Incremental, Team-Oriented Activity and Plan Recognition for Software Agents

Daniele Masato\textsuperscript{1}, Timothy J. Norman\textsuperscript{1}, Wamberto W. Vasconcelos\textsuperscript{1}, and Katia Sycara\textsuperscript{2}

\textsuperscript{1} Department of Computing Science, University of Aberdeen, Scotland, UK
\textsuperscript{2} Robotics Institute, Carnegie Mellon University, Pittsburgh, PA US

Abstract. Activity and plan recognition are well-developed research fields, where much work has been done. There has been, however, almost no effort in combining both to create intelligent software that can effectively support the decision-making process faced by human teams enacting a joint plan. In this paper, we present an architecture to integrate activity recognition and plan recognition using a team-oriented, incremental and distributed approach. In this way intelligent software assistants will be able to interact effectively in human-agent teams, and offer helpful and meaningful support to humans during plan enactment.

Keywords: Software agents, Activity Recognition, Plan Recognition, Incremental, Team-oriented, Distributed

1 Introduction

Activity and plan recognition can be combined to create intelligent software that can effectively support the decision-making process that human teams face when performing a joint plan. This is particularly desirable when humans are working under stress conditions in an uncertain environment. In these situations, they may lose awareness of the plan progress because of cognitive overload, geographical distance, or because there has been limited time for co-training [12]. In this paper, we present a software agent architecture that will support humans in maintaining synchronisation between plan components when different components are assigned to different teams. Agents associated with each team will recognise the teams’ activities during execution, given the positions of team members in the environment. This continuous monitoring will enable agents to match recognised activities with the plan components being executed, hence to keep track of the plan progress and to detect potential deviations from an established course of action. The rest of this document is structured as follows:

\* This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation hereon.
Section 2 defines activity and plan recognition and reviews work related to ours; Section 3 presents our agent architecture; Sections 4 and 5 describe our approach to achieving incremental recognition; Section 6 presents some early results on the evaluation of our activity recogniser. Finally, Section 7 gives details on how we intend to improve the recogniser, then summarises and concludes this paper.

2 Related Work

2.1 Activity Recognition

Activity recognition concerns the problem of identifying a single action performed by a subject according to collected observations. Its simplest form consists in classifying a sequence of observations \( x = (x_1, \ldots, x_T) \) collected over time \( 1 \leq t \leq T \) in order to obtain a sequence of corresponding labels \( y = (y_1, \ldots, y_T) \), where \( y_t \) identifies an action performed at time \( t \) and assumes values from a finite set of labels \( \mathcal{L} \). The recognition is based on a set of features that characterise the observations in \( x \). A feature could be, for example, the location, speed or colour of an object, and so on. There are two main approaches to activity recognition: template-based matching and graphical models.

In template-based matching, a library of behaviours is assumed to be available. Given an observation sequence, this approach matches the observation features against the library to find one or more suitable labels for each observation. Template-based matching does not require machine learning techniques because relationships between observations and labels are encoded into the library. Considerable effort, however, is required to elicit and encode the appropriate knowledge, and matching algorithms may be affected by noise in the observations. Template-based matching approaches are presented by Sukthankar and Sycara [17], and by Bobick and Davis [2].

Graphical models, instead, represent probability distributions using weighted graphs where nodes represent random variables and edges represent probabilistic relationships among the variables. The edges link some input variables – the observations – to some output variables of which we wish to predict the value – the activities – and the edge weights can be computed via supervised learning techniques. Vail et al. [18] explored Conditional Random Fields (CRFs) to recognise behaviours in the RoboCup domain, where two teams of five robots compete in a soccer match. A single CRF is used to identify the role played by one robot (e.g., attacker, goal defender, marker); the approach is then extended to recognise multiple-robot roles by creating a CRF for each robot. Liao et al. [9] employ CRFs organised in a hierarchical structure to generate a model of a person’s high-level activities (e.g., being at work, being at home, travelling) and significant places visited (e.g., workplace, a friend’s house), based on GPS traces of that person’s movements throughout the day.

2.2 Plan Recognition

Plan recognition consists in inferring a subject’s goals based on a set of observed actions (i.e., activities or behaviours) by constructing a plan, or multiple
plans, in which those actions result in goal achievement [5]. Plans may be represented with a pair of action hierarchies [7]: a decomposition hierarchy describing how an abstract action decomposes in simpler ones; and a specialisation hierarchy describing those action types that are a subtype of another action type. The hierarchies are easily mapped onto directed graphs where nodes represents plans/actions and edges constitute hierarchical relationships between nodes. Hierarchical Task Networks, Context-Free Grammars and AND/OR trees [13] are all suitable representations for such hierarchies.

Plan recognisers that can perform incremental monitoring are based on a probabilistic model of plan execution which computes the likelihood of one or more plan branches, according to a stream of incoming observations. Geib and Goldman [5] propose such a recogniser, called PHATT. Their approach addresses a number of simplifying assumption present in earlier work that are too restrictive for effective application in monitoring. Their system can cope with subjects pursuing multiple plans at the same time, partial ordering and temporal relationships in the plan hierarchy, and failure to observe some actions. Kaminka et al. [6] present a similar system, capable of monitoring large teams of pre-deployed software agents performing a joint plan by overhearing the messages they exchange. The system supports execution ordering constraints and actions belonging to multiple plans, and it exploits knowledge about social relationships and communication procedures between teams to achieve efficient monitoring. Multiple alternatives in the decomposition hierarchy are, however, not supported.

3 Software Agent Design

Software agents that address the challenges introduced in Section 1 can be built on the following considerations:

– Environmental sensors are becoming simpler to deploy and can collect large amounts of environmental data during plan execution. There is, however, a risk that processing such information for decision-making may result in cognitive overload for humans [11].

– Humans work in teams to leverage each other’s skills and achieve their goals. A subject’s behaviour can then be inferred by observations about that subject and their teammates. In an uncertain environment, teams may be assembled during execution, and their composition may change according to the environmental conditions [12, 17]. Therefore, software agents should identify a team’s composition and activities during execution (team-oriented).

– The course of action may be adjusted as well, hence it should be identified on the basis of the observations about a team’s activities, by incrementally incorporating new observations about a team over time.

– Teams usually maintain awareness about each other’s progress within the plan. This may become difficult when humans are geographically separated and engrossed with the execution [3]. Software agents should collect this distributed knowledge and facilitate its sharing among the teams.

These points are catered for by the three-level architecture in Figure 1:
1. On the lower level, we assume that a software agent is associated with each team, working as that team’s assistant. Given the teammates’ locations over time, the agent performs incremental activity recognition (see Section 4) and produces a stream of recognised activities $A_i$. Locations can be gathered, for example, by GPS sensors attached to the teammates.

2. On the middle level, the recognised stream of activities $A_i$ is mapped, if possible, onto atomic actions in the plan (hexagons), so to identify one or more active plan branches (bold lines). New branches may be incrementally added or removed as new activities are recognised (see Section 5). At this level the agent needs to have some knowledge about the sub-plan structure its team is pursuing ($STask_i$), and about the dependencies between that sub-plan and those assigned to other teams (links between $STask_i$). Dependencies could be of any nature (e.g., ordering and time constraints, team size) and are usually specified as part of the joint plan.

3. On the upper level, knowledge coming from the plan recognition process allows the agents to monitor crucial circumstances that guarantee goal achievement or determine plan failures. Accessing other teams’ local, up-to-date evidence will allow an agent to predict the impact of such evidence on its own team. This could be used to provide early warnings about plan deviations, constraints likely not to be met, and to draw the attention of the team to pressing issues when it is appropriate [4].

4 Activity Recognition via Conditional Random Fields

Activity recognition in time-stressed scenarios often requires to obtain a less accurate prediction early rather than a very accurate one too late; in this way
corrective actions may be undertaken on time when deviations from the plan occur and synchronisation losses among teams are predicted. Prediction should take a fraction of the time required for the monitored subject to execute the next activity, and the recogniser should be able to produce a reasonable output even when the entire observation sequence has not yet been seen. For this reason we are proposing an incremental activity recogniser, based on Conditional Random Fields (CRFs) [8], that can generate a prediction given a stream of incoming observations, even if the sequence length is not known in advance.

4.1 Linear Conditional Random Fields

A Linear CRF (LCRF) [8] is a probabilistic graphical models to label data sequences. It is structured as a linear chain and can be used to compute the conditional probability distribution $p(y|x)$ of a label sequence $y = (y_1, \ldots, y_T)$ given an observation sequence $x = (x_1, \ldots, x_T)$, with $y_i \in \mathcal{L}$. A LCRF defines $p(y|x)$ as a product of non-negative, real-valued functions $\psi_j(x, y)$ called factors. Factors are captured by an undirected graph where edges represent influence links between the observations $x$ and the labels $y$. In particular, a factor is a set of nodes that are fully connected (also called a clique). Two LCRFs are illustrated in Figure 2, one consisting of nodes $x$ and $y_t$, the other consisting of nodes $x$ and $w_t$. In the illustration, factor $\psi_1$ (hatched area) is represented by the set of fully connected nodes $\{y_1, y_2, x\}$. In order to obtain the linear chains in Figure 2, it is sufficient to repeat factors of type $\psi_1$, with the appropriate changes, for each position in the sequence.

The probability $p(y|x)$ is defined as:

$$p(y|x) = \frac{1}{Z(x)} \prod_{j=1}^{T} \psi_j(x, y) \quad \text{where} \quad Z(x) = \sum_{y' \in \mathcal{Y}} \prod_{j=1}^{T} \psi_j(x, y') \quad (1)$$

The normalisation constant $Z(x)$ is computed over the set $\mathcal{Y}$ of all possible label sequences to guarantee that $p(y|x)$ is a valid probability distribution.

Factors consist in a linear combination of $F$ weighted features $f_i$. Given a sequence of real-valued weights $\lambda = (\lambda_1, \ldots, \lambda_F)$ the factors are defined by:

$$\psi_{\lambda, j}(x, y) = \exp \left( \sum_{i=1}^{F} \lambda_i f_i(x, y_{j-1}, y_j, j) \right) \quad (2)$$

Each weight $\lambda_i$ determines how much the corresponding feature $f_i$ contributes to the probability of $y$ given $x$. In LCRFs, features $f_i$ are arbitrary, real-valued functions defined by four parameters: the observation sequence $x$, the previous and current label values $y_{i-1}, y_i$, and the factor index $j$, which identifies the $j$-th observation in $x$. This makes LCRFs similar to Hidden Markov Models.

\footnote{Note that the label sequence $y$ begins in fact with node $y_0$ (not represented) that accounts for the prior probability of each label value $y_0 \in \mathcal{L}$. $Y_0$ is used by the features constituting factor $\psi_{\lambda, 1}$.}
(HMMs)\(^4\) [13], and it also means that efficient inference and classification algorithms developed for HMMs can be applied to LCRFs with minor adjustments. More importantly incremental algorithms, such as incremental Viterbi [16], can be adapted to work for LCRFs.

Let us now assume that the features \(f\) are known\(^5\). We can employ supervised training together with the Expectation-Maximisation (EM) method [13] to compute the weights \(\lambda\) that completely define \(p(\mathbf{y}|\mathbf{x})\). Given a training set \(\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_M, \mathbf{y}_M)\}\) of \(M\) observations/labels pairs, EM is a numerical method that computes \(\lambda^*\) such that \(\sum_{(\mathbf{x}_m, \mathbf{y}_m) \in \mathcal{T}} p_{\lambda^*}(\mathbf{y}_m|\mathbf{x}_m)\) is maximum. Once \(\lambda^*\) has been found, inference on the LCRF becomes possible. Given a previously unseen observation sequence \(\hat{x}\), there are two kinds of predictions we are interested in: the sequence \(\hat{y}\) of individually most likely labels; and the most likely whole sequence of labels \(\hat{y}'\). The latter is an application of the Viterbi algorithm [13] to LCRFs. In both cases, the costlier computation is \(Z_{\lambda^*}(\mathbf{x})\) defined in Eq. 1, because it loops over all the \(|\mathcal{L}|^N\) possible label sequences.

4.2 Incremental Activity Recognition

The problem of activity recognition in our domain is characterised as follows. We assume that a set \(\mathcal{S} = \{s_1, \ldots, s_S\}\) of subjects is performing a number of coordinated team activities to achieve some goals in a joint plan. Each subject \(s_k\) is uniquely identified and his positions in the environment \(\mathbf{x}_k = (p_1, \ldots, p_N)\) are acquired incrementally as time goes by over a time window of length \(N\). A triple \(p_t = (px_t, py_t, pz_t)\) represents a 3D position in the environment at time \(t\).

\(^4\) With the important difference that CRFs impose no independence requirements on the observation sequence, as opposed to HMMs.

\(^5\) See the Section 4.3 for their definition within the scope of this work.
We will therefore consider the set of observation sequences $X = \{x_1, \ldots, x_S\}$ as evidence. Given a subject $s_k$, we can then build two separate LCRFs:

- The first infers one of $s_k$’s current teammates $y = (y_1, \ldots, y_N)$ over the time window and the probability $p(y|X)$ of such labelling. Although the team $s_k$ belongs to might be specified as part of the plan, it is still necessary to monitor changes.
- The second infers $s_k$’s current behaviour $w = (w_1, \ldots, w_N)$ over the time window and the probability $p(w|X)$ of such labelling. Again, monitoring changes is still necessary.

In Figure 2, nodes $x$ can then be replaced with nodes of type $X$ to perform the two inference computations outlined above, and factors $\psi_i$ will need to be adjusted to consider multiple observation sequences $x_k$. It would be possible to instantiate a pair of LCRFs for each subject $s_k$, however we are investigating a choice of features that would allow us to build only a single pair of LCRFs, which can then perform efficient inference on any subject, once trained. Note also that because of the dependency between $s_k$’s current teammates and behaviours it would be desirable to introduce a dependency between the two LCRFs in the form of additional factors $\phi_j$ that link corresponding nodes $y_i$ and $w_i$, as illustrated in Figure 2. Unfortunately, this would introduce loops among label nodes in the graph, preventing the application of efficient training and inference algorithms, which only work when the graph has a tree-like structure \[9\].

Once the labels $y$ and $w$ have been determined, this information will be used to compute a feasible assignment of subjects $s_k$ to teams $TM_t$ and activities $A_t$ over the time window (see hexagons in Figure 2).

Assigning subjects to teams presents a challenge because, even assuming a subject cannot belong to multiple teams, the number of possible assignments grows exponentially with $|S|$. Examining all of them would mean having a label set $\mathcal{L}$ of exponential size. Therefore, we are recognising only one of the possible subject’s teammates, and we will reconstruct the teams by grouping pairs of teammates. Although this may introduce some uncertainty on the team composition, it should be more efficient than identifying whole teams. There are, in-stead, no problems when recognising a behaviour, as we are assuming a subject can only perform one behaviour at a time.

**On-line inference** plays a crucial part in the software agents’ prediction process, hence it requires efficient algorithms. The recursive forward-backward algorithm \[13\] can be borrowed from HMMs and applied in this case to perform inference in polynomial time\[8\]. This algorithm can work both on the entire observation sequence and on a fixed-size window of most recent observations (fixed-lag smoothing). We are employing the latter to provide incremental prediction in our activity recogniser, under the assumption that evidence collected before the window becomes more and more irrelevant in determining the current activities.

---

6 There are, however, efficient approximate algorithms that perform well with arbitrary graph structures.

7 The number of assignments is the $S$-th Bell number \[17\], where $S = |S|$.

8 For a more detailed explanation refer to \[8\].
4.3 Features for Activity Recognition

The recognition accuracy is mainly determined by the features that constitute the LCRFs, hence they have to be chosen carefully. As we have access to the subjects’ positions over time, our investigation is focusing on features \( f_i \) that capture the following aspects over a time window:

- Magnitudes/headings of the individual position, speed and acceleration of the subjects.
- Magnitudes/headings of the pairwise (i.e., between a pair of subjects) speed and acceleration falling within/outside a specific range.
- Pairwise distance falling within/outside a specific range.
- Rate of change in the pairwise distance.
- Average distance between a subject and all the others.
- Relative speed between a subject and all the others.
- Whether a subject is following another (i.e., angle between two subjects’ speed vectors close to zero).

More complex features may be built on the basis of those listed above, by combining them together. Note also that, although building features considering more than two players might yield better accuracy, that would increase the complexity and may lead to a worst case of \( O(2^{|S|}) \) features when considering the power set \( \mathcal{P}(S) \). By using only features defined on pairs of subjects, or on the entire subject set, we can limit the number of features to \( O(|S|^2) \). Finally, although we are considering only location-based relationships between subjects, it is possible to introduce relationships of any nature as features (e.g., role of a subject, distance between subjects and landmarks).

5 Planning under Uncertainty

A common feature to most probabilistic plan recognisers (see Section 2) is the way they represent the probability of multiple sub-plans when there is sufficient evidence to entail them, but not enough to distinguish the actual course of action. In such cases, a uniform probability distribution is assumed, i.e., each alternative receives the same proportion of the total probability mass\(^9\). The uniform distribution may be replaced with a more suitable one, and the suggestion is usually to learn it from previous examples. There are two issues with this approach. First, learning such a distribution may be difficult when there are few examples, which happens when an ad-hoc plan is devised by humans to cope with a very specific scenario\(^{10}\). Second, relying on a uniform distribution when lacking a better explanation may not help humans in their decision-making process under uncertainty. Lipshitz and Strauss [10] define uncertainty in the context of

\(^9\) \(1/N\) where \(N\) is the number of alternatives entailed by the observations.

\(^{10}\) Of course, when a plan is executed by software agents, running it multiple times and acquiring enough data to model the distribution is much easier.
acting as “a sense of doubt that blocks or delays action”. Uncertainty is ubiquitous in realistic settings and, in particular, equally attractive or unattractive alternatives given the evidence at hand constitute a major obstacle in effective decision-making.

Let us imagine a plan where the evidence collected so far does not allow a software agent monitoring a team to discern between two alternatives. Let us also assume that a human subject is relying on some advice by that agent in order to take different decisions according to which alternative is actually being executed, because that subject is a member of a different team than the one being monitored. A software agent presenting its assessment in the form of two equally likely options to a human planner would not be very helpful to assist the decision-making process. It would be better for the agent to admit a lack of knowledge on the matter, so that the human could try to communicate directly with the other team to reduce his uncertainty. On the other hand, when the agent is able to provide a discriminating estimate, that could save the subject from engaging in communication with others, and, in turn, spare the others from switching their attention to the subject’s request. Therefore, we propose that the concept of missing knowledge, or ignorance, be embedded into the monitoring agent design to promote helpful behaviour towards humans. This can be achieved by considering epistemic uncertainty, as opposed to standard probability [14].

5.1 The Dempster-Shafer Theory

One mechanism that deals with epistemic uncertainty is the Dempster-Shafer Theory (DST) [15]. To perform the matching between behaviours recognised by incremental activity recognition and possible courses of action, we will follow the method introduced by Bauer in [1]. Given a finite set of hypotheses $\Theta$ whose power set is $P(\Theta)$, the DST defines a Basic Probability Assignment (BPA) as a function $m : P(\Theta) \rightarrow [0,1]$ where:

$$m(\emptyset) = 0 \quad \text{and} \quad \sum_{X \subseteq \Theta} m(X) = 1 \quad (3)$$

The hypothesis sets $X$ for which $m(X) > 0$ holds are called focal elements of the BPA; $m(X)$ represents the amount of belief committed in exactly the hypothesis subset $X$, and to no more specific subsets of $X$. Thus, the total belief committed to $X$ and its subsets is given by:

$$Bel(X) = \sum_{Y \subseteq X} m(Y) \quad \text{where} \quad X \subseteq \Theta \quad (4)$$

The DST also states how multiple, independent sources of evidence (probability masses on sets of events) should be combined to obtain an aggregate event likelihood, based on Dempster’s rule of combination [15]. Given two BPAs $m_1$ and $m_2$, and the hypothesis sets $Y$ and $Z$, their combination is calculated as in

$\Theta$ is also known as frame of discernment in the DST.
Eq. 5, where $c$ quantifies the conflict between two evidential sets. $c$ is used to normalise the resulting BPA $m_{12}$ so that both constraints in Eq. 3 hold.

$$m_{12}(X) = \frac{\sum_{Y \cap Z = X} m_1(Y) \cdot m_2(Z)}{1 - c} \quad \text{and} \quad c = \sum_{Y \cap Z = \emptyset} m_1(Y) \cdot m_2(Z)$$ (5)

In our approach, each hypothesis in $\Theta$ will represent one of the behaviours that can be recognised by the incremental activity recogniser, whereas a set of hypotheses will represent a branch of the plan (e.g. the expected course of action).

For example, according to Figure 2, $\Theta = \{A_1, A_2, \ldots, A_i, \ldots, A_P\}$, $Task_1 = \{A_1, A_2, A_3, A_4\}$ and $STask_1 = \{A_1, A_2\}$. Given a plan branch $X$, $Bel(X)$ will represent the amount of belief that the team is actually performing one of the activities in $X$. For example in Figure 2 $Bel(Task_1) = Bel(\{A_1, A_2, A_3, A_4\})$ will evaluate whether the team is performing one of the expected behaviours in the actual course of action. There are three cases that can arise when a monitoring agent tries to detect a behaviour. With reference to Team$_1$ in Figure 2, the agent may:

1. Recognise an activity $A_i$ which can be mapped to Team$_1$’s sub-plan (e.g., $A_1$, $A_2$). In this case the agent will produce a BPA $m_E$ where $m_E(A_i) = p$ and $m_E(\Theta) = 1 - p$. The BPA $m_E$ represent the new evidence to be combined with the current assessment.

2. Recognise an activity $A_i$ which cannot be mapped to Team$_1$’s sub-plan (e.g., $A_5$, $A_6$, assuming that Team$_1$ and Team$_2$ have swapped their assignments). The agent will produce a BPA $m_E$ where $m_E(A_i) = p$ and $m_E(\Theta) = 1 - p$.

3. Fail to recognise any activity. In this case the agent will produce a BPA $m_E$ where $m_E(\emptyset) = p$, $m(\Theta) = 1 - p$.

Given a BPA $m_C$ describing the current assessment of the plan execution, $m_C$ will then be combined with $m_E$ to obtain an updated assessment $m_{CE}$. There are a few observations to make about the three cases above:

- The parameter $p$, $0 \leq p \leq 1$, is the likelihood of the recognised activity. This can be obtained directly from incremental activity recognition given a prediction $\hat{y}^*$ or $y^*$ over a time window (see Section 4).
- In all three cases, assigning a probability mass of $1 - p$ to the set $\Theta$ accounts for the agent’s ignorance about what behavior is, or is not, being performed. Moreover, associating probability masses either to a singleton set or the set $\Theta$ of all hypotheses, reduces the computational complexity of combination rules from exponential time to polynomial time in the number of focal elements in the BPAs to be combined.

The activities $A_i, \ldots, A_P$ are generated by $Task_2, \ldots, Task_N$, and $P = |\Theta|$.

BPAs where the belief mass is assigned only to singletons or to the entire frame of discernment $\Theta$ are also called simple support functions in the DST.

The issue of how to initialise $m_C$ when no evidence has been collected yet needs to be considered as well. The approach in [1] suggests using the vacuous hypothesis $m_C(\Theta) = 1$ when no evidence is available, but other kinds of assignment are possible as well.
– in Case 3, the resulting function $Bel_{CE}(X)$ calculated over $m_{CE}$ should assume a value below 1 for every $X \subseteq \Theta$, including the case $X = \Theta$. This would be a strong signal that a deviation is occurring during the execution.

To achieve this, however, we need to relax the definition of BPA (see Eq. 3) by allowing $m(\emptyset) \geq 0$, such that $Bel(\emptyset)$ will not account for some of the total belief mass, as it has been assigned to the empty set.

None of the several combination rules for the DST reported in [14] seems to satisfy all the above requirements. For example, Yager's rule requires Eq. 3 to hold and assigns the amount of conflict $c$ to the set $\Theta$ (hence transforming conflict into ignorance), so it cannot deal with situations covered by Case 3. On the other hand Smets’ rule always assigns conflict to the empty set, as a consequence if Team$_1$ starts performing $A_1$ but then switches to $A_2$ for any reason, the belief mass intially assigned to $A_1$ will be transferred to the empty set, resulting in $Bel_{CE}(\emptyset)$ evaluating to less than 1 even if in fact Team$_1$ is still operating within the expected course of action. Moreover, the computation of $c$ in Eq. 5 has been disputed in several publications (they are reviewed in [14]) because it completely removes the conflict and can yield counter-intuitive results. For these reasons, we will employ the Combination by Compromise (CBC) rule$^{15}$ [19], which is based on Eq. 5 with $c$ set to 0. Given two BPAs $m_1$ and $m_2$, CBC assigns a part of $m_1(Y) \cdot m_2(Z)$ to $C = Y \cap Z$, and distributes the remaining part to $Y_Z = Y \cap \neg Z$ and $Z_Y = \neg Y \cap Z$, proportionally to the values of $m_1(Y)$ and $m_2(Z)$. Note that the CBC will generate probability mass assignments that are more complex than simple support functions, so it will be necessary to coarsen them back into simple support functions to maintain the computational advantage gained in the computation of $m_{CE}$.

![Fig. 3. Mechanism for incremental plan recognition](image)

$^{15}$ A detailed comparison between the CBC rule and the other combination rules is being carried out.
5.2 Incremental plan recognition

In order to perform incremental plan recognition, we will integrate the above DST-based probability model into Phatt [5] (see Section 2), hence replacing its original uniform probability assumption. Phatt adopts a model of plan execution to perform plan recognition on plans defined by a modified Context Free Grammar, called Plan Tree Grammar. A Plan Tree Grammar defines a set of production rules that generates an AND/OR tree with the following additions: for every terminal in the grammar there exists a unique non-terminal that produces solely that terminal symbol, and there are explicit ordering constraints embedded in the production rules. The former addition is exploited to generate plan hypothesis following an efficient bottom-up approach, while the latter enables handling partially ordered plans. Terminal symbols in the grammar represent atomic actions; atomic actions may be of arbitrary nature, and those recognised via activity recognition (see Section 4) will fit the role well. Generating the complete set of plan hypotheses given a sequence of recognised activities can be a computationally expensive task; indeed, given a plan library and a single explanation for the sequence of activities observed so far, just adding one additional observation may generate a large number of new explanations for the updated sequence. This is because the new observation could both contribute to an existing plan or begin one or more new plans. Phatt uses a bottom-up approach that generates a new explanation only after having observed an activity that supports it, possibly back-propagating the new explanation to the beginning of the evidence sequence.

Although Phatt was devised to recognise plans performed by a single individual, it should be easily adapted to handle team plans where teams are considered as units, as follows. With reference to Figure 3, once a team (Team$_1$) has chosen a goal (Goal$_1$) to achieve, there will be a set of activities in the plan that can be performed (i.e. enabled) without prerequisites, which are illustrated as PendingSet$_1$. PendingSet$_1$ contains the activities that the evidence from incremental activity recognition will be matched against. For example, if Team$_1$ is recognised performing activity A$_1$ (leftmost hexagon in Team$_1$’s plan) with a certain probability $p$, $p$ will be integrated as new evidence using the DST model to generate an updated assessment of the plan status. The probability mass on Team$_1$ performing A$_1$ increases over a time window if the activity is consistently recognised with high probability, meaning that A$_1$ has been effectively chosen for execution by Team$_1$ (hexagon colour changing from red to green). When the probability mass assigned to A$_1$ exceeds a specific threshold, A$_1$ can be considered achieved (done). This in turn modifies the current pending set: new activities may become enabled and be added to the pending set (e.g. A$_2$), and available ones may be removed. For example, Team$_1$ completing A$_1$ allows A$_2$ to be added to PendingSet$_2$.

Computing the probability mass of a higher-level plan step can be achieved by applying Eq. 4. Indeed, those aggregate plan steps are represented as sets of activities. This is illustrated in Figure 3 with the nodes’ colour changing from red to green throughout the plan tree. The other extension Phatt requires concerns
the Plan Tree Grammar which in [5] supports only ordering constraints between sibling nodes (i.e., those appearing on the right-hand side of the same production rule). Because a joint plan may be broken down in separate components which are executed in parallel, it will be necessary to introduce links between nodes appearing in different production rules.

6 Preliminary Result for the Activity Recogniser

For our evaluation we are using OpenArena\(^\text{16}\), an open-source game based on the Quake III Arena codebase which offers an environment where players play in a simulated battle-space. Our previous investigation on off-the-shelf computer games [11] confirmed that they offer a viable solution as virtual environments for collecting both synthetic and human-generated data. In particular, we have collected position data from “capture the flag” game sessions, where two opposing teams have to collect a flag located somewhere in the environment and bring it to the opponent’s base, while protecting their own base. We have instrumented OpenArena to run simulated game sessions using embedded AI players (bots). Bots can perform several behaviours (e.g., fight, retreat, stand) and can be organised in teams performing higher level activities (e.g., attack, patrol, defend).

We have collected the bots’ positions, behaviours, and team activities over time with a variable number of bots in each team (6, 8 and 10) and in different scenarios (game maps). At the moment, we are evaluating the activity recogniser on its accuracy in predicting a team’s composition, given the ground truth collected from simulations. We have run experiments in the following conditions:

- Two opposing teams (Red and Blue) of 10 bots each played for 30 minutes in four games; position data were collected for all bots every 2 seconds.
- Each team repeatedly split into disjoint subteams (sized from 1 to 8 bots) performing different tasks according the current situation; e.g., when the flag was free, a 2-bot subteam attacked the enemy base, a 2-bot subteam tried to capture the flag, and a 6-bot subteam protected the home base.
- Killed bots were respawned into the game again after a 15-second delay.

In these initial experiments, one LCRF was created for each bot to identify one of the teammates in the bot’s subteam\(^\text{17}\). The CRFs were trained with data from three of the game sessions and the features defined in Section 4.3, while the fourth session was used for prediction. The prediction accuracy is listed in table Table 1. There are a few reasons why the accuracy is not very high:

- The game is very dynamic, hence subteams within one team form and disband quickly, according to the current flag position (at the center, at the home/enemy base).

\(^\text{16}\) See \url{http://openarena.ws} for more information about the game and the game modes.

\(^\text{17}\) Subteams will need to be reconstructed at a later stage.
<table>
<thead>
<tr>
<th>Classification method</th>
<th>Red 1</th>
<th>Red 2</th>
<th>Red 3</th>
<th>Red 4</th>
<th>Red 5</th>
<th>Red 6</th>
<th>Red 7</th>
<th>Red 8</th>
<th>Red 9</th>
<th>Red 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum-a-Posteriori ($\hat{y}$)</td>
<td>53.6%</td>
<td>43.2%</td>
<td>46.3%</td>
<td>53.9%</td>
<td>40.9%</td>
<td>53.6%</td>
<td>43.2%</td>
<td>46.3%</td>
<td>53.9%</td>
<td>40.9%</td>
</tr>
<tr>
<td>Viterbi ($\hat{y}'$)</td>
<td>49.5%</td>
<td>39.5%</td>
<td>42.7%</td>
<td>49.3%</td>
<td>40.1%</td>
<td>49.5%</td>
<td>39.5%</td>
<td>42.7%</td>
<td>49.3%</td>
<td>40.1%</td>
</tr>
<tr>
<td>Blue 1</td>
<td>Blue 2</td>
<td>Blue 3</td>
<td>Blue 4</td>
<td>Blue 5</td>
<td>Blue 6</td>
<td>Blue 7</td>
<td>Blue 8</td>
<td>Blue 9</td>
<td>Blue 10</td>
<td></td>
</tr>
<tr>
<td>Maximum-a-Posteriori ($\hat{y}$)</td>
<td>65.3%</td>
<td>60.8%</td>
<td>35.4%</td>
<td>68.4%</td>
<td>55.6%</td>
<td>65.3%</td>
<td>60.8%</td>
<td>35.4%</td>
<td>68.4%</td>
<td>55.6%</td>
</tr>
<tr>
<td>Viterbi ($\hat{y}'$)</td>
<td>64.0%</td>
<td>61.0%</td>
<td>32.7%</td>
<td>65.1%</td>
<td>55.1%</td>
<td>64.0%</td>
<td>61.0%</td>
<td>32.7%</td>
<td>65.1%</td>
<td>55.1%</td>
</tr>
</tbody>
</table>

Table 1. LCRF accuracy in predicting a bot’s teammate for the Red and Blue teams

– Bots do not operate in fixed-shape formations, and sometimes pursue individual goals, e.g., when they are under attack or looking for ammunitions. In these situations a bot can shortly leave its subteam and rejoin it later.

– Although subteams are disjoint, sometimes they work in close proximity hence distinguishing which bots belong to which subteam becomes difficult.

On the other hand, the accuracy obtained through LCRFs is at least three times higher than randomly guessing a bot’s teammate, which would result in an accuracy of just 11% (1/9 bots). In addition, Maximum-A-Posteriori classification seems to perform generally better then the Viterbi algorithm.

7 Conclusions and Future Work

In this paper, we propose an approach to enable software agents to offer helpful and meaningful support to humans teams performing a joint plan. The mechanism integrates incremental activity and plan recognition in a distributed setting to monitor the plan enactment, with the aim of maintaining awareness and synchronisation among the teams. We are currently working on different ways to improve the activity recognition accuracy. Using more examples for training is a possibility but it would become increasingly difficult when data is collected from human subjects in training rather than from artificial players. Therefore, we are exploring additional features that capture relationships between subteam members. Relationships that rely on the activity the bots are performing (e.g. attack, retreat) will also be encoded, as they will be recognised through LCRFs as well. Finally, we will evaluate the activity recogniser on position data collected during a simulation with human subjects. In the simulation scenario, two opposing teams consisting of military personnel performed a sequence of manoeuvres in order to secure a location, so it will be possible to train the recogniser on a partial sequence of the collected positions and then perform a prediction on the remaining data.
References