Learning the Sequential Coordinated Behavior of Teams from Observations

Gal Kaminka, Mehmet Fidanboylu, Allen Chang, and Manuela Veloso

Computer Science Department
Carnegie Mellon University

Abstract. The area of agent modeling deals with the task of observing other agents and modeling their behavior, in order to predict their future behavior, coordinate with them, assist them, or counter their actions. Typically, agent modeling techniques assume the availability of a plan- or behavior-library, which encodes the full repertoire of expected observed behavior. However, recent applications areas of agent modeling raise challenges to the assumption of such a library, as agent modeling systems are increasingly used in *open* and/or *adversarial* settings, where the behavioral repertoire of the observed agents is unknown at design time. This paper focuses on the challenge of the unsupervised autonomous learning of the sequential behaviors of agents, from observations of their behavior. The techniques we present translate observations of the dynamic, complex, continuous multi-variate world state into a time-series of recognized atomic behaviors. This time-series is then analyzed to find repeating subsequences characterizing each team. We compare two alternative approaches to extracting such characteristic sequences, based on frequency counts and statistical dependencies. Our results indicate that both techniques are able to extract meaningful sequences, and do significantly better than random predictions. However, the statistical dependency approach is able to correctly reject sequences that are frequent, but are due to random co-occurrence of behaviors, rather than to a true sequential dependency between them.

1 Introduction

The area of agent modeling deals with the task of observing other agents and modeling their behavior, in order to predict their future behavior, coordinate with them, assist them, or counter their actions. For instance, agent modeling techniques have been used in intelligent user interfaces [1,2], in virtual environments for training [3,4], and in coaching and team-training [5]. Typically, agent modeling techniques assume the availability of a plan- or behavior- library, which encodes the complete repertoire of expected observed behavior. The agent modeling techniques typically focus on matching the observations with library prototypes, and drawing inferences based on successful matches.

Recent applications areas of agent modeling raise challenges to the assumption of an available complete and correct behavior library. Agent observers are increasingly used in *open* and/or *adversarial* settings, where the behavioral repertoire of the observed agents are unknown at design time. Thus agents that are engaged in modeling must be able to autonomously acquire the models used in agent-modeling prior to using them in the agent modeling tasks. While there have been reports on successful techniques that begin to address of the task of constructing the behavior library (e.g., [6,2]), key challenges remain open.

In particular, this paper focuses on the challenge of autonomous unsupervised learning of the sequential behavior of agents and teams from observations of their behavior. Our approach combines plan-recognition techniques and symbolic time-series sequence identification to translate raw multi-agent, multi-variate observations of a dynamic, complex environment, into a set of sequential behaviors that are characteristic of the team in question. Our approach is appropriate for domains in which recognizing the basic behaviors of agents and/or teams is a tractable task, but the space of sequential combinations of these behaviors is practically unexploreable.

In our approach, the observing agent first uses a set of simple basic-behavior recognizers to parse the stream of raw observations and translate it into a stream of recognized behaviors, which we call *events*. Unlike previous approaches, the construction of such recognizers does not attempt to cover the entire multi-agent behavioral repertoire of the team. Instead, the behavior recognizers focus on recognizing only the most basic, atomic behaviors, without addressing how they are combined together.

The stream of raw multi-variate, multi-agent observations is therefore turned into a stream of labeled events. The recognized behavior stream is first broken into segments, each containing the uninterrupted consecutive events of only one team. Multiple segment are collected together in a *trie*-based data structure [7], that allows for on-line, incremental identification of important and predictive sequences (this data-structure is explained in Section 3.2).

For instance, given a stream of raw observations about soccer players' position and orientation, and the position of the ball, the behavior recognizers may produce a stream of recognized passes, dribbles, and other soccer-playing behaviors, for instance

$$Pass(Player_1, Player_2) \rightarrow Pass(Player_2, Player_3) \rightarrow Dribble(Player_3) \rightarrow \ldots$$

Identification of important (sub)sequences in this resulting stream can be difficult in dynamic settings. Ideally, the segments contain examples of *coordinated* sequential behavior, where the execution of one atomic behavior is dependent on the prior execution of another behavior, possibly by a different agent. However, such coordinated sequences are sometimes disrupted by the need to respond to dynamic changes in the environment or the behavior of other agents. Thus the resulting segments are often noisy, containing only portions of the true sequences, intermixed with possibly frequent observations of reactive behaviors that are not part of any sequence. For example, in the sequence above, the $Dribble$ event may have not been planned part of the team's sequence, and may have been independent of the coordinated two-pass sequence that preceded it. In other words, the $Dribble$ was possibly executed by $Player_3$ to respond dynamically to the situation that it faced, independently of whom passed the ball to it. In this case, a subsequence $Pass(Player_2, Player_3) \leftarrow Dribble(Player_3)$ would not have been a useful sequence to discover—it reveals nothing about the coordinated behavior of the team.

We present a technique for processing such a stream of events, building on earlier work by Howe and Cohen [8], and extending it in novel ways. The technique we present rejects such false sequences, by uncovering sequential statistical dependencies between atomic observed behaviors of agents. We demonstrate the efficacy of our approach in learning sequential behavior of several RoboCup simulation teams from observations, and compare the results of using frequency counts [9] and statistical dependency detection [8], two important sequence-mining techniques, in identifying important sequences. We show that while the two techniques are both significantly better than random selection, statistical dependency detection has important benefits in being able to distinguish causal dependencies between observed behaviors, from frequent co-occurrence, due to chance, of independent behaviors.

This paper is organized as follows. Section 2 presents the motivation for our approach, and related work. Section 3 presents the behavior recognizers and the trie-based data structure used in determining subsequences. Section 4presents the dependency-detection method and frequency counts. Section 5 discusses results of experiments, validating our approach. Section 6 presents conclusions and directions for future work.

2 Learning Coordinated Sequential Behavior in Open, Adversarial Settings

Most agent modeling techniques assume that an accurate behavior library is available to the observing agent such that it has to match its observations with the library, identify successful matches, and possibly select between them. As noted, recent applications of agent modeling raise challenges to this assumption, as agents are increasingly applied in settings that are *open* and/or *adversarial*. In such settings, the behavioral repertoire of observed agents is unknown to the observer. Indeed, such is the case in our application area in robotic soccer. We are developing a coach that is able to improve the performance of a team by analyzing its behavior, and that of the opponent [5]. Since both the coached team and the opponent are unknown at the time of the design, an accurate behavior library describing their expected behavior is unavailable to the coach agent. In particular, a key challenge here is to determine what sequential behaviors best characterize a team, and best serve for predicting its goals and its future behaviors.

Unfortunately, this challenge of constructing an agent-modeling library from observed sequences has only partially been addressed. Bauer [6] presents an approach using clustering to find clusters of similar sequences which are then fed into a specialized supervised plan-acquisition algorithm that determines abstract plans from these clusters. This approach does not attempt to rank sequences by frequency or discover statistical dependencies. Also it has not been applied in multi-variate continuous domains: It assumes that actions are discrete and are executed by a single agent. In addition, Bauer's method depends on the manual selection of a *clustering factor* parameter (in the range $[0, 1]$), a process which we seek to avoid.

Davidson and Hirsh present the IPAM algorithm [2] that successfully predicts a user's next shell command from prior commands, and accumulates such predictions from unsupervised training data. However, IPAM focuses solely on prediction of the next command, and is unable to support modeling in service of more sophisticated tasks that requires analysis. Such analysis, however, is very much needed in the coaching application which motivates our research. Zaki et al. [9] use an efficient sequence clustering algorithm to determine causes for failures in plan execution, by analyzing sequences of plans ending with failure and success. Their approach focuses on labeled examples, and assume that spurious or irrelevant details may be observed, and must therefore be filtered out from the result. Their approach relies heavily on frequency counts to assess the significance of discovered sequences, and on the user manually fine-tuning frequency and support thresholds. In contrast, we seek sequence detection with unlabeled data, and wish to avoid any manual threshold settings.

Howe and Somlo [10] present an approach, called dependency-detection, to identifying sequences in an incoming data stream, which ignores frequency counts in favor of statistical dependency checks. They thus discover sub-sequences which are statistically significant, in that they appear too often compared to random chance. However, their approach is only applicable to categorical time-series, and cannot be used as is to discover sequential patterns in multi-variate, multi-agent, continuous domains.

To address multi-variate, multi-agent, continuous data, we investigate an approach whereby the basic behaviors of a team are recognized by manually-constructed set of recognizer, which parses a stream of multi-variate, multi-agent observations and translates it to a stream of recognized *atomic* behaviors, annotated by the agents taking part in them. We note that in many domains, the different basic behaviors of agents and teams are often easily described by domain experts, and are often easily recognizable. For instance, in the soccer domain, behaviors such as passing, shooting to the goal, intercepting the ball, etc. are all easily recognized. However, teams differ in how they sequence these basic behaviors, to form coordinated activities which attempt to achieve the team's goals.

This stream of annotated behaviors provides a symbolic abstraction of the raw observations. It is a categorical time-series, amenable to sequence learning of the type investigated by Howe and Somlo [10]. However, while they focused on extracting patterns limited by length, our algorithm extracts all sequences of any length, and facilitates prediction and analysis based on either statistical dependency checks, or on frequency counts. Indeed, one of the contribution of this paper is its comparison of these two techniques and their relative performance.

3 Discovering Sequences of Behavior

This section provides a detailed description of the approach we present in this paper. Our approach takes several stages. First, the raw multi-variate observation stream is turned into a stream of recognized atomic behaviors. This is achieved by utilizing a set of atomic behavior recognizers that parse the observations, and determine what behavior is being executed by players at any given moment. This process is described in Section 3.1. The resulting event stream is segmented to distinguish one team's behaviors from its opponent, and these segments are stored in a trie for analysis (Section 3.2). The next section will describe methods for identifying important sequences using this representation (Section 4).

3.1 Atomic Behavior Recognition

The incoming observation stream is a multi-variate time-series of sensor readings. Each observation is a snapshot of the sensors. In our application, these report state of the soccer field including the coordinate position and orientation of each player and its identification (team and uniform number), the position of the ball, and the state of the game (e.g., goal scored, normal play, free kick, off-side). As such, these sensor snapshots do not offer any information about actions taken by agents, only about the state of the environment and agents embodied in it. However, by contrasting consecutive snapshots one can estimate the movement and rotation velocity of the players and ball, and from those, infer (with some uncertainty) what actions have been taken. For instance, the observer may be able to infer that a player had kicked the ball, based on observations of the changes to the ball's velocity vector that had occurred when the ball was in proximity to the player.

The multi-variate stream is fed, one raw observation after another, into a set of independent atomic-behavior recognizers, working in parallel. These opportunistically parse the observed world-state snapshots and extract from them a summary: a series of recognized atomic behaviors, that each summarizes the team activity during a time interval.

Each recognizer is built as a logical combination of quantifiers and predicates, indexed by time, testing spatial relationships between the ball and player, and organizational relationships between players. Here are a few of the ones we used:

- $Possessor(p, t)$ true if, as of time t, player p is able to kick the ball, i.e., it is within its kicking range. Note this may be true of several players at time t.
- $Owner(p, t)$ true if, as of time t, player p was the last player to kick the ball. This will hold true after the ball leaves the control of the player, until it is either kickable by another player (i.e., another player becomes a possessor), or the game mode changes (e.g., a goal is scored or the ball goes outside of the field).
- $Receiver(p, t)$ true if, as of time t, player p was within the region defined by an angle of $10°$ centered around the ray defined by the velocity vector of the ball. In other words, true if the ball was possibly aimed at the player. A receiver may be of any team.

- $Teammate(p, q, T)$ true if players p, q are both on team T.
- $Moved(p, t_1, t_2)$ is true if the player p has moved more than 2 meters between time t_1 and time t_2.

For instance, a pass between two members of team T is recognized as occurring between times $t-k$ and t (where k is a positive integer) if the ball has been kicked by one player p at time $t-k$, and then the ball came into the control of *another* player q at time t, where p, q are on the same team. A final condition is that during the interval $[t-k+1, t-1]$ the ball should not have been under the control of any player, nor the game play interrupted. Thus a pass would be recognized if the following holds: (1) the ball possessor at time $t-k$ and the ball possessor at time t are teammates in T, but (2) are not the same player; (3) the ball possessor at time $t-k$ is was the ball owner during the interval $[t-k+1, t-1]$, but was no longer in control of the ball:

$$\begin{array}{ll} & Possessor(q,t) \land Possessor(p,t-k) \\ \land & \neg Possessor(q,t-k) \land Teammate(p,q,T) \\ \land & \forall i \in [t-k+1, t-1] \Rightarrow (Owner(p,i) \land \neg Possessor(p,i)) \end{array}$$

Here is another example. An atomic dribble is recognized when a player runs while maintaining the ball under its control, or when it shoots the ball a bit forward and then runs up to catch it. Thus the following conditions must hold: (1) a player d was a possessor at time $t-k$, and is a possessor at time t; (2) d moved more than a specified minimum distance between $t-k$ and t; and either (3a) d remained possessor during $[t-k, t-1]$, or (3b) d was owner during $[t-k, t-1]$:

$$\begin{array}{ll} & Possessor(d,t-k) \land Possessor(d,t) \\ \land & Moved(d,t-k,t) \\ \land & \forall i \in [t-k, t-1] \Rightarrow (Owner(d,i) \lor Possessor(d,i)) \end{array}$$

Each such behavior recognizer is essentially a rule that captures an atomic behavior, typically of only one or two teammates, and ignores most of what is happening in the world. Its design obviously builds much on the expertise of the domain experts, and their ability to describe the behavior and differentiate it from others using the predicates.

As is commonly recognized in the literature, observations can often lead to ambiguous interpretations. For instance, the ball velocity vector may be a complex function of the kick actions taken by several players, environmental noise, and the timing of agents' actions. Thus the actual kicker or kickers may not be known with certainty: The observer may only be able to narrow it down to three possible players, for instance. It is therefore possible that recognized behaviors may have overlapping times.

However, we believe that in many domains, much like in the soccer domain, simple atomic behaviors are easily describable by domain experts, and would be easily recognizable with little uncertainty. Our hypothesis is that in these domains, it is the sequential combinations of such behaviors that gives rise to the complexity of the recognition task, and the uncertainty that associates it. Indeed, these sequential combinations of these atomic behaviors by a team of agents that are difficult to describe in advance, since the open nature of the applications causes a variety of sequences to appear. For instance, the basic Unix shell commands are common to all users. However, users differ in how they interact with the shell using these commands [2].

Furthermore, since behaviors typically span over several actions and game states taken in aggregate, uncertainty can often be reduced by looking at the results of an hypothesized action several time-steps after it has taken place. For instance, suppose two players of opposing teams were both within a kicking range of the ball just before it changed its velocity. Though it cannot be decided from this observation snapshot which one (if any) has kicked the ball, if the ball is observed to be headed towards a teammate, than a pass could be recognized despite the uncertainty about the kicker. While it is possible that this interpretation is wrong, our results have shown that this is only rarely the case[1]. Our system treats any ambiguity in recognized behaviors using a set of domain-dependent heuristic rules, that prefer specific interpretations over other, and preprocess the stream of recognized behaviors. For instance, an uninterrupted sequence of atomic dribbles is combined together into a longer-duration recognized dribble.

Using the recognizers and these heuristic rules, the stream of world state observation is to turned into a steam of symbols called *events*, each denoting a recognized atomic behavior. For instance, a resulting event stream may look like:

$$Pass(Player_1, Player_2) \rightarrow Pass(Player_2, Player_3) \rightarrow Dribble(Player_3) \rightarrow \ldots$$
$$\ldots \rightarrow Pass(Player_3, Player_4) \rightarrow Intercept(Player_{16}) \rightarrow Pass(Player_{16}, Player_{20}) \rightarrow \ldots$$

3.2 Storing Sequences in a Trie

The event stream is processed in several stages to discover sequences of behaviors that characterize the observed teams. First, the event stream is segmented by team, such that the event sequence of one team is separated from the events of its opponent team. This creates multiple segments for each team, each segment composed of an uninterrupted sequence of recognized behaviors that were executed by the team in question. The next stage stores these segments in a trie (one for each team), such that all possible subsequences are easily accessible and explicitly represented. Finally, the trie is traversed to calculate statistical dependencies between events and extract useful subsequences.

We begin by describing the segmentation process. Every event—recognized atomic behavior—is annotated by the agents that take part in the recognized

[1] We point the reader to [11] for a more systematic method of recognizing such coordinated activities, and to [12] for an approach dealing explicitly with uncertainty in recognizing behaviors

activity. Based on the agents that are involved, the events can be categorized as describing one of three parties to the game: the two teams and the referee. All consecutive events belonging to one party are grouped together, in order, into a segment. Each segment is therefore made from a series of uninterrupted activities of only one team (or the referee).

For instance, Figure 1 presents the segmentation of a portion of an event stream. The resulting segments are stored and processed separately, to focus the learning on examples of only one team at a time. In other words, separate models are learned for each team. From this point on, our description will focus on the learning task as it is carried out for one team.

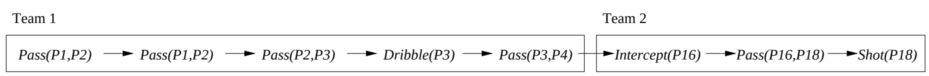

Fig. 1. A sample segmentation of a stream of events, based on agent team membership.

The learner is therefore presented with multiple segments, each composed of a number of events, in order, describing an uninterrupted sequence of behavior. Ideally, each such sequence is an example of a well-executed pre-planned coordinated activity leading to a goal. However, though agents may try to carry out pre-planned coordinated activities, their behavior in the simulated environment has to address the dynamic, complex nature of the game, the autonomy of their teammates (each with its own limited sensing and noisy actions) and the behavior of the opponents. Thus, the segments are very rarely the same.

Each segment represents the emergent results of complex interactions between teammates, opponents, and the dynamic, noisy environment in which all of them are embodied. Thus each segment may contain examples of coordinated sequential behavior, but also examples of independent responses to dynamic situations, which are not sequentially dependent on prior decisions by the agents. With sufficient examples, one hopes to uncover repeating patterns of behavior, and capture significant pieces of the team's coordinated plans.

To find such repeating sequences, the learner stores all segments in a trie [7], inserting each segment, and each of its suffixes. A trie is a tree-like data structure, which efficiently stores sequences such that duplicated sub-sequences are stored only once, but in a way that allows keeping a count of how many times they had appeared. In the trie, every node represents an event, and the node's children represent events that have appeared following this event. Thus a path from the root to a node represents a (sub)sequence that has appeared in at least one, but possibly more, segments. An entire segment would therefore be represented as a complete path through the trie, root node to leaf. Each node keeps track of the number of times an event has been inserted on to it, and the number of times a segment ended with this node. These two values are called the node's *count* and the node's *end-count*, and are marked $C(x)$ and $N(x)$ for a node x.

For example, suppose the learner has seen two segments, $ABCA$ and $BCAD$, where A, B, C, D stand for different events. Suppose that it is starting with an empty trie. It would first insert $ABCA$ into it, resulting in a trie in Figure 2-a. It would then insert the three remaining suffixes of $ABCA$: $\{BCA, CA, A\}$, resulting in the trie in Figure 2-b. Note that the insertion of the suffix A does not entail creating a new node. Instead, the count in the top node representing A is updated to 2. Next, the learner would insert the next segment and its suffixes, $\{BCAD, CAD, AD, D\}$, into the trie, resulting in the trie in Figure 2-c. Note that in this final trie, the most common single-event sequence is A, the most common two-event sequence is CA and the most common three-event sequence is BCA.

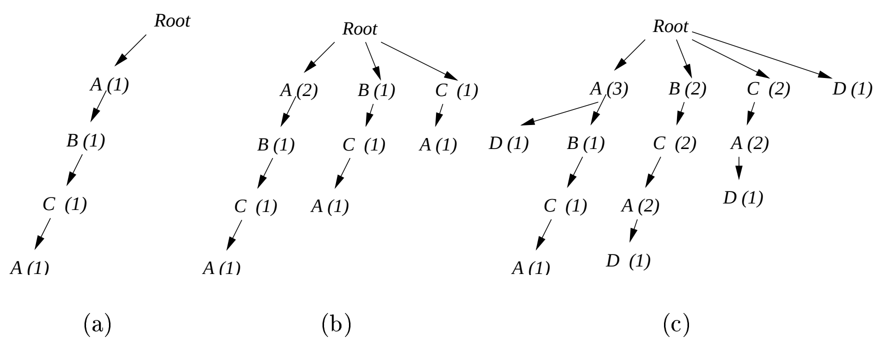

Fig. 2. A trie in construction. Numbers in parentheses are the node counts.

4 Sequential Behavior Identification

Two methods have been discussed in the literature for evaluating the significance of subsequences. The first one, frequency counting, determines the significance of sub-sequences from their frequency in the data [9]. The second method uses statistical dependency checks to determine sequences whose constituent events are deemed statistically dependent on each other; that is, their co-appearance is not due to chance [10]. While the frequency-count method has the benefit of simplicity and intuitive appeal, and the second method has better filtering of sequences that appear common, but in fact are not characteristic of a team. We describe these two methods in detail below, and show how the trie representation can be used efficiently for either method.

We begin by describing how the trie representation can be used for frequency counting. As previously discussed (Figure 2), the simple insertion of the two segments $ABCA$ and $BCAD$ (with their suffixes) almost immediately reveals frequent sub-sequences of different lengths: A (three times), CA (twice), and

BCA (twice). The count in each node can therefore be used to calculate frequencies of different sequences: A count in a node is exactly the number of times the sequence beginning with the root and ending with the node has appeared in the data. A simple traversal through the trie can determine all high-frequency sequences at any given length. However, there is a difficulty with this method, in that it may incorrectly determine that sequences are significant even when in fact they are due to common, but independent, co-occurrence of two highly-frequent events (see a detailed example in Section 5).

Fortunately, a second method for evaluating the significance of sequences exists. This method is called *Dependency Detection*, and was first used by Howe and Cohen in analyzing how plan failures were effected by recovery actions in the Phoenix planner [8]. This method works by statistically testing for the a dependency between a prefix (a sequence containing at least one event) and an event that follows it. In other words, it tests whether the co-occurrence of the prefix and the event is due to chance.

This is achieved by constructing a 2×2 contingency table, relating four different frequency counts (Table 1): (a) the number of times the prefix was followed by the event (n_1); (b) the number of times the prefix was followed by a different event (n_2); (c) the number of times a different prefix (of the same length) was followed by the event (n_3); and (d) the number of times a different prefix was followed by a different event (n_4).

	X	Not X	
Prefix	n_1	n_2	Prefix margin
Not Prefix	n_3	n_4	Not-Prefix margin
	X margin	Not-X margin	Table total

Table 1. A contingency table relating a prefix and a single event. n_1, n_2, n_3 and n_4 are frequency counts that are derived from the trie.

The prefix margin is simply the number of times the prefix was seen (that is, it is equal to $n_1 + n_2$). The not-prefix margin is similarly $n_3 + n_4$. The X margin stands for the number of times X was seen, independently of the prefix, and is $n_1 + n_3$, the not-x margin is $n_2 + n_4$. The table total is $\sum_{i=1}^{4} n_i$. Once the counts are known for a given prefix and an event X, a statistical test can be applied to determine the likelihood that X depends on its prefix, i.e., that a sequence composed of a prefix followed by X is unlikely to be due to a random co-occurrence of the prefix and of X. One such test is the G statistic test, which is calculated as follows

$$G = 2 \sum_{i=1}^{4} n_i log(\frac{n_i}{E_i})$$

where E_i is the *expected frequency* in each cell. These expected frequencies are calculated by multiplying the margins and dividing by the table total:

$$
\begin{aligned}
E_1 &= (Prefix\,margin) * (X\,margin) / (Table\,Total) \\
E_2 &= (Prefix\,margin) * (not\,X\,margin) / (Table\,Total) \\
E_3 &= (not\,Prefix\,margin) * (X\,margin) / (Table\,Total) \\
E_4 &= (not\,Prefix\,margin) * (not\,X\,margin) / (Table\,Total)
\end{aligned}
$$

The G statistic test is a well-known statistical dependency test, similar in nature to the familiar chi-square test. The result of the computation is a positive number, and the greater it is, the greater the likelihood that the prefix and X are dependent on each other. That is, X is more likely to appear after the prefix, compared to random chance[2] One can establish the significance of this dependency, by consulting statistical tables, however, in general, we use it to rank hypothesized sequences, and thus simply take greater values to signify stronger dependencies.

With the addition of one data-structure, the trie-based representation supports calculation of the G statistic for each node, i.e., for any sub-sequence. This additional data structure is a two-dimensional table that records, for each sequence length and event, how many events of the given length ended with this event. This table can be updated during the insertion of each event into the trie, with access time $O(1)$. For a given event e and sequence length l, we denote this number by $\alpha(e, l)$.

Given a node x, representing a subsequence ending with the event e, we use the node's count $C(x)$ and end-count $N(x)$ and the event-length entries $\alpha(e, l)$, to calculate its associated G statistic, which is the G rank for the sequence beginning with the root and ending with the node. The following algorithm recursively calculates this for all nodes in a trie, when called with the root node and depth 0. Once the algorithm has calculated a G value for every node, a traversal of the trie can sort sub-sequences by their G value, thus ranking hypothesized sequences, and highlighting those that are likely to be indicative of coordinated sequential behavior.

Algorithm 1 CALCG(TRIENODE T, DEPTH D)

$s \leftarrow \sum_e \alpha(e, D)$ {Calculate the total number of strings of the depth D}
for all children X of T **do**
    $n_1 \leftarrow C(X)$, $smargin \leftarrow C(T) - N(T)$
    $n_2 \leftarrow smargin - n_1$
    $n_3 \leftarrow \alpha(X, D) - n_1$
    $n_4 \leftarrow s - smargin - n_3$
    Calculate margins and expected frequencies E_i
    $G \leftarrow 2 \sum_{i=1}^{4} n_i log(\frac{n_i}{E_i})$ {This is child X's G}
    CALCG(X, $D + 1$)

[2] Negative dependencies (where X is less likely to appear after the prefix) are also discovered, but are filtered out in our approach

5 Experiments and Results

To evaluate our approach, we have fully implemented a system that observes recordings (logs) of RoboCup soccer simulation games, and is able to identify and extract important sequences of coordinated team behaviors, for the two opposing teams. The system is able to identify 16 different events, corresponding to basic behaviors by agents and game-state changes (such as referee decisions or goals scored). It is able to learn and identify sequences from only a single game or from a number of games, and outputs all the sequences found for each team, together with their frequency counts and with their G statistic ranks.

We have conducted several experiments using this system, intended to test the ability of the different techniques to identify sequences that were characteristic of the team. This was evaluated by using each technique to pick the top 10, 25, 50 and 100 sequences for an observed team, on training data. Then, each technique was used to pick the top 10, 25, 50, and 100 sequences out of test data, for the same observed team. Finally, the size of the overlap in these selections was computed. The hypothesis underlying this test was that greater overlap between the selections based on the training data and the test data, would indicate greater ability to pick sequences that were characteristic of the team, rather than arbitrary.To provide a baseline for this task, random selection of sequences was also used.

This set of experiments used a 5-fold cross-validation design, conducted on a series of 75 games conducted between the Andhill97 team [13] and itself, all using the exact same settings, down to machines and network settings. Andhill was chosen because of its simplicity—we believed that its game play will indeed contain repeating patterns. The games were arbitrarily divided into 5 groups, each containing 15 games. In each experiment, 4 groups (60 games) were used as a training set, and the remaining group (15 games) used as a test set. Five experiments were run, each one using a different training and test groups. In each experiment, a trie for the *left* team was built and maintained, using all observed behavior segments from the games in the training set. Frequency counts and G statistics were then calculated for all nodes in the trie. On average, 4600 different sub-sequences were found for each of the training sets (left team only), and 1200 sub-sequences were found for each test set.

The results of this first set of experiments were very surprising, with frequency counts having an accuracy of 90%-100% in all ranges, and the G statistic method having only a 30%, 51%, 70% and 86% success rate in the top 10, top 25, top 50 and top 100 tests, respectively. However, we quickly found out that the high accuracy for the frequency counts was mostly due to it picking, as the most frequent, sequences containing only a single event. Since single events often occur much more frequently than any sequences they are part of, they were likely to be present in the test set as well. In contrast the G statistic technique flagged two-event and three-event sequences as being representative of the team.

This initial result of course made us reconsider the usefulness of the frequency-count technique, since it was obvious that its result were simply not indicative of any real sequential behavior in the observed team. To make it comparable to the G statistic technique, we restricted the considered length to two events and above, and re-ran the test. The results of this second set of experiments are presented in Figure 3. The Y-axis in Figure 3 shows the percentage of sequences picked by each technique, that were included both in the training set and the test set. The X-axis shows the three different ranges of prediction that were tested: Top 10 sequences, top 25, top 50, and top 100. For each of these categories, three bars are shown, contrasting the performance of the frequency counts with that of G statistic and random selection. The error bars on each column show the minimum and maximum results across the 5 test.

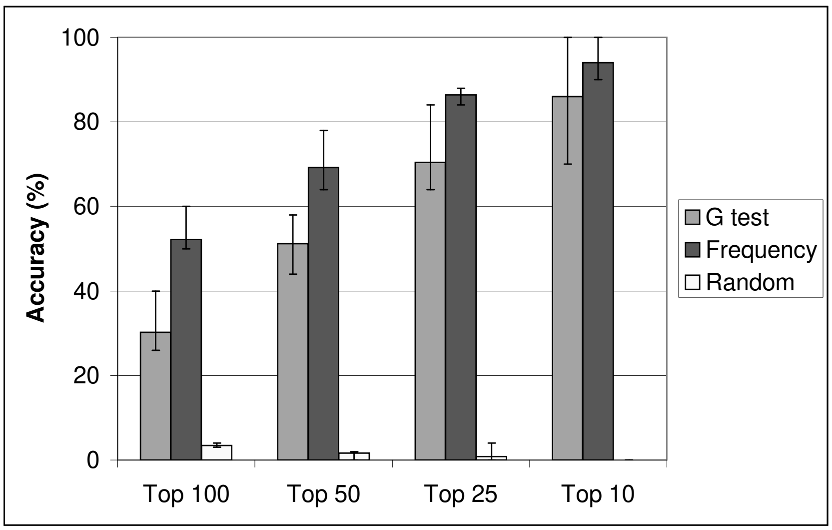

Fig. 3. Prediction accuracy using random selection, G statistics, and frequency counts.

The figure shows that both frequency counts and G do significantly better than random selection at predicting what sequences will be picked in the test set. Indeed, random selection success is almost nil. Frequency counts do better than G. However, the advantage for frequency counts grows weaker with narrower selection ranges, i.e., with the selection becoming more critical for evaluation. Furthermore, as the error bars show, the range of performance for frequency counts is well within the range of performance for G for the top-10 selection.

Building on our experience with misleading results with frequency counts, we examined closely the sequences selected by the two techniques, and found interesting discrepancies. In particular, highly frequent two-event sequences found by frequency counts sometimes had relatively low G rankings. Closer examination showed that indeed, the events in these sequences were not always dependent on each other. For instance, player 7 in the observed team sometimes chose the *Clear-Ball* behavior (kicking it outside or to a free location on the field, essentially giving it up). Since it was an active player in many of the games, sequences involving its recognized behavior of clearing the ball were all fairly frequent. Some of the training sets showed that when player number 7 plays *Keepaway* (i.e., keeps the ball to itself, away from opponents, but not moving forward), then the next behavior it is likely to display is clearing of the ball. Other sequences that involved player 7 clearing the ball showed that in fact this Clear-Ball behavior also commonly occurred after the ball has been passed to player 7 by his teammates (regardless of whom passed), or if player 7 dribbled the ball first.

The G rankings for the same sequences were considerably different. The G method indicated that indeed player 7 is likely to clear the ball when it first plays keepaway. However, all other sequences received much lower rankings—often not even in the top 100. The reason for this difference is that the G statistic judged that the Clear-Ball behavior was not dependent on the passes leading to it, but was only dependent on whether player 7 played keepaway. In other words, the G technique, unlike the frequency counts, correctly identified that player 7 was *not* any more likely to clear the ball if the ball was passed to it by a teammate. However, player 7 was indeed likely to clear the ball if it first attempted to keep it away from opponents. The G technique was thus able to correctly distinguish a sequential pattern that was not due to random co-occurrence, but instead was due to a particular sequence of executed behaviors.

6 Conclusions and Future Work

We presented a hybrid approach to learning the coordinated sequential behavior of teams, from a time-series of continuous multi-variate observations, of multiple interacting agents. The hypothesis underlying this approach is that such a complex time-series can be parsed and transformed into a single-variable categorical time-series, using a set of behavior recognizers that focus only on recognizing simple, basic, behaviors of the agents. Once the categorical time-series is available, it is segmented and stored in a trie, an efficient representation amenable to statistical analysis of the data.

We demonstrated the usefulness of the trie representation in supporting two important methods of statistical analysis—frequency counting and statistical dependency detection—in service of discovering important sequences of behaviors. We evaluated these two techniques in a rigorous set of experiments, and concluded that statistical dependency detection may be more suitable to discovering coordinated sequential behavior, since it is able to reject frequently observed sequences whose constituent behaviors have co-occurred due to chance.

We are currently investigating ways to build on our approach in several directions. First, we are investigating ways of using both frequency counts and the G rankings on-line, such that predictions of the next k events can be made, based on an observed prefix. Second, we are investigating a method for transforming the discovered sequences into a graphical model, essentially a probabilistic finite state automata, that can be used for deeper analysis of team behavior. In this task, we are aided by prior work by Howe and Somlo [10].

References

1. Lesh, N., Rich, C., Sidner, C.L.: Using plan recognition in human-computer collaboration. In: Proceedings of the International Conference on User Modelling (UM-99), Banff, Canada (1999)
2. Davidson, B., Hirsh, H.: Probabilistic online action prediction. In: Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments. (1998)
3. Rickel, J., Johnson, W.L.: Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. Applied Artificial Intelligence **13** (1999) 343–382
4. Tambe, M., Rosenbloom, P.S.: RESC: An approach to agent tracking in a real-time, dynamic environment. In: Proceedings of the International Joint Conference on Artificial Intelligence. (1995)
5. Riley, P., Veloso, M., Kaminka, G.: An empirical study of coaching. In: Proceedings of Distributed Autonomous Robotic Systems 6, Springer-Verlag (2002) (to appear).
6. Bauer, M.: From interaction data to plan libraries: A clustering approach. In: Proceedings of the International Joint Conference on Artificial Intelligence. Volume 2., Stockholm, Sweden, Morgan-Kaufman Publishers, Inc (1999) 962–967
7. Knuth, D.E.: Sorting and Searching. Volume 3 of The art of computer programming. Addison-Wesley (1973)
8. Howe, A.E., Cohen, P.R.: Understanding planner behavior. Artificial Intelligence **76** (1995) 125–166
9. Zaki, M., Lesh, N., Ogihara, M.: Planmine: Sequence mining for plan failures. In: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, AAAI Press (1998) 369–374
10. Howe, A.E., Somlo, G.L.: Modeling intelligent system execution as state-transition diagrams to support debugging. In: Proceedings of the Second International Workshop on Automated Debugging. (1997)
11. Wendler, J., Kaminka, G.A., Veloso, M.: Automatically improving team cooperation by applying coordination models. In: The AAAI Fall symposium on Intent Inference for Collaborative Tasks, AAAI Press (2001)
12. Han, K., Veloso, M.: Automated robot behavior recognition applied to robotic soccer. In: Proceedings of the IJCAI-99 Workshop on Team Behavior and Plan-Recognition. (1999) Also appears in Proceedings of the 9th International Symposium of Robotics Research (ISSR-99).
13. Ando, T.: Refinement of soccer agents' positions using reinforcement learning. In Kitano, H., ed.: RoboCup-97: Robot soccer world cup I. Volume 1395 of LNAI. Springer-verlag (1998) 373–388