In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (IJCAI 2019), Macao, China, August 2019

Ad Hoc Teamwork with Behavior Switching Agents

Manish Ravula, **Shani Alkoby** and **Peter Stone**
The University of Texas at Austin, Texas, USA
manishreddy@utexas.edu, shani@cs.utexas.edu, pstone@cs.utexas.edu

Abstract

As autonomous AI agents proliferate in the real world, they will increasingly need to cooperate with each other to achieve complex goals without always being able to coordinate in advance. This kind of cooperation, in which agents have to learn to cooperate on the fly, is called ad hoc teamwork. Many previous works investigating this setting assumed that teammates behave according to one of many predefined types that is fixed throughout the task. This assumption of stationarity in behaviors, is a strong assumption which cannot be guaranteed in many real-world settings. In this work, we relax this assumption and investigate settings in which teammates can change their types during the course of the task. This adds complexity to the planning problem as now an agent needs to recognize that a change has occurred in addition to figuring out what is the new type of the teammate it is interacting with. In this paper, we present a novel Convolutional-Neural-Network-based Change Point Detection (CPD) algorithm for ad hoc teamwork. When evaluating our algorithm on the modified predator prey domain, we find that it outperforms existing Bayesian CPD algorithms.

1 Introduction

Autonomous agents, both in software and robotics, are becoming increasingly capable of solving complex tasks. However, if these agents are to perform day to day activities as a part of society, they will need to be able to cooperate with other agents. Often in studies of cooperative agents, the coordination strategy is either learned or decided *a priori* while assuming full knowledge of the teammates and the task at hand. However, as agents become more robust and diverse, it will become progressively more difficult to ensure that all the agents share the same communication and coordination protocols. Thus, these agents will need to be able to cooperate on the fly. For example, in case of a disaster, it might not be possible (due to lack of time or resources), to reprogram the existing heterogeneous robots deployed in the area and provide them with the knowledge of each other's capabilities to assist the search and rescue operations. The Drop-in Player competition at RoboCup [MacAlpine *et al.*, 2014] is another setting that necessitates ad hoc cooperation. In this variant of robot soccer, new robot teams are formed by mixing robot players from different teams. They have to cooperate and play together to win. Such challenging tasks can only be accomplished if these robots are able to work together without the need to be explicitly provided with strategies in advance.

This problem, in which a team of agents is formed ad hoc, for a particular purpose, and the team strategies cannot be developed *a priori*, is called the "ad hoc teamwork" problem [Stone *et al.*, 2010]. Several works approach this setting by assuming that every agent behaves according to one out of a set of predefined behaviors [MacAlpine *et al.*, 2014; Albrecht and Ramamoorthy, 2015; Albrecht *et al.*, 2016; Auer *et al.*, 2002; Barrett *et al.*, 2012; 2011]. These, behaviors (also called types), are often assumed to be defined in the form of probability distributions mapping states to actions. Cooperation then is effectively split into reasoning and planning, where the ad hoc agent first reasons about the teammates' capabilities and behaviors and then plans actions to optimally accomplish the task at hand. If the types are sufficiently descriptive and the reasoning algorithms are capable enough, the agent's beliefs regarding the other agents' type will rapidly converge, leading to successful performance of the task. Common to all past work is the assumption that *teammates maintain the same type throughout the entire task*. Real world teammates, however, may not be static in terms of agent behaviors. If the ad hoc agent doesn't swiftly recognize such changes and adapt accordingly, teamwork will surely degrade. Search and rescue tasks are an important class of such examples.

In this paper, we relax the assumption of the agents' types being fixed through the task and consider the more realistic problem of agents dynamically switching between types through the course of the task. We formulate this problem as a Change Point Detection (CPD) problem in which ad hoc agents are required to identify throughout the task, whether a change in the type of the other agents has occurred and if so, what the new type is. We investigate the use of existing CPD algorithms and propose a new CNN (Convolutional Neural Network)-based CPD algorithm. Finally, using a modified version of the predator prey domain we find that our algorithm outperforms other CPD algorithms in detecting and adapting to changes in agent types.

2 Related Work

In this section we discuss the current state of the art in the area of ad hoc teamwork, specifically in the type-based approach. Next, we discuss the change point detection problem and its connection to our research.

2.1 Type-Based Ad Hoc TeamWork

Approaches to ad hoc teamwork broadly fall into two categories based on how the ad hoc agent models the rest of the team [Albrecht and Stone, 2018]. The well-studied first category involves modeling agents individually with distributions over action probabilities at each timestep [Brown, 1951]. The second approach involves modeling the group as a whole and its joint action/planning dynamics [Tambe, 1996]

Type-based reasoning falls into the first category. In the last decade, multiple works have studied this problem in various contexts and experimental domains. Several works have concentrated on investigating likelihood methods for efficient inference of type given the predefined behavior/type set, using Monte Carlo Tree Search (MCTS) for planning actions accordingly [Barrett *et al.*, 2011; Stone *et al.*, 2010; Albrecht *et al.*, 2016; Albrecht and Stone, 2017]. Going one step further, a number of works have investigated algorithms that can build this set of behaviors while performing the task instead of assuming that it is given beforehand [Nikolaidis and Shah, 2013; Nikolaidis *et al.*, 2014].

All of the above works assume however, that the teammates' types remains fixed and do not account for type switches. The only paper that does consider non-stationary teammates [Hernandez-Leal *et al.*, 2017] focuses on detecting drift between a learned set of types and the agent's current behavior to help decide when the current type set isn't expressive enough of the behavior. While their work aims to build a repertoire of behavior types, ours focuses more on the immediate problem of co-operation in the presence of type-changing teammates.

2.2 Change Point Detection

Change points are abrupt variations in time series data. Such abrupt changes may represent transitions that occur between states. Change Point Detection (CPD) has been investigated in many application areas such as climate change detection [Reeves *et al.*, 2007] and robotics [Aminikhanghahi and Cook, 2017]. Various algorithms have been proposed to detect and track these changes, both offline and online. Algorithms like the CUMSUM [Page, 1954], KLIEP [Sugiyama *et al.*, 2008] and SPLL [Kuncheva, 2013], that work with repeated hypothesis tests fall under the category of Likelihood based statistical methods and are strongly tailored to numerical time series sampled from parametric probability distributions. Bayesian Methods ([Xuan, 2007; Fearnhead and Liu, 2007; Niekum *et al.*, 2015]) involve priors on change point locations and can work on arbitrary, non-parametric model specifications. Both the online and offline versions of Bayesian CPD algorithms often grow in $O(T^2)$ in complexity as the total number of timesteps increases. Finally, recent work on LSTM-RNN based change point detection [Meng *et al.*, 2017] has been promising due to the representational power afforded by the neural networks as well as the long range time dependencies captured by the LSTM architectures. These methods first learn a predictive model of the time-series data distributions and then measure the drift from the predicted value to the true value to identify changes.

All of the aforementioned algorithms assume that the time-series data at any given timestep within a segment is generated from a stationary random process. This assumption proves detrimental when we want to infer switches in types solely based on observing an agent's actions. Since an agent's probability distribution over actions is generally conditionally dependent on its state at every timestep, this assumption of stationarity is invalid and as will be shown, affects the performance of current CPD algorithms. The algorithm presented in this paper does not make this assumption and is specifically tailored to work with non-stationary agent models.

3 Preliminaries

This paper's terminology and notation follows that of Albrecht and Stone [2017].

3.1 Model

We consider a multi-agent model where agents interact with each other in order to achieve a common goal. The process starts at time $t = 0$. At time t, each agent i receives a signal s_i^t and independently chooses an action a_i^t from some countable set of actions A_i. We do not put any limitations on s_i^t's structure and dynamics. This process continues indefinitely or until some termination criterion is satisfied (i.e., a goal is achieved).

We will use $P(a_i^t|H_i^t, \theta_i)$ to denote the probability with which the action a_i^t is chosen where $H_i^t = (s_i^0, ..., s_i^t)$ is agent i's history of observations and θ_i is i's *type*. Since this work mainly focuses on detecting type changing points and since the work of Albrecht et al. provided a method for reasoning about the values of any bounded continuous parameters within types, we will assume that the types are characterized without the need of parameters.

To simplify the exposition, we assume that we control a single agent, i, which reasons about the behavior of another agent, j. We also assume that i knows j's action space A_j and that it can observe j's past actions, i.e. $a_j^{t-1} \in H_i^t$ for $t > 0$. The true type of j, denoted θ_j^* is unknown to i. However, i has access to a finite set of *hypothetical types* $\theta_j \in \Theta_j$, with $\theta_j^* \in \Theta_j$. We furthermore assume that all agents share the same global state and by extension, i has all information relevant to j's decision making, so that H_j^t is a function of H_i^t. Finally, we assume that agent j will change its type during the process at a number of chosen time points set exogenously.

Our goal is twofold. First, we aim to devise a method which allows agent i to be able both to identify the specific time point in which the change in agent j's type has occurred and to identify its new type, based only on agent j's observed actions. Second, we aim to adapt the planning method to cope with these changes.

3.2 Reasoning in the Absence of Change points

Without considering the option that agents are allowed (or able) to change their type during the task to be accomplished,

Algorithm 1 MAP Type Estimation

```
Given type space $\Theta$, initial belief $P(\theta_i|H_i^t)$
Output: Type estimates at each timestep, $\hat{\theta}_t$
for each timestep t > 0 do
    Observe action $a_m^t$ of $m^{th}$ agent
    for each type $\theta_i$ in type space $\Theta_j$ do
        $P(\theta_i|H_i^t) \leftarrow P(a_m^t|\theta_i) * P(\theta_i|H_i^t - 1)$
    end for
    set $\hat{\theta}_t \leftarrow \text{argmax}_{\theta_i}(P(\theta_i|H_i^t))$
end for
```

our agent will use the *MAP type estimation* method as defined in the work of [Albrecht and Stone, 2017] in order to identify the other agents' type and plan accordingly. According to the *MAP type estimation* method, our agent maintains individual probability for each possible type in Θ_j and updates them after each observation. This process is formally described in Algorithm 1.

3.3 Planning

Given an assumption of teammate types, the agent can then *plan* a sequence of actions that, in conjunction with the predicted actions of teammates, will lead to the best team utility. Previous work for planning [Fridman, 2018] has used Monte Carlo Tree Search (MCTS) as it has relatively few restrictions on the domain and often works quite well for short-term planning. To reduce computational complexity and simplify exposition and since planning itself is not the focus of our work, we use a simple, domain-specific planning algorithm that is described in Section 5.4.

4 Proposed Methodology

Algorithm 1 does not explicitly consider the possibility of teammates changing types. Since the belief is propagated from the beginning, it often takes many timesteps of lag for the posterior $P(\theta_i|H_i^t)$ to reflect the changed type, owing to drift in belief. Hence, identifying change points in this way can be detrimental to fast inference of the new type after a change occurs. We aim to solve this belief-drift problem by incorporating a change point detection phase where the ad hoc agent inspects the history to identify possible switches in its teammates' types. If any such switches are found, then the reasoning algorithm resets its recent history to begin just after the change point and uses only the reset history for inference. Specifically, we reset the evidence $P(\theta_i|H_i^t)$ immediately after a change point is observed. This modification in reasoning strategy helps rapid convergence of the type-inference procedure to the new type after a change point and consequently aids in minimizing planning lag.

Since the choice of the change point algorithm is not straightforward, we compare existing algorithms with a newly proposed CNN-based change point detection method. This new algorithm is described in detail in the following section, while the existing algorithms are described in Section 5.3.

4.1 Convolutional CPD Network

Convolutional Neural Networks [LeCun *et al.*, 2015] have shown remarkable performance on many image-related tasks. Effective composition of convolutions coupled with non-linear transformations give CNNs the power to learn and distinguish spatial patterns accurately. We aim to leverage this power by representing our change point detection problem as a 2D image classification like problem. This process is illustrated in Figure 1. The figure illustrates how different the likelihood matrices $P(a^T|\theta_i)$, called $\boldsymbol{L}_{|\Theta|\times n_t}$ are, for a given timestep T and observed action a^T. In the ideal case, when the types are very different from each other, an observed action should have a high likelihood only from the actual type that generated it and near zero likelihood from all others [1]. Thus, when a change point occurs, the likelihood mass must also shift towards the new type. Such a shift in likelihood will show up as a break in the highest likelihood line (colored yellow), as illustrated in the figure. Thus, recognizing this break in the image-like representation can help us detect change points.

Since detecting such a pattern requires both horizontal (time) and vertical (type) related analysis, Convolutional Neural Networks are a natural fit. Thus, we can pose the change point detection problem as an image-classification problem, where each likelihood-matrix when interpreted as an image can be classified into one of $n_{classes} = {}^{|\Theta|}P_2 + 1 = |\Theta| \times (|\Theta| - 1) + 1$ labels based on the presence/absence of a change point and the pre-change-point type, post-change-point type.

The architecture we used to solve this classification problem is summarized in Table 1. The network takes as input the matrix $\boldsymbol{L}_{|\Theta|\times n_t}$. The first layer has multiple 40 2d-convolutional filters followed by a max-pooling layer. The activations are then passed through the ReLU non linearity into a series of fully-connected (FC) layers. Finally, the output is soft-maxed to get the probability of each of $n_{classes}$ happening in the last T-timesteps. Here, the width n_t of $\boldsymbol{L}_{|\Theta|\times n_t}$ is considered a hyper-parameter and is chosen to facilitate the best accuracy for a particular task and type-set at hand. Larger widths translate to access to increased length of history and hence better accuracy. This trade-off is discussed further in the experiments section.

At each timestep, we pass the last n_t timesteps' likelihood information to the matrix and retrieve the output probabilities for all possible switches at $T - \frac{n_t}{2}$. This process is similar to a sliding-window approach, where we are sliding over likelihood matrices. If the network outputs the highest probability for a change-point at timestep T, then a change-point is marked at timestep $T - \frac{n_t}{2}$.

The network is trained using the likelihood matrices derived from simulation. Inside each simulation run, we infuse changepoints randomly in a teammate and collect likelihood matrices pertaining to its actions centered around the change-point. The changepoints are sufficiently spaced apart so that the likelihood matrices collected only contain

[1] In the extreme-ideal case, the types would be sufficiently different to have non-overlapping likelihoods for each action: in every state, each type generates a different action.

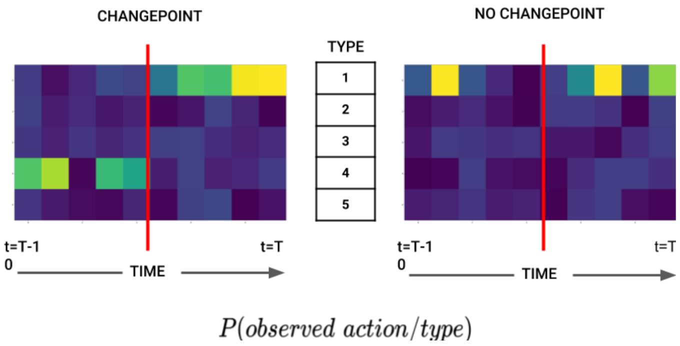

Figure 1: Image-like representation of $P(a^T|\theta_i)\ \forall\{\theta_i \in \Theta\ ,\ |\Theta| = 5\}$ where a^T is the action observed at timestep T. The image-like patterns are starkly different for change points vs no change point.

Algorithm 2 Convolutional Changepoint Detection (ConvCPD) for each agent

```
Output: $p_c^{m,n}$ = Probability of a type change from m to n
    occurring within the last $h_l$ timesteps).
  $out1 \leftarrow ConvCP_1.forward(L^t)$
  $p_c^{m,n} \leftarrow Softmax(out1)$
      return $p_c^{m,n}$
```

information about a single changepoint. This set of matrices is augmented with another set of matrices which is collected without changepoints so as to have a balanced dataset. The details are further described in Section 5.

Layer Name	Layer Dimensions
Input	$(\|\Theta\|, n_t)$
Conv1	(40 x 3 x 3)
Maxpool	(2x2)
ReLU1	-
FC1	$(40 * ((n_s \times 2) - 2) \times (n_t - 2) \times 100)$
ReLU2	-
FC2	(100 X 100)
ReLU3	-
FC3	(100 X 20)
ReLU4	-
FC4	$(20,\ n_{classes}\ )$

Table 1: Architecture of the CPD Network used in our experiments. The choice of layer sizes are specific to our experiments and can be changed/resized accordingly for other applications

5 Experimental Evaluation

We provide a detailed experimental evaluation of the proposed method in a modified predator-prey domain [Benda *et al.*, 1986].

5.1 Domain Description

Our domain is a modified version of the predator-prey domain. The domain models the environment as a square grid in which two agents (predators) are acting and $n \in \mathbb{N}$ preys are present. A prey is stationary, i.e., cannot change position on the grid during the task. A predator however, can change position during the game by executing one out of the following actions: U for moving up, D for moving down, R for moving right, and L for moving left. Predators can also stay put by executing the action N. At each timestep, both predators decide separately upon an action they are interested in performing. A conflict can occur if both agents chose actions that move them to the same position on the grid. In case of such conflicts, ties are resolved randomly and the losing predator is forced to perform action N (i.e stay put). Otherwise, they simply proceed by performing their chosen actions. We denote the amount of timesteps that the task is allowed to continue by N_{MAX}. Other than moving across the grid, the predator can capture a prey by performing the C (for CAPTURE) action. C can be executed only when the predator neighbors the prey (no matter from which direction). Once an agent performs a C action, it remains locked onto the target and can no longer execute any other action, i.e., remains in its current position in a capturing mode for the rest of the time left. If both agents perform the C action on the same prey, the prey is captured. If the predators were able to capture one of the preys, then the task terminates successfully. However, if a prey is not captured within N_{MAX} timesteps, the task is terminated as a failure. Figure 2 depicts an example grid configuration of our domain where $n = 3$, i.e., there are 3 preys (yellow squares) and 2 predators (blue circles). Finally, we note that in our experiments, only one agent is an ad hoc agent trying to track the other agent's type. The other agent's type is randomly chosen at the beginning of the simulation episode.

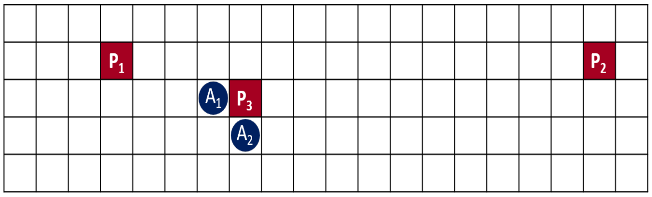

Figure 2: An example state in the modified predator prey domain.

5.2 Agent Types

We consider the pre-planned predator agent's type characterized by the prey it currently is in pursuit of, i.e., its type is θ_i if it pursues prey number i.[2] At each timestep the agent calculates a path to its prey using the A^* algorithm. It then assigns a high-probability (0.9), to the action suggested by the path-planning algorithm and a low-probability (0.1 evenly distributed over the rest) to all other valid actions. This distribution is passed through a *softmax()* function to infuse randomness in actions. The introduction of the *softmax()* function enables adjustment of agent behavior with a single parameter - alpha. Finally, the agent samples an action from this distribution and executes it if there are no conflicts with

[2] The ad hoc agent does not have a type since its goal is to identify the other agent's type and to best cooperate with it, i.e., choose the prey it will pursue based on the prey the pre-planned agent chose.

Algorithm 3 Template for Agent Types

```
Type $\theta_i$
Output: $(p_a, a_i^t) = P(a_i^t|H_i^t, \theta_i), a_i^t$
$Target \leftarrow Objects[\theta_i]$
initialize the probability vector $p_a$ to 0s.
if Agent is a neighbor of Target then
    Assign probability 1 to C - CAPTURE action;
    break
else
    Use $A*$ to estimate path to Target
    Assign probability 0.9 to first move from the path
    Distribute the remaining probability of 0.1 equally and assign the parts to valid moves apart from the move generated by A*.
    Perform softmax($p_i$) $= \frac{e^{\alpha * p_i}}{\sum e^{\alpha * p_i}}$ with temperature $\alpha = 2$ over non-zero probabilities $p_i$ to derive final action probabilities.
end if
$a_i^t \leftarrow sample(p_a)$
return $(p_a, a_i^t)$
```

other agents. The full algorithm for the predator agent's behavior given its type is described in Algorithm 3. This algorithm outputs a probability distribution over actions (p_a) and a single action (a_i^t) sampled from p_a to perform at time-step t.

Both pre-planned and ad hoc agents navigate to their target prey using the A^* algorithm.

If the adhoc agent doesn't correctly infer the type of its teammate, the simulation can result in failed termination because both agents perform the CAPTURE action on different preys.

5.3 Bayesian Change Point Detection Algorithm

The widely used Bayesian model-based change point detection algorithm was first presented by [Fearnhead and Liu, 2007]. Their model assumes time-series observations $y_{1:n} = (y_1, y_2, ..., y_n)$ and a set of candidate models Q. The goal is to infer the number of changepoints *m* and their MAP (Maximum A-Posteriori) times $c_1, c_2, ..., c_m$, where $c_0 = 0$ and $c_{m+1} = n$ (i.e., there exist $m + 1$ segments). The observations $y_{c_i+1:c_{i+1}}$ forming the ith segment are assumed to be produced by the associated model $q_i \in Q$ with parameters θ_i.

The basic assumption in this model is that data after a change point is independent of data prior to that change point. Thus, we can model the change point positions as a Markov chain in which the transition probabilities are defined by the time since the last change point in the following way: $Pr(c_i + 1 = t|c_i = s) = g(t - s)$, where $g(x)$ is a probability distribution over time.

The model evidence for a model q and a given segment starting from a time point s and ending at a time point t is defined by: $L(s, t, q) = Pr(y_{s+1:t}|q) = \int Pr(y_{s+1:t}|q, \Theta)Pr(\theta)d\theta$.

We denote the event that a change point will occur at time j by ψ_j and the event that given a change point at time j, the MAP choice of change points has occurred prior to time j by ω_j. We can now use the notations: $Pr_t(j, q) = Pr(FC_t = j, q, \omega_j, y_{1:t})$ and $P_t^{MAP} = Pr(\psi_j, \omega_j, y_{1:t})$, where FC_t is the distribution over the position of the first change point prior to time t which can be efficiently estimated using the standard Bayesian filtering recursions and an on-line Viterbi algorithm [Forney, 1973].

From the problem setup, we proceed to derive:

$$Pr_t(j, q) = (1 - G(t - j - 1))L(j, t, q)Pr(q)P_j^{MAP} \quad (1)$$

$$P_t^{MAP} = \max_{j,q}\Big[\frac{g(t - j)}{1 - G(t - j - 1)} Pr_t(j, q)\Big] \quad (2)$$

Here, $G(x)$ is the cumulative distribution function of $g(x)$. Finally, the Viterbi path can be recovered by finding the j and q values that maximize (2) at time t. We then can repeat the process again in order to find the values which maximize (2) at time j or any time point beforehand until reaching zero. The algorithm is fully on-line, but requires $\mathcal{O}(n^2)$ computation at each timestep.

5.4 Results

In our experiments we simulate agents' behaviors at every timestep. The ad hoc agent runs Algorithm 2. The ConvCPD algorithm is trained with 10,000 samples (batch size = 64, learning rate = 0.01, decay = 0.1, optimizer = SGD) involving equal proportions of all classes. After passing the matrix through the CNN, we retrieve the probabilities of all possible sequences of types before and after the center-point in the matrix, i.e at time $T - \frac{n_t}{2}$. Using these probabilities, we compute the location and nature of the change point as the class with the maximum probability output by the CNN. The ad hoc agent plans simply by moving to the prey that it infers as the target of the other agents' type. Since the task at hand is simple, this planning algorithm works well. Results are averaged over 150 trials.

Table 2 displays the influence of n_t (the width of L) on both the change point detection accuracy and the mean squared error (MSE) of change point time estimation. From looking at the table, one can see that as n_t increases, the accuracy of the detection increases and the MSE decreases. This result makes sense, since the more timesteps the agent has as an input to the CNN, the more information it has on which to base its prediction.

	$n_t = 20$	$n_t = 16$	$n_t = 14$	$n_t = 10$
Accuracy	88%	72%	53%	22%
MSE	1.2	2.4	3.5	4.2

Table 2: Change point detection accuracy and the mean squared error (MSE) of change point time estimation for different values of n_t.

For evaluating the overall improvement in the teamwork performance where agents' types are dynamic, we tested the average number of timesteps it took the agents to successfully finish the task where there are 6 preys on the grid, i.e., $|\Theta| = 6$. Figure 3 depicts the number of time-steps the

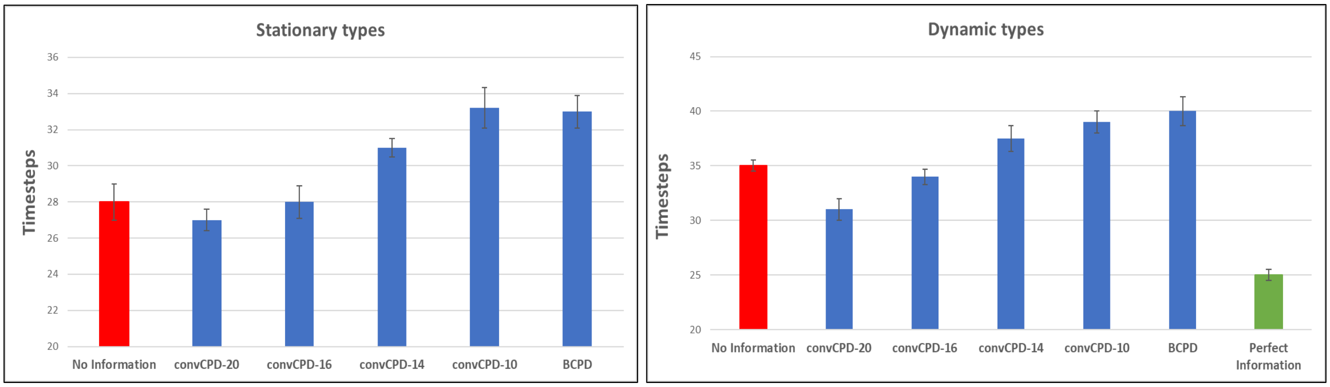

Figure 3: The number of timesteps required for completing the task using the different change point detection algorithms for both stationary and dynamic types.

team required to successfully complete the task using different change point detection algorithms both for the case where agents are stationary (left) and dynamic (right). We note that for the dynamic case, if the ad hoc agent knew the pre-planned agent's type at every timestep, i.e., has perfect information, the number of timesteps needed for successfully completing the task, as can be seen from the figure, is the lowest possible. Thus, we consider this case to be our (unachievable) lower bound.

As mentioned above, when using the Conv-CPD algorithm, the network performs with highest accuracy when n_t is 20. This is also observed from the right graph appearing in Figure 3. As the value of n_t decreases the number of timesteps it takes the team to complete the task increases. Moreover, in the case where $n_t = 14$ or 10 it takes the team longer to complete the task than it would have taken them to complete it in the case of no information, i.e., without any awareness to the fact that agents are changing their types throughout the task.[3] If using the Bayesian CPD (BCPD) however, the number of timesteps it takes the team to complete the task is the highest observed.

Finally, in many real life situations, agents may not know in advance whether their teammates will change their type throughout the task or not. Therefore we want to make sure that applying a change point detection algorithm even if the types are fixed will not lead to falsely detecting of change points and decrease in performance. The left graph in Figure 3 illustrates the number of timesteps it takes the team to complete the task under the different algorithms when agents do not change their types throughout the task. Here again, using ConvCPD with $n_t = 20$ the team performs just as well (statistically) as the case of N Information. In the case where $n_t = 16$ it takes the team a bit longer to finish but still no more than the case of no information. When n_t is 14 or 10, or when using BCPD, however, the number of timesteps it takes the team to complete the task is higher than the no information case. Overall, our results indicate that with the proper window length, our novel CNN-based CP-detection algorithm performs better than the existing alternatives and can be used without any prior knowledge regarding whether agents' types are stationary or not.

6 Conclusions

This paper considered an extended version of the ad hoc teamwork problem in which agents can change their behavior types through the task. We approached the resulting problem by treating it as a change-point detection problem. Then, we solved it efficiently by proposing a new change-point detection algorithm based on convolutional neural networks. The proposed algorithm's efficacy over classical Bayesian changepoint detection algorithms was verified by experiments in a modified predator-prey domain. The experiments reveal that the algorithm improves performance even when there are no changepoints and hence can be added as an additional layer on current reasoning algorithms.

This paper opens several interesting possibilities for future research. We wish to investigate the problem of detecting changepoints in parameterized types by the proposed ConvCPD, and hope to solve this problem in interesting domains/settings like the more complicated variant of the modified Pursuit Domain in [Albrecht and Ramamoorthy, 2015] and Half Field Offense in robot soccer [Hausknecht *et al.*, 2016]. In addition, whereas the current MCTS planning algorithm executes its entire look-ahead with the current models, there is an opportunity to augment our approach with novel planning algorithms that take predicted future type changes into account.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

[3] In the no information case, the ad hoc agent uses CP-unaware reasoning for figuring out the agent's type.

References

[Albrecht and Ramamoorthy, 2015] Stefano V. Albrecht and Subramanian Ramamoorthy. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. *arXiv*, Jun 2015.

[Albrecht and Stone, 2017] Stefano V Albrecht and Peter Stone. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 547–555. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[Albrecht and Stone, 2018] Stefano V. Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 5 2018.

[Albrecht *et al.*, 2016] Stefano V. Albrecht, Jacob W. Crandall, and Subramanian Ramamoorthy. Belief and truth in hypothesised behaviours. *Artificial Intelligence*, 235:63–94, 6 2016.

[Aminikhanghahi and Cook, 2017] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

[Auer *et al.*, 2002] Peter Auer, Nicol Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2/3):235–256, 2002.

[Barrett *et al.*, 2011] Samuel Barrett, Peter Stone, and Sarit Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. *Autonomous Agents and Multiagent Systems (AAMAS)*, (May):567–574, 2011.

[Barrett *et al.*, 2012] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. Learning teammate models for ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, pages 57–63, 2012.

[Benda *et al.*, 1986] Benda, V Jagannathan, and R Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. (BCSG201028), 1986.

[Brown, 1951] George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[Fearnhead and Liu, 2007] Paul Fearnhead and Zhen Liu. Online inference for multiple changepoint problems. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 69(4):589–605, 2007.

[Forney, 1973] G. D. Forney. The viterbi algorithm. *Proc. IEEE*, 61(3):268–278, Mar 1973.

[Fridman, 2018] Lex Fridman. 6.S094: Deep Learning for Self-Driving Cars. (January), 2018.

[Hausknecht *et al.*, 2016] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.

[Hernandez-Leal *et al.*, 2017] Pablo Hernandez-Leal, Yusen Zhan, Matthew E Taylor, L Enrique Sucar, and Enrique de Cote. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 31(4):767–789, 7 2017.

[Kuncheva, 2013] Ludmila I. Kuncheva. Change Detection in Streaming Multivariate Data Using Likelihood Detectors. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1175–1180, 5 2013.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[MacAlpine *et al.*, 2014] Patrick MacAlpine, Katie Genter, Samuel Barrett, and Peter Stone. The RoboCup 2013 drop-in player challenges: Experiments in ad hoc teamwork. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 382–387. IEEE, 9 2014.

[Meng *et al.*, 2017] Zhao Meng, Lili Mou, and Zhi Jin. Hierarchical RNN with Static Sentence-Level Attention for Text-Based Speaker Change Detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*, pages 2203–2206, New York, New York, USA, 2017. ACM Press.

[Niekum *et al.*, 2015] Scott Niekum, Sarah Osentoski, Christopher G. Atkeson, and Andrew G. Barto. Online Bayesian change-point detection for articulated motion models. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):1468–1475, 2015.

[Nikolaidis and Shah, 2013] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 33–40, Mar 2013.

[Nikolaidis *et al.*, 2014] Stefanos Nikolaidis, Keren Gu, Ramya Ramakrishnan, and Julie Shah. Efficient Model Learning for Human-Robot Collaborative Tasks. *arXiv*, May 2014.

[Page, 1954] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1-2):100–115, 6 1954.

[Reeves *et al.*, 2007] Jaxk Reeves, Jien Chen, Xiaolan L. Wang, Robert Lund, and Qi Qi Lu. A review and comparison of change-point detection techniques for climate data. *Journal of Applied Meteorology and Climatology*, 46(6):900–915, 2007.

[Stone *et al.*, 2010] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 7 2010.

[Sugiyama *et al.*, 2008] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V. Buenau, and Motoaki Kawanabe. Direct Importance Estimation with Model Selection and Its Application to Covariate Shift Adaptation, 2008.

[Tambe, 1996] Milind Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, pages 80–87. AAAI Press, 1996.

[Xuan, 2007] Xiang Xuan. *Bayesian inference on change point problems*. PhD thesis, University of British Columbia, 2007.