Search Performance of Multi-Agent Plan Recognition in a General Model

Bikramjit Banerjee and **Landon Kraemer**
School of Computing
The University of Southern Mississippi
118 College Drive #5106
Hattiesburg, MS 39406
{Bikramjit.Banerjee,Landon.Kraemer}@usm.edu

Abstract

Multi-Agent Plan Recognition (MAPR) seeks to identify the dynamic team structures and team behaviors from the observations of the activity-sequences of a set of intelligent agents, based on a library of known team-activities (plan library). It has important applications in analyzing data from automated monitoring, surveillance, and intelligence analysis in general. Recently, we have introduced a model for MAPR with a flat library structure, to study the complexity of basic MAPR, and also possibly its extensions in the future. Interestingly, this model makes fewer assumptions than existing models, and hence is more general. Therefore, as no existing algorithm would apply to this model, we have developed an hypothesis generation algorithm for this model, and adapted Knuth's Algorithm X for branch and bound search in the resulting hypothesis space. In this paper, we establish the time complexity of hypothesis generation in this model, propose and evaluate 3 different bounding criteria, and also empirically study the dependence of runtimes (hypothesis generation, and search times separately) on the model parameters.

Introduction

Multi-Agent Plan Recognition (MAPR) seeks an explanation of the observed activity-sequences of a set of intelligent agents, in terms of a given library of team-activities, by identifying the dynamic team structures and team behaviors of the agents. MAPR has important applications in analyzing data from automated monitoring, surveillance, and intelligence analysis in general. In a recent report in New York Times, it was revealed that in the year 2009, unmanned aerial vehicles collected about 24 years of video data alone from theaters of warfare (Drew Jan 10 2010). Clearly, intelligence analysis needs advances in AI now more than ever, and MAPR constitutes an important subproblem in this application.

We have recently introduced a new model of MAPR that uses a "flat" plan library structure, and established hardness results in MAPR (Banerjee, Kraemer, and Lyle 2010). In this paper, we describe and analyze a solution approach in this model. This approach first generates the hypothesis space, and then performs a branch and bound search in this space. Since the MAPR problem in this model has significant similarities to the way that Knuth approaches the Pentominoes puzzle (and some other puzzles) (Knuth 2000), we adopt this as the first-cut approach. We generate the hypothesis space the same way, but adapt Knuth's Algorithm X (with the efficient dancing links representation) (Knuth 2000) for branch and bound search to return the optimal solution instead of all solutions. The main contributions of this paper are

- Detailed algorithms for hypotheses generation and search;
- Analysis of the time complexity of hypothesis generation;
- Description of three different bounding criteria for the search;
- Empirical analysis of the relative performances of these bounding criteria;
- Empirical analysis of the dependence of run-times of hypothesis generation and search on the various model parameters.

Although the time complexity of hypothesis generation is shown to be exponential in the number of agents, we find empirically that it is small relative to the search time. Furthermore, our experiments uncover interesting aspects of the dependence of overall run-time (hypothesis generation + search) on the model parameters.

Related Work

In this paper, we are more interested in symbolic approaches to plan recognition, and as such do not review the literature on probabilistic approaches. Plan recognition (or more appropriately *keyhole* plan recognition (Cohen, Perrault, and Allen 1981)) has a long and rich history, dating back to the eighties. The problem admits a natural abductive reasoning approach, with some of the earliest work (Kautz and Allen 1986; Charniak and Goldman 1993) emphasizing the central and inherent role of uncertainty manifested by the disconnect between the multiple possible explanations behind the observations, and the true explanation. Kautz (Kautz and Allen 1986) reduced the problem to deductive inference of explanations for the observations using an action taxonomy, and satisfying a set of simplicity constraints, by performing a series of circumscriptive minimizations that "closed" (in the sense of completeness) the interpretation of the action

taxonomy (Kautz and Allen 1986). The formalization by Kautz and Allen (Kautz and Allen 1986) showed an approximate correspondence to the vertex cover problem in plan graphs, thus establishing a complexity baseline. Related formalization approaches have encouraged a pre-enumeration of the explanation space before reasoning about observations (Bui 2003). The first significant attempt at establishing the complexity of plan recognition was by Vilain (Vilain 1990) who transformed Kautz's plan hierarchies to context-free grammars and framed the problem as one of parsing the observation strings (similar approaches were also proposed before (Sidner 1985), while Pynadath and Wellman (Pynadath and Wellman 2000) proposed probabilistic state dependent grammars to model the recognition of hierarchical behaviors) using the plan grammar. He showed that plan recognition with abstraction and partial ordering in the plan hierarchies is NP-complete, but also identified easier classes under Kautz's scheme.

In recent years, the focus has shifted from formalization to plan *execution*, whereby issues related to plan execution such as interleaved execution of multiple goals, multiple instantiations of the same goal (such as in a cyber attack), goal abandonment (Geib and Goldman 2003), and partially ordered plan hierarchies became the natural extensions of the basic problem (Goldman, Geib, and Miller 1999). A more recent significant attempt at assessing the complexity of plan recognition under the new model was made by Geib (Geib 2004), where Hierarchical Task Network (HTN) plans (Erol, Hendler, and Nau 1994) were used as the appropriate representation of plans. This work centered on estimating the growth in the number of explanations with each new observation subject to the properties of the plan library, chiefly the number of unordered leaders (this roughly stands for the initial action under a goal) and the number of repeated actions.

In contrast to plan recognition, multi-agent plan recognition (MAPR) is a much younger area of research. In MAPR, observations of the activities of a set of agents are made over time, and the goal is to identify the dynamic teams and their activities in the observations by matching with a library of various team activities. The significance of considering group actions in order to isolate team plans, rather than a sequential process of recognizing plans of the individual agents separately, has been a repeatedly emphasized theme (Castelfranchi and Falcone 1995; Devaney and Ram 1998; Tambe 1995; Huber and Durfee 1992; Avrahami-Zilberbrand and Kaminka 2007). In the simpler case where all agents are executing a single plan as one team, the observations can be concatenated and matched against the library (Intille and Bobick 1999). But the more realistic cases involve *dynamic* teams (Tambe 1997), where agents can join and leave teams over the period of the observations. Some previous work has considered static social structures to facilitate the formation of hypotheses on teams and their plans (such as YOYO) (Kaminka and Bowling 2002; Kaminka, Pynadath, and Tambe 2002). *RESC*$_{team}$ (Tambe 1996) is a symbolic MAPR approach that relaxes the assumption of static social structures, but considers only one coherent hypothesis. Hongeng and Nevatia (Hongeng and Nevatia 2001) eschew group plans and instead use action threads of single agents that are related by temporal constraints generating a muli-agent event graph. Similarly, Avrahami-Zilberbrand and Kaminka (Avrahami-Zilberbrand and Kaminka 2007) opt for a library of single agent plans instead of team plans, but identify dynamic teams based on the assumption that all agents in a team execute the same plan under the temporal constraints of that plan. While the idea of renouncing a multi-agent plan library is attractive in terms of complexity management, the constraint on the actions of the agents that can form a team can be severely limiting when team-mates can execute coordinated but different behaviors.

More recently, attempts were made to limit the run-time of MAPR by imposing structure (such as temporal dependencies and constraints (Geib 2004; Sukthankar and Sycara 2008)) in the library. Although such structure-based heuristics can limit the hypotheses space, little research has focused on better ways to search this space. For instance, the STABR algorithm (Sukthankar and Sycara 2006) uses sophisticated heuristics to limit the hypothesis space, but uses brute-force to search it. In this paper, we propose and evaluate a branch and bound algorithm for searching the hypothesis space, produced in a model that is more general than the existing ones.

Problem Formulation

Let A be a set of n agents, $\{a_1, a_2, \ldots, a_n\}$. We are given a trace of activities of these agents over T periods of time, in the form of a matrix, $M = [m_{ij}]$, where m_{ij} is the action executed by agent a_j at time i, $j = 1, \ldots, n$ and $i = 1, \ldots, T$. The actions are represented by symbols from an alphabet Σ. We are also given a library (set) of team plans L, where each team plan, $P \in L$, is in the form of an $x \times y$ matrix (where $1 \leq x \leq T, 1 \leq y \leq n$), $P = [p_{ij}]$, where p_{ij} is the action expected from the jth team-member ($j = 1, \ldots, y$) at the ith ($i = 1, \ldots, x$) step from the start of the plan. The actions in the team-plans are also represented by symbols from Σ. A simple example is shown in Figure 1. In this example, the traces of 4 agents' activities over 4 steps are available, as is a library of team plans including at least the 4 plans shown, L_1–L_4. L_1 for example, says that 3 agents in a team execute the (coordinated) sequences cca, aab and bcb.

Definition 1. (Occurrence) *A team-plan (submatrix) $P = [p_{ij}]_{x \times y}$ is said to* occur *in a matrix M if x contiguous rows ($t_1, \ldots, t_x$, $t_i = t_{i-1} + 1$), and y columns (say $k_1, \ldots, k_y$, a y-selection in any order from n agent indices) can be found in M such that*

$$p_{ij} = M(t_i, k_j), i = 1, \ldots, x, j = 1, \ldots, y$$

We say that each $M(t_i, k_j)$ above is covered *by P. We represent the above occurrence as a tuple $(P, k_1, k_2, \ldots, k_y, t_1)$.*

For instance, in Figure 1, L_1 occurs in the trace matrix (left), with $(t_1, t_2, t_3) = (2, 3, 4)$ and $(k_1, k_2, k_3) = (4, 1, 2)$. The text at the bottom of this figure describes the occurrences of the other plans. Using these 4 plans (L_1–l_4), the following hypothesis can be created: agents 2 and 1 work in a team during the first time step implementing plan L_3 (in

that order), while agents 3 and 4 work individually. At time step 2, agents 4, 2, and 1 form a team for the rest of the observation period to implement plan L_1 (in that order), while agent 3 continues to repeat a pattern of activities (plan L_2) over 2 time steps each, on its own. The utility of this hypothesis is $v(L_1)+2v(L_2)+v(L_3)+v(L_4)$ for some utility function v, and the MAPR problem seeks the maximum-utility hypothesis that explains every observed action *uniquely*.

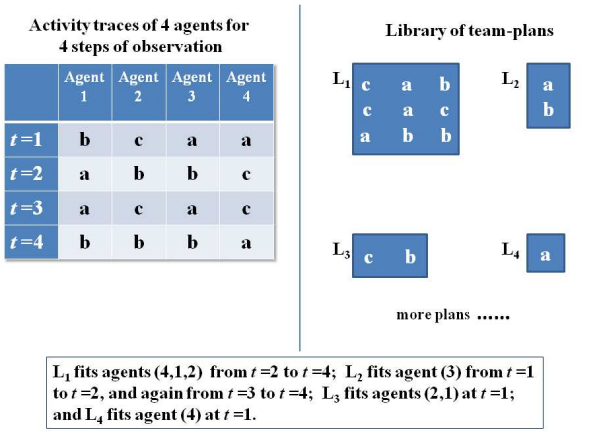

Figure 1: An example of the formalized multi-agent plan recognition problem.

We formalize the above notion of MAPR, parameterized by the sizes of the trace (i.e., T and n) and the library ($|L|$), below:

Definition 2. (MAPR) *The multi-agent plan recognition problem, represented as MAPR$(n, T, |L|, k)$ is defined as follows:*

Instance: *Activity matrix M (of size $T \times n$) such that $m_{ij} \in \Sigma$, a set/library L of submatrices, each containing of symbols from Σ, and $k \in Z$.*

Question: *Is there a collection L' of submatrices from L, along with their occurrences (Definition 1) such that for each m_{ij} there is exactly one $l \in L'$ that* occurs *in M and* covers *m_{ij}, with $\sum_{l \in L'} v(l) \geq k$?*

Notice that a given $l \in L$ can be used multiple times, e.g., in Figure 1 L_2 is used twice.

We have shown in (Banerjee, Kraemer, and Lyle 2010) that MAPR$(n, T, |L|, k)$ is NP-complete, while it is in P if $n = 1$. We have also presented the algorithms for hypothesis generation and branch and bound search, and given a bounding a criterion for the latter. As unique contributions of this paper, we describe these algorithms in detail, along with the complexity of hypothesis generation. We also present 3 different bounding criteria for the search, and experimentally evaluate them and their dependence on the problem parameters, as well as the relative contributions of hypothesis generation and search to the overall runtimes.

Algorithms

In order to solve MAPR in this new model, we adopt Knuth's Algorithm X with the efficient "dancing links" (DLX) (Knuth 2000) representation, but modify it to perform branch and bound search. We first produce a boolean matrix E that encodes the hypothesis space, such that e_{ij} is 1 if and only if plan i in the library L can cover (in the sense in Definition 1) element j in the observations (when the trace matrix is transformed into a linear observation array). Of course, according to Definition 1, this also means that some more $e_{ij'}$ need to be 1 in the vicinity of observation j to ensure that the entire plan matrix i occurs in the trace. Once E is produced, Algorithm X is applied to find a set of rows of E such that for every column of E, there is exactly one row that contains a 1, by backtracking search using the efficient dancing links representation (see (Knuth 2000)).

We can convert any instance of MAPR with a trace M and plan library L to a boolean matrix E as follows. E shall have Tn columns, and a row for each occurrence (Definition 1) of each plan in L. We use the notation $r(l)$ to denote the number of rows in a matrix l and $c(l)$ the number of columns in it. For a given occurrence of $l \in L$, $(l, k_1, k_2, \ldots, k_{c(l)}, t)$, we can create a row in E as

$$e_{row,col} = \begin{cases} 1 & \text{if } col = (t+i-1)n + k_j, \text{ for some} \\ & i \in [0, r(l)-1], j \in [1, c(l)] \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Algorithm 1 describes this process of creating E, by detecting every possible occurrence of every plan in M. For a plan, P, of size $x \times y$ picked in the loop in line 2, it extracts a part of column c from M, of length x, starting at row r, in line 7. This column, s_c is created for all possible $c = 1, \ldots, n$ in the loop of line 6. Line 9 checks if there is any *unordered* y-selection, $(s_{i_1}, \ldots, s_{i_y})$, from $(s_1, s_2, \ldots, s_n)$ such that the concatenation of these columns matches the plan (matrix) P. If so, an occurrence has been found, and equation 1 is applied (lines 10–13) to create a new row for E. This new row is then appended to the current E in line 14. The complexity of this algorithm is established by the following Lemma.

Lemma 1. *The time complexity of Algorithm 1 is $O(nT^2|L|(2e)^{n/2})$.*

Proof: The key steps that dominate the complexity are in the loop 9–17. The construction in line 9 requires us to check for each of the y_i columns in the ith plan, which (if any) agent's x_i steps of activities (starting from row r) matches the plan column. This costs $O(nx_iy_i)$ and produces $n_1, n_2, \ldots, n_{y_i}$ agent-clusters for the y_i plan columns, such that $n_j \geq 0$ agents match the jth column of plan, and $\sum n_j \leq n$. We consider two extreme cases:

Case 1: All y_i columns of the plan are distinct. Then the number of occurrences of this plan at step t is given by

$$\begin{aligned} & n_1 \cdot n_2 \cdot \ldots n_{y_i} \\ \leq\ & (n/y_i + 1)^{y_i+1} \text{ (since } \textstyle\sum n_j \leq n) \\ \leq\ & e^{n/e} \end{aligned}$$

Case 2: All y_i columns of the plan are identical, i.e., all team-mates execute the same activity sequence, according to the plan. In this case, y_i is effectively 1, and $n_1 \leq n$. Then the number of occurrences at step t can be at most

$$\binom{n}{y_i}$$

which is upper bounded by $(2e)^{n/2}$.

Let X represent the above cost. For each such occurrence, a vector A of length Tn is created. Therefore, the total cost of steps 9–17 is

$$(nx_iy_i + X)\,Tn$$

Taking the outer loops of size $|L|$ (line 2) and $T - x_i + 1$ (line 5) into account, we get the overall complexity $O(nT^2|L|(2e)^{n/2})$. □

Evidently, however, this assessment is grossly overestimated. For instance, we disregard the fact that there can be no valid occurrence of a plan if *any* $n_j = 0$, i.e., the jth column of the ith plan does not match any agent's activity trace. Furthermore, we find empirically that the actual runtimes on large random instances seem to be quite insignificant (see section "Evaluation").

Since each row of E corresponds to one occurrence of a plan, it can be associated with the value of that plan, which we shall call the value of the row. Once E has been produced, the rest of the problem is to search (using Algorithm X) for the maximum-valued set of rows of E such that for each of the Tn columns of E, say c, there is exactly one row, r, in this set with $e_{r,c} = 1$. In other words, we apply Algorithm X to find the best-valued *cover* of the matrix E.

One important benefit of using Algorithm X to solve MAPR is that it provides us with search tree which, in turn, allows us to prune subtrees from our search with *branch and bound*. Algorithm 2 shows our modified version of Knuth's Algorithm X, adapted for branch and bound, to find the highest valued exact cover of the matrix E constructed by Algorithm 1. In order to perform branch and bound we need three important functions. First, let $h(i)$ be a function which maps the row i in the matrix E to the plan from which it was generated. Then, if S is a whole or partial solution (i.e. a list of rows in E), the value of S is given by

$$\texttt{Value}(S) = \Sigma_{s \in S}\, v(h(s))$$

Most importantly, for branch and bound we need a function that gives us an upper bound on the value of the remaining part of any full solution (of which S is a part) that we can achieve by traversing any given subtree. In Algorithm 2, we refer to this function as `BestPossible`. We discuss three different implementations of this function in the next section. Lines 2–5 and 8–10 in Algorithm 2 pertain to branch and bound, but the rest of the lines come directly from Algorithm X.

The function `ChooseColumn` returns the next column that should be covered. It can be defined in a variety of ways, but (Knuth 2000) reports that choosing the column with the fewest number of ones reduces the branching factor quite effectively. In our experiments we have used this heuristic.

Algorithm 1 `Build-E`(M, L)

Input: A trace matrix M (of size $T \times n$) and a plan library L. ***Output:*** A boolean matrix E.

1: $E \leftarrow \emptyset$
2: **for** each plan $P \in L$ **do**
3: $x \leftarrow$ number of rows of P
4: $y \leftarrow$ number of columns of P
5: **for** $r \leftarrow 1 \ldots T - x + 1$ **do**
6: **for** $c \leftarrow 1 \ldots n$ **do**
7: $s_c \leftarrow \begin{bmatrix} m_{r,c} \\ m_{r+1,c} \\ \vdots \\ m_{r+x-1,c} \end{bmatrix}$
8: **end for**
9: **for** each unordered selection of y indices $(i_1, i_2, \ldots, i_y)$ from the agent set $\{1, 2, \ldots, n\}$ such that

$$\begin{bmatrix} s_{i_1} & s_{i_2} & \ldots & s_{i_y} \end{bmatrix} = P$$

do
10: $A \leftarrow 0_{1 \times (Tn)}$
11: **for** $i' \leftarrow i_1 \ldots i_y$ **do**
12: **for** $j' \leftarrow 0 \ldots x - 1$ **do**
13: $A(j'n + i') \leftarrow 1$
14: $E \leftarrow \begin{bmatrix} E \\ A \end{bmatrix}$
15: **end for**
16: **end for**
17: **end for**
18: **end for**
19: **end for**
20: Return E

Another benefit of our approach is that it cleanly separates the hypothesis generation (Algorithm 1) and hypothesis search (Algorithm 2). This allows us to study their complexities separately, and also their relative contributions to the overall runtime.

Bounding Criteria

In Algorithm 2 `BestPossible` can have a variety of definitions producing different tightnesses of the upper bound. We first consider the possibility of calculating `BestPossible` in constant time. One possibility is to precalculate the highest valued integer-partition of all j, $i = 2, \ldots, T \cdot n$, efficiently by dynamic programming. If an integer partition of j is $I(j) = [s_1, s_2, \ldots, s_m]$ (i.e., $s_k \in Z^+$ and $\sum_{k=1}^{m} s_k = j$), then its value is given by

$$v_{IP}(I(j)) = \sum_{k=1}^{m} \max_{p | s_k = r(p) \cdot c(p)} v(p) \qquad (2)$$

Using this value function, we can find the highest value that can be obtained by partitioning j *in any way*, as

$$v^*_{IP}(j) = \begin{cases} \max_{1 \leq k \leq j} \left(v_{IP}([k]) + v^*_{IP}(j-k)\right) & j > 0 \\ 0 & j = 0 \end{cases}$$

Algorithm 2 `Search`(E, S)

```
1: if E = ∅ then
2:    if best_solution_value < Value(S) then
3:       best_solution_value ← Value(S)
4:       best_solution ← S
5:    end if
6:    Return
7: end if
8: best_possible ← Value(S) + BestPossible(E)
9: if best_solution_value > best_possible then
10:   Return
11: end if
12: C ← ChooseColumn(E)
13: for each row such that E(row, C) = 1 do
14:   S ← S ∪ {row}
15:   B ← E
16:   for each col such that E(row, col) = 1 do
17:      Remove col from B.
18:      for each r such that E(r, col) = 1 do
19:         Remove r from B
20:      end for
21:   end for
22:   Search(B, S)
23:   S ← S \ row
24: end for
```

for all $j = 2, \ldots, Tn$ in a bottom-up dynamic programming formulation similar to "rod cutting" (Cormen, Leiserson, and Rivest 1990), that runs in time $O(T^2n^2)$. We thus define `BestPossible` as

$$\texttt{BestPossible}_1(E) = v^*_{IP}(c(E))$$

which is simply a constant time table look-up during the search. Since each choice of a row (i) from E made by branch and bound corresponds to eliminating some columns from E (those columns j that satisfy $E(i, j) = 1$, i.e., covered by row i), and this occurs recursively on E, the recursive process of covering all the columns of E by branch and bound can be seen as some partitioning of $c(E)$. However, even the optimal of such partitions cannot be higher valued than $v^*_{IP}(c(E))$, since the optimal partition would have more constraints (from the MAPR problem) than the ones imposed in equation 2. Therefore the above bounding criterion is sound.

The above scheme, being based on a pre-computation, is rather insensitive to the *actual* plans that correspond to the rows of E. That is, the optimal partition of j, $I^*(j)$ (based on equation 2), may use a set of integers that is very different from the set counterpart of the vector $IP_E = \langle r(p) \cdot c(p) | p = h(x), x = 1, \ldots, r(E) \rangle$ [1], corresponding to the set of the plans, p, actually represented in the (current) matrix E. Therefore, it is possible to redefine equation 2 as

$$v_{IP}(I(j)) = \sum_{k=1}^{m} \max_{p | s_k = r(p) \cdot c(p) \text{ and } s_k \in IP_E} v(p)$$

[1] If two plans, p_1 and p_2 are such that $r(p_1) \cdot c(p_1) = r(p_2) \cdot c(p_2)$, then they produce two entries in IP_E but only one entry in its set counterpart.

This way we impose more constraints on equation 2, but this is still a subset of the constraints that MAPR imposes, hence the soundness of the bounding criterion is preserved. However, this computation can no longer be performed off-line, and thus the dynamic programming needs to be performed at each call to `BestPossible`. Therefore the new version, $\texttt{BestPossible}_2(E)$ will be $O(c^2(E))$ i.e., $O(T^2n^2)$,

A line of attack that is particularly favored by the DLX implementation, is to define

$$\texttt{BestPossible}_3(E) = \sum_{j=1}^{c(E)} \max_{1 \le i \le r(E)} \{v(h(i)) | E(i, j) = 1\} \quad (3)$$

This is advantageous because DLX allows us to find $max_i \{v(h(i)) | E(i, j) = 1\}$ in *constant* time, thus allowing the bound to be calculated in time $O(Tn)$ since E has *at most* Tn columns. The constant time results from the fact that the rows in E are already sorted by v, so that $v(h(1)) \ge v(h(2)) \ge \ldots \ge v(h(r(E)))$, in the hope of finding a high-valued solution early to be able to bound more solutions later. It can be seen that since equation 3 covers each column of (current) E with the highest valued plan that could possibly cover that column, `Value`(S)+`BestPossible`(E) always exceeds the highest valued solution that could possibly match S, and is therefore a feasible pruning criterion. Although the linear complexity of $\texttt{BestPossible}_3$ lies between those of versions 1 (constant) and 2 (quadratic), it is not immediately clear which of the three should be preferred since the relative amounts of pruning achieved by these criteria are unknown. A criterion that is more expensive (per call) than another may still achieve greater pruning and yield lower total runtime. However, our experiments are somewhat inconclusive on this aspect since all criteria seem to only marginally improve on the unpruned version, although we do see that the overall run-time of $\texttt{BestPossible}_2$ is often worse than the other two.

Evaluation

We have run the algorithms on a series of random instances of the MAPR problem, with varying parameters and different pruning criteria. In order to generate each random instance, we first generate a random trace with dimensions $T \times n$, with each element of the trace belonging to the set of possible actions, Σ. Then, we randomly partition the trace matrix, adding the submatrices to the plan library L, so that at least one possible solution exists, and then add z random new plans to the library. These new plans may (or may not) yield a better cover than the one initially produced by partitioning the trace.

In (Banerjee, Kraemer, and Lyle 2010) we have shown that for n agents, MAPR is NP-complete. However, it is unknown how the overall run-time would depend on the other parameters, viz., $|\Sigma|$, T, and $|L|$. In this paper, we have shown that the time complexity of Algorithm 1 depends polynomially on T and $|L|$, but its dependence on $|\Sigma|$ has not been captured. Moreover, it is unclear how expensive Algorithm 1 is compared to Algorithm 2. We fill these gaps through the following experiments. We note that $|L| > z$,

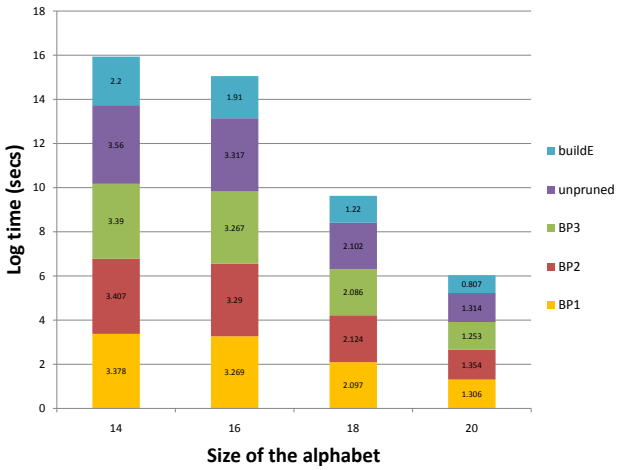

Figure 2: Logarithm of run-times for different $|\Sigma|$, when $n = 30$, $T = 60$ and $z = 50$.

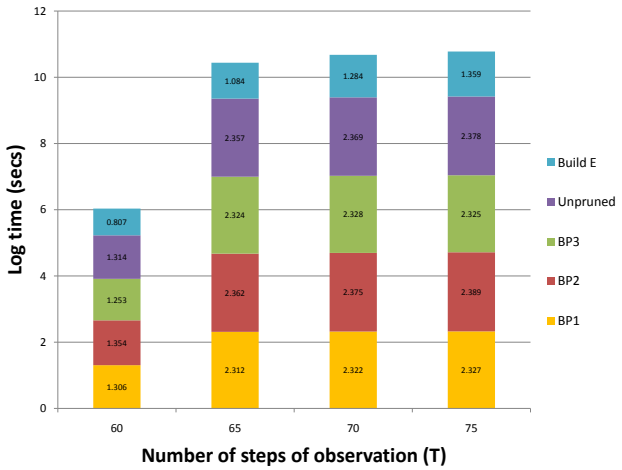

Figure 3: Logarithm of run-times for different T, when $|\Sigma| = 20$, $n = 30$, and $z = 50$.

and z is easier to control in the data generation, therefore we consider the parameter z instead of L. We ran the following experiments on a microway cluster with 10 nodes and 10×4 processors (@ 2.54GHz and 3.86G RAM each), allotting each problem instance to a separate processor.

Through repeated trials, we have found that by setting $n = 30$ $|\Sigma| = 20$, $T = 60$, $z = 50$, not only is the problem reasonably large, but also we get average run-times (over 100 random instances) of 20 seconds, which is an acceptable basis for varying one variable at a time, keeping the others fixed. In the first experiment, we varied $|\Sigma|$ from 20 down to 14 in steps of 2, while the other variables were held fixed at their base values. The *logarithm* of the average run-times (over 100 random instances) for different pruning criteria (including unpruned), as well as the runtime of Algorithm 1 (shown as "Build E") are shown in Figure 2. Here (and in other plots) BP_i refers to $\texttt{BestPossible}_i$. The most interesting finding here is that the overall run-time *descreases exponentially* as $|\Sigma|$ is increased, keeping the other parameters fixed at their base values. This is not unexpected, since a larger number of distinct activities (i.e., larger $|\Sigma|$) tends to make it harder for a randomly generated plan to occur (see Definition 1) in a randomly generated trace, both generated by sampling from Σ. This reduces the average number of occurrences per plan, yielding smaller E, and consequently lower search times as well. This is further validated by the exponentially descreasing time for "Build E" to less than a second in the base setting ($|\Sigma| = 20$) in Figure 2.

Figure 2 also shows that the relative time taken by Algorithm 1 is small compared to that of Algorithm 2. Furthermore, both $\texttt{BestPossible}_1$ and $\texttt{BestPossible}_3$ perform slightly faster than unpruned search, whereas $\texttt{BestPossible}_2$ is slower than unpruned search for smaller overall run-times, clearly due to its quadratic cost and reduced scope for pruning. However, the general conclusion must be that the pruning techniques are at best marginally competitive with unpruned search when the run-times are upto 10^4 seconds. In informal experiments we have found that on instances where unpruned takes significantly longer than 10^4 seconds, the amounts of pruning (by all criteria) are also significant. But data collection from such lengthy runs have been painstakingly slow.

Figure 3 shows the result of varying T from 60 to 75 in steps of 5, with the other parameters held at their base values. The most interesting finding from this experiment is a roughly logarithmic trend in the overall log-run-time, suggesting that the run-time varies polynomially with T. Based on Lemma 1, we expect "Build E" to be polynomial in T, which is validated by the logarithmic trend of the "Build E" bars in Figure 3. Therefore, the major conclusion from this experiment is that not only Algorithm 1 costs polynomially in T, but also *Algorithm 2 costs polynomially in* T. This is a significant result since T is the one parameter that may need to grow unboundedly in generating the observation traces for agents, even when n and $|\Sigma|$ may conceivably remain fixed in a given application.

Figure 4 again shows a similar trend as Figure 3, signifying polynomial dependence of the overall run-time on $|L|$ as well. Interestingly, comparing the two figures, it seems adding more plans to tbe library has a greater impact on the search time than on "Build E". This might be explained as an insignificant growth in the number of rows in E (which affects "Build E"), but a significant qualitative impact on the content of E (which affects search time).

It should be noted that the number of samples on which the averages are based are not exactly 100 in each plot, and neither are the samples unbiased. This is because we eliminated entire runs (all pruning criteria) if any criterion (unpruned or $\texttt{BestPossible}_2$) failed to finish in 10 hours. We found that even under this restriction, we always had 95 or more complete data points per 100 trials, and given that the average run-times are (mostly) significantly less than 10 hours, the resulting bias may be interpreted as (partial) *outlier elimination*.

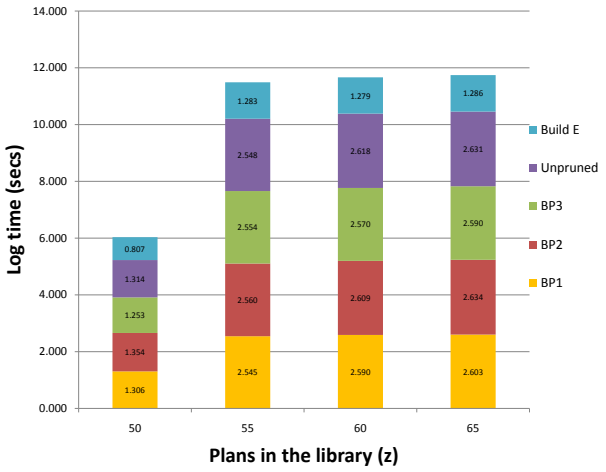

Figure 4: Logarithm of run-times for different z, when $|\Sigma| = 20$, $n = 30$, and $T = 60$.

The variances in the plots were high enough for us to de-emphasize the differences in the run-times between different pruning criteria as statistically insignificant, and instead emphasize the general trends. The variances are high because the randomly generated instances behave very differently, for instance with many smaller sized plans in the library (one agent one step plans in the worst case) E can be very large (thousands of rows). More importantly, the number of potential solutions in the search space can sometimes be extremely large (billions), perhaps due to many independent subproblems which is more related to the content of E rather than its size. In summary, it is likely that 100 samples is too few to establish any strong claims.

Conclusion

We have recently formalized MAPR using a new model and analyzed its complexity. In this paper we have analyzed the algorithms for this model mainly empirically.

We found that the overall run-time decreases exponentially with $|\Sigma|$, while we know that it increases exponentially with n. This means there are settings in which MAPR can be solved efficiently even for a very large number of agents, provided the activities being monitored have a rich variety as well. This is an interesting result, that suggests an alternative way to factor the problem. An existing approach to factoring MAPR along the agent dimension is to consider the social structure of agents (as discussed in related work). Our finding suggests that it could be more beneficial to factor the agent dimension by the diversity of activities instead of a static social structure.

Another important result is that the overall run-time grows only polynomially in T. This, together with the result discussed in the previous paragraph, means that in an application where a fixed set of agents are being observed, the solver can afford to continually observe the agents without being concerned about the number of steps accumulated. Perhaps even 24 years (Drew Jan 10 2010) worth data is not a challenge after all!

Among less significant results, we found that the pruning criteria do not afford significant benefits for small problems, i.e., those that run is 10^4 seconds or less. Since we have seen each criterion perform significant pruning on some such instances but nearly no pruning on others (resulting in greater run-time than the unpruned search on these instances due to their overhead), it would be interesting to characterize the problem structures where each criterion performs well. Such a predictive capability could be applied to exploit the appropriate criteria at the appropriate places in the search tree, although that would also introduce an overhead of its own.

There are many directions for the future. One avenue is to explore problem factoring, and prediction of prunability, based on the observations described above. Moreover, there is a need to improve the pruning criteria before we explore more complex models that allow plan abandonment and interleaving of plan steps. Another avenue is to explore approximation approaches that produce a maximal *packing* i.e., coverage of as much of M as possible with highest valued plans, with no overlap, while possibly leaving out some observations unexplained. Since such an approximation could have a higher total value than the exact cover, it could offer an interesting tradeoff regarding the choice of observations that can be left unexplained.

Acknowledgement of Support

This work was supported in part by a start-up grant from the University of Southern Mississippi.

References

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2007. Towards dynamic tracking of multi-agent teams: An initial report. In *Proceedings of AAAI conference*.

Banerjee, B.; Kraemer, L.; and Lyle, J. 2010. Multi-agent plan recognition: Formalization and algorithms. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*.

Bui, H. 2003. A general model for online probabilistic plan recognition. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI*, 1309–1315.

Castelfranchi, C., and Falcone, R. 1995. From single-agent to multi-agent: Challenges for plan recognition systems. In *Proceedings of the IJCAI-95 Workshop on The Next Generation of Plan Recognition Systems*, 24–32.

Charniak, E., and Goldman, R. 1993. A bayesian model of plan recognition. *Artificial Intelligence* 64:53–79.

Cohen, P.; Perrault, C.; and Allen, J. 1981. *Strategies for Natural Language Processing*. Lawrence Earlbaum Assoc. chapter Beyond question answering.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. Cambridge, MA: MIT Press.

Devaney, M., and Ram, A. 1998. Needles in a haystack: Plan recongition in large spatial domains involving multiple agents. In *Proceedings of AAAI conference*.

Drew, C. (Jan-10) 2010. Military is deluged in intelligence from drones. In *New York Times*.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Htn planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, 1123–1128. AAAI Press.

Geib, C., and Goldman, R. 2003. Recognizing plan/goal abandonment. In *Proc. of IJCAI-03*.

Geib, C. 2004. Assessing the complexity of plan recognition. In *Proc. of AAAI-04*.

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. *Artificial Intelligence* 64:53–79.

Hongeng, S., and Nevatia, R. 2001. Multi-agent event recognition. In *Proceedings of the Eighth IEEE International Conference on Computer Vision*, 84–91 vol.2.

Huber, M., and Durfee, E. 1992. Plan recognition for real-world autonomous robots: Work in progress. In *Working notes of AAAI symposium: Applications of AI to Real-World Autonomous Robots*.

Intille, S., and Bobick, A. 1999. A framework for recognizing multi-agent action from visual evidence. In *Proc. of AAAI*.

Kaminka, G., and Bowling, M. 2002. Towards robust teams with many agents. In *Proc. AAMAS-02*.

Kaminka, G.; Pynadath, D.; and Tambe, M. 2002. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research* 17.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proc. AAAI*.

Knuth, D. E. 2000. Dancing links. In Davies, J.; Roscoe, B.; and Woodcock, J., eds., *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare*, 187–214.

Pynadath, D. V., and Wellman, M. P. 2000. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI2000*, 507–514. Morgan Kaufmann Publishers.

Sidner, C. 1985. Plan parsing for intended response recognition in discourse. *Computational Intelligence* 1(1):1–10.

Sukthankar, G., and Sycara, K. 2006. Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In *Proceedings of AAAI conference*.

Sukthankar, G., and Sycara, K. 2008. Hypothesis pruning and ranking for large plan recognition problems. In *Proc. of AAAI*.

Tambe, M. 1995. Recursive agent and agent-group tracking in a real-time, dynamic environment. In *Proceedings of International Conference on Multiagent Systems*, 368–375.

Tambe, M. 1996. Tracking dynamic team activity. In *Proc. of AAAI*.

Tambe, M. 1997. Towards flexible teamwork. In *Journal of Artificial Intelligence Research*, volume 7, 83–124.

Vilain, M. 1990. Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proc. of AAAI-90*.