the second row and the last column, "392" is the number of clauses that are generated from 20 team plans with 50% of missing values. Note that the numbers of clauses in Table 2 are average results over 3000 MAPR problems. We observe that we can fit the performance curve with a polynomial of order 2. As an example, we provide the polynomial

Table 2: Average numbers of clauses with respect to different percentages of missing values

# team plans	percentage of missing values					
	0%	10%	20%	30%	40%	50%
20	181	233	269	282	338	392
40	225	253	341	482	619	773
60	536	611	657	718	821	1059
80	692	785	843	929	983	1123

for fitting the numbers of clauses of the last row in Table 2, which is $y = 0.0391x^2 + 6.1446x + 703.0357$, where x is the number of percentage points (e.g., $x = 50$ in the last column of Table 2) and y is the number of clauses. This suggests that MARS can handle MAPR problems with missing values well since the number of clauses would not increase fast when missing values increase. Note that clauses increase fast may make the weighted MAX-SAT solver difficult or even fail to solve. Likewise, we can also verify that the number of clauses would not increase fast when the size of team plans increases.

The running time

To test the running time of MARS, we set the number of team plans M to be 20, 40, 60 and 80 respectively, and test MARS with respect to different percentages of missing values. The testing results are shown in Figure 4. By comparing Figures (I)-(VI), we can find that, generally, the larger the size of team plans is, the higher the running time is. This is because there are more constraints generated when the size of team plans becomes larger. We also observe that the running time of MARS increases polynomially with the percentage of missing values increasing. To verify our claim, we use the relationship between percentages of missing values and the CPU time to estimate a function that could best fit these points. We find that we can fit the performance curve with a polynomial of order 2 or order 3. We provide the polynomial for fitting cpu time of (II) in Figure 4, i.e., $0.1475x^2 + 8.3593x + 76.6429$, where x is the number of percentage points.

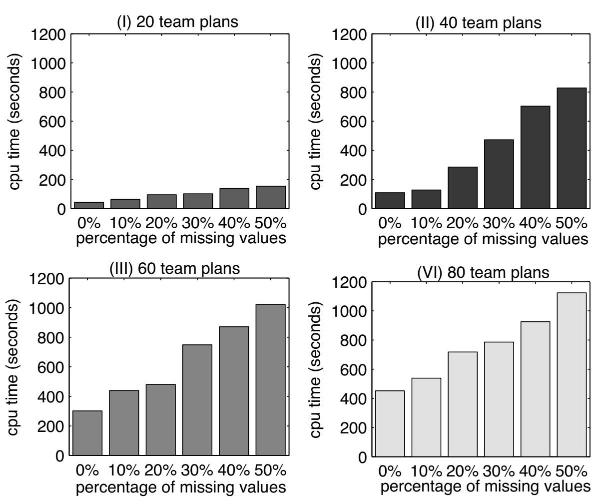

Figure 4: CPU time with respect to different percentages of missing values.

6 Conclusion

In this paper, we have presented a novel algorithm MARS to recognize multi-agent plans. Given a team trace and a library of team plans, MARS builds a set of soft/hard constraints, and solves them using a weighted MAX-SAT solver. The solution obtained is a set of occurrences that cover each element in the team trace exactly once. We observed the following conclusions from our empirical evaluation: (1) Using the observing rate function can help improve the recognizing accuracy, especially when the percentage of missing values and the size of team plans are large, (2) The recognizing accuracy decreases with the missing values or the size of the library increasing, and (3) The running time of our algorithm increases polynomially with the percentage of missing values increasing.

Acknowledgement

Hankz Hankui Zhuo thanks China Postdoctoral Science Foundation funded project(Grant No.20100480806) and National Natural Science Foundation of China (61033010) for suport of this research. Lei Li thanks Macao FDCT 013/2010/A for support of this research.

References

[Avrahami-Zilberbrand and Kaminka, 2007] Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Towards dynamic tracking of multi-agents teams: An initial report. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR 2007)*, 2007.

[Banerjee *et al.*, 2010] Bikramjit Banerjee, Landon Kraemer, and Jeremy Lyle. Multi-agent plan recognition: formalization and algorithms. In *Proceedings of AAAI*, 2010.

[Bui, 2003] Hung H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of IJCAI*, 2003.

[Geib and Goldman, 2009] Christopher W. Geib and Robert P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173(11):1101–1132, 2009.

[Kautz and Allen, 1986] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of AAAI*, 1986.

[LI *et al.*, 2009] Chu Min LI, Felip Manya, Nouredine Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *In proceedings of 12th international conference on the Theory and Applications of Satisfiability Testing (SAT-09)*, pages 467–480, 2009.

[Ramrez and Geffner, 2009] Miquel Ramrez and Hector Geffner. Plan recognition as planning. In *Proceedings of IJCAI*, 2009.

[Sadilek and Kautz, 2010] Adam Sadilek and Henry Kautz. Recognizing multi-agent activities from gps data. In *Proceedings of AAAI*, 2010.

[Sukthankar and Sycara, 2008] Gita Sukthankar and Katia Sycara. Hypothesis pruning and ranking for large plan recognition problems. In *Proceedings of AAAI*, 2008.