DISCLAIMER:

This document does not meet the
current format guidelines of
the Graduate School at
The University of Texas at Austin.

It has been published for
informational use only.

Ad Hoc Teamwork with Behavior Switching Agents

APPROVED BY

SUPERVISING COMMITTEE:

Peter Stone, Supervisor

Shani Alkoby

Ad Hoc Teamwork with Behavior Switching Agents

by

Manish Chandra Reddy Ravula

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2019

Dedicated to my parents, to whom I owe everything good and great in my life. Without them, none of this would have been possible.

Acknowledgments

I wish to thank the multitudes of people who helped me through the course of my Masters' program.

I would like to thank my advisor, Prof. Peter Stone at The University of Texas at Austin. His depth of knowledge in the subject matter and creativity in solving problems helped pave new research directions for this work. He was extremely resourceful and helped me interact with many other relevant subject matter experts whenever I needed external help. His quick responses and patient answers have helped me push through many slumps in the course of my research. The countless meetings I've had with him shaped my entire outlook on research and turned my life around for the better.

I would also like to thank Shani Alkoby for being the best mentor I could ask for. She was extremely patient and supportive throughout the course of this research work. Her kind words, positive outlook and encouragement helped me push through the tough times. Without her, the research wouldn't have had progressed this far.

I would finally like to thank my friends and roommates, thank you for listening, offering me advice, and supporting me through this entire process. Special thanks to my roommate: Suraj Pawar, my brother: Jayanth, my UT Friends: Karthikeya, Haresh, Sid, Eddy, David, Ishan, my college friends: Arish, Sahil, Udbhav, Srikar, Sankrandan, Praneeth, Rajkumar, Dhruva, Tanya, Nishant, Hriday and many more.

Ad Hoc Teamwork with Behavior Switching Agents

Manish Chandra Reddy Ravula, M.S.
The University of Texas at Austin, 2019

Supervisor: Peter Stone

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, I 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 thesis, I 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, I show that it outperforms existing Bayesian CPD algorithms.

Table of Contents

List of Tables

List of Figures

Chapter 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 [25] 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 [33]. Several works approach this setting by assuming that every agent behaves according to one out of a set of predefined behaviors [22, 25, 2, 1, 6, 8, 7]. 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 finish the task at hand. If the types are sufficiently descriptive and the planning algorithms are capable enough, the agent's beliefs regarding the other agents' type will rapidly converge leading to a successful completion 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 work, I 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. I 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. I 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 I find that our algorithm outperforms other CPD algorithms in detecting and adapting to changes in agent types.

1.1 Related Work

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

1.1.1 Type-Based Ad Hoc TeamWork

Approaches in ad hoc teamwork broadly fall into two categories based on how the ad hoc agent models the rest of the team [4] . The well-studied first category involves modeling agents individually with distributions over action probabilities at each timestep [18, 11]. The second approach involves modeling the group as a whole and its joint action/planning dynamics [34, 36]

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 MCTS (Monte Carlo Tree Search) for planning actions accordingly [7, 33, 1, 3]. 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 [29, 28, 10].

All of the above works assume however, that the teammates' types remains fixed and do not account for type switches. The only work that does consider non-stationary teammates [21] 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. This helps the learning algorithm decide when it is time to start modeling the teammate's behavior as a new type instead of updating the probabilities of the current types.

1.1.2 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 [31], speech and image analysis, human activity analysis, and robotics [5]. Various algorithms have been proposed to detect and track these changes, both offline and online. Algorithms like the CUMSUM [30], KLIEP [35] and SPLL [23], 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 ([38, 12, 27]) 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 [26, 39, 13] 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 processes. 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 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 work however, does not make this assumption and is specifically tailored to work with non-stationary agent models.

1.2 Preliminaries

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

1.2.1 Model

I 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. I 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).

I 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, 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, I will assume that the types are characterized without the need of parameters.

To simplify the exposition, I assume that I control a single agent, i, which reasons about the behavior of another agent, j. I also assume that i knows j's action space A_j and that it can observe j's past actions, i.e. $a_j^{t-1} \in s_i^t$ 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 $\theta_j^* \in \Theta_j$. I 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, I assume that agent j will change its type during the process at a number of chosen time points $\Lambda = \{\lambda_1, ..., \lambda_k\}$ set extraneously.

Our goal is two fold. First, I 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, I aim to adapt the planning method to cope with these changes.

1.2.2 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, our agent will use the *MAP type estimation* method as suggested in the work of [3] 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 1.

1.2.3 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 [15] 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, I use a simple, domain-specific planning algorithm

Algorithm 1 MAP Type Estimation

Given type space Θ, initial belief $P(\theta_i|H_i^t)$

Output: Type estimates at each timestep, $\hat{\theta}_t$

1: **for** each timestep t > 0 **do**
2: Observe action a_m^t of m^{th} agent
3: **for** each type θ_i in type space Θ_j **do**
4: $P(\theta_i|H_i^t) \leftarrow P(a_m^t|\theta_i) * P(\theta_i|H_i^t - 1)$
5: **end for**
6: set $\hat{\theta}_t \leftarrow \text{argmax}_{\theta_i}(P(\theta_i|H_i^t))$
7: **end for**

that is described in Section 5.4.

Chapter 2

Proposed Methods

2.1 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. An example from a simple Gaussian time-series model is demonstrated in Figure 2.1.

Hence, identifying change points in this way can be detrimental to fast inference of the new type after a change occurs. We aim to integrate this idea into the original inference framework 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 resetted 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. This strategy is termed CP-Adjusted Inference as illustrated in

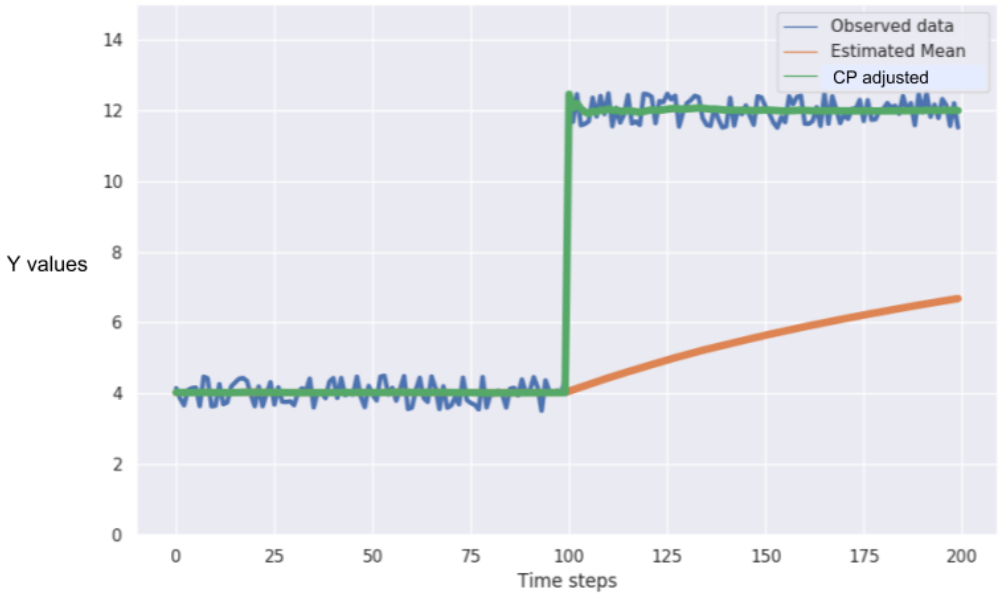

Figure 2.1: Illustration of difference between CP-Aware Inference and Naive (CP-Unaware) Inference. The data is sampled from $x \sim \mathcal{N}(4, 0.1)$ and $x \sim \mathcal{N}(12, 0.1)$ later on. The green and read lines illustrate running estimates of means computed from Maximum Likelihood Estimation with and without the awareness of the change point. The second estimate catches up to the true value at the 450th timestep.

Figure 2.1.

Since the choice of the change point algorithm is not obvious, 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.

2.1.1 Convolutional CPD Network

Convolutional Neural Networks [24] 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 is illustrated in figure 2.2. 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 radically 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]. And 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.

[1] In the extreme-ideal case, the types would be sufficiently different to not have similar likelihoods for the same action, since each type is different enough to generate a different action.

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 $n_{classes}$, given by equation 2.2.

$$n_{classes} = {}^{|\Theta|}P_2 + 1 \tag{2.1}$$

$$n_{classes} = |\Theta| \times (|\Theta| - 1) + 1 \tag{2.2}$$

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

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).

1: $out1 \leftarrow ConvCP_1.forward(L^t)$
2: $p_c^{m,n} \leftarrow Softmax(out1)$
return $p_c^{m,n}$

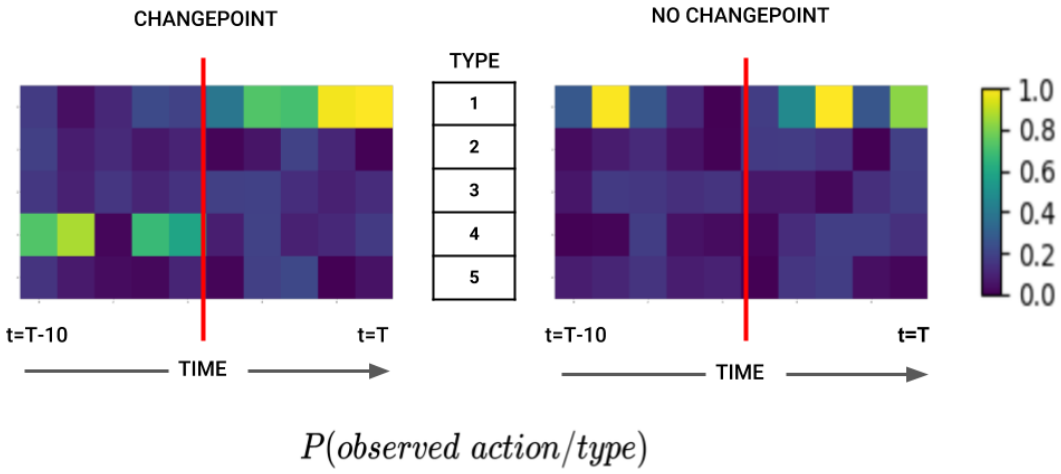

Figure 2.2: Image-like representation of $P(a^T|\theta_i)\ \forall\{\theta_i \in \Theta\ ,\ |\Theta| = 5\}$ where a^T is the action observed at timestep T. The image-like patterns are starkly different for change points vs no change point.

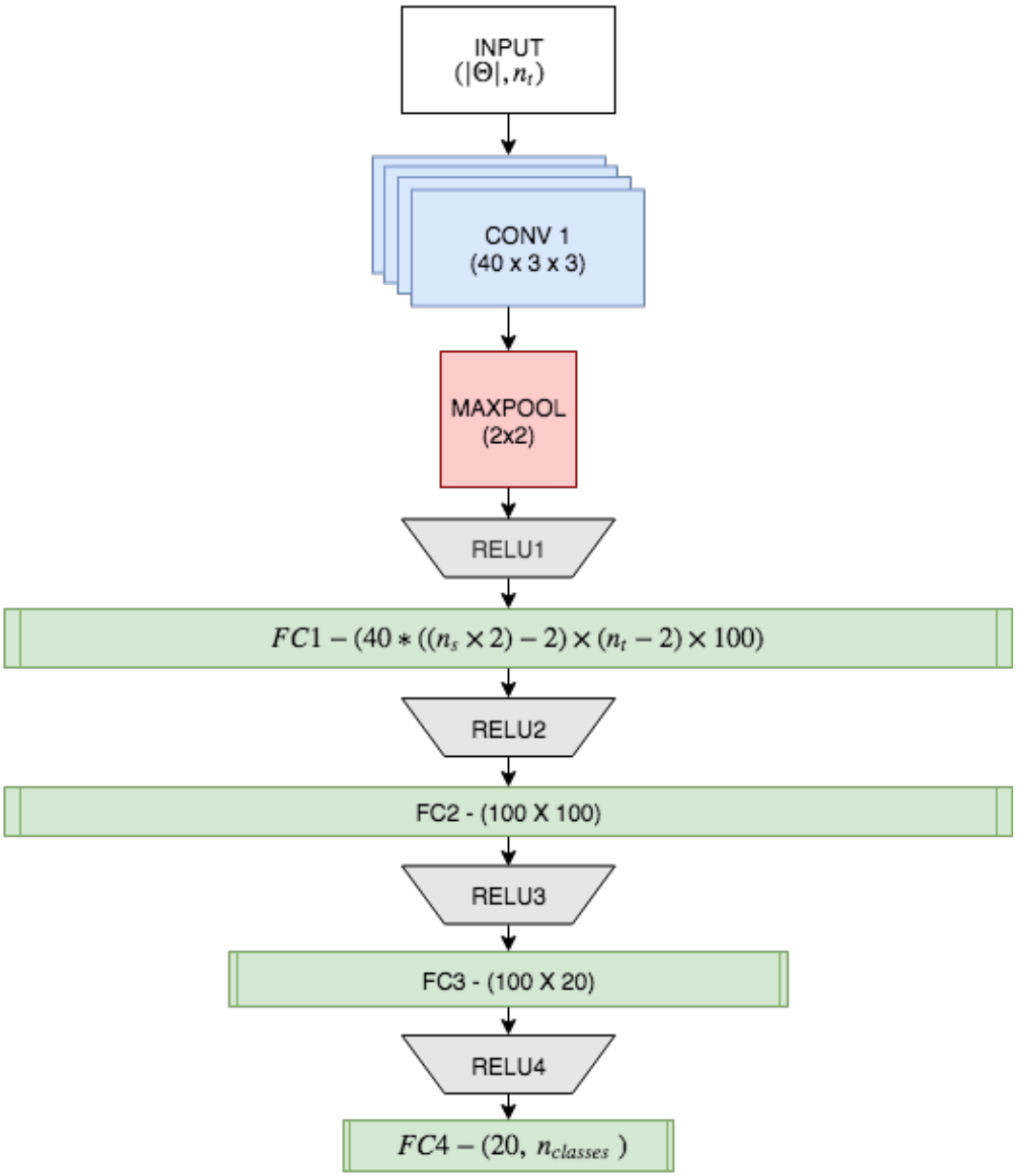

Figure 2.3: Architecture of the CPD Network used in our experiments. The choice of layer sizes are specific to our experiments and can be changed/resized accordingly for other applications.

Chapter 3

Experiments

3.1 Experimental Evaluation

I provide a detailed experimental evaluation of the algorithm in two domains; the modified predator prey domain and a multi-agent collision avoidance navigation setting. The first domain has types guided by the simple A^* navigation algorithm with very limited, discrete action space. The second domain has a more complex ORCA collision avoidance algorithm with a continuous action space, making the domain more realistic and natural. [9].

3.1.1 Setting 1 - Modified Predator Prey Domain

We modified the classic Predator Prey Domain to demonstrate a task which requires tracking type switches and adapting swiftly.

3.1.1.1 Domain Description

The first experimental 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., can not 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 chooses a different action. 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. This can be done 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 3.1 depicts an example grid configuration of our domain where $n = 3$. Finally, we note that in our experiments, only one agent is an ad hoc agent trying to track the other agents' type. The other agent's type is randomly chosen at the beginning of simulation episode.

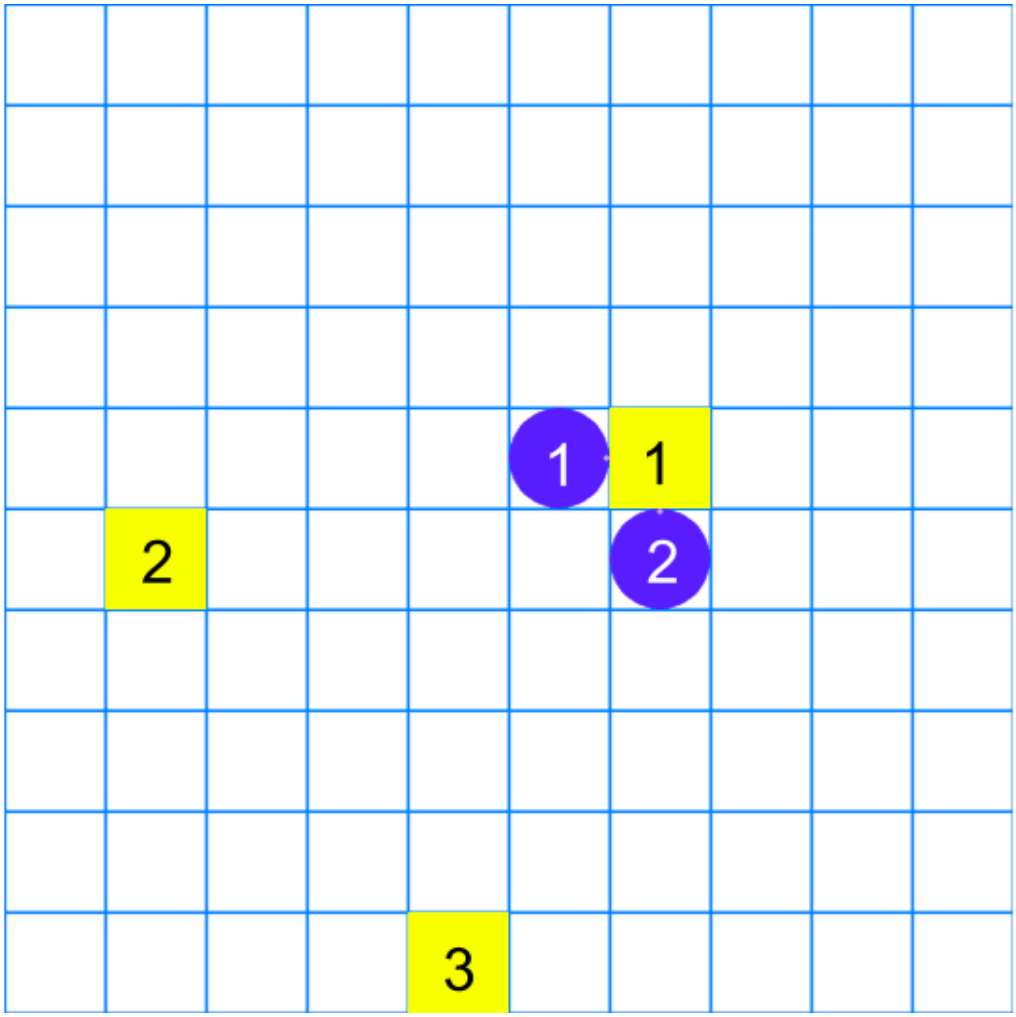

Figure 3.1: An example state in the modified predator prey domain with 3 preys (yellow squares) and 2 predators (blue circles). Here, the predators have successfully finished the task by executing CAPTURE action on the same prey.

3.1.1.2 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.[1] 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. Finally, the agent samples an action from this distribution and executes it if there are no conflicts with other agents. Conflicts in actions are handled randomly at each timestep by the simulation controller. The full algorithm for the predator agent's type is described in Algorithm 3.

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.

[1]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 pursuit based on the prey the pre-planned agent chose.

Algorithm 3 Template for Agent Types

Type θ_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 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

    Get all other valid moves

    Split probability of 0.1 over all the valid moves obtained from above

    Perform $\text{softmax}(p_i) = \frac{e^{\alpha * p_i}}{\sum e^{\alpha * p_i}}$ 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)

3.1.1.3 Bayesian Change Point Detection Algorithm

The widely used Bayesian model-based change point detection algorithm was first presented by Fearnhead and Liu [12]. 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:

$$Pr(c_i + 1 = t | c_i = s) = g(t - s) \tag{3.1}$$

where $g(x)$ is a probability distribution over time and $G(x)$ is its cumulative distribution function.

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:

$$L(s, t, q) = Pr(y_{s+1:t}|q) = \int Pr(y_{s+1:t}|q, \Theta) Pr(\theta) d\theta \tag{3.2}$$

We will 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 following notations:

$$Pr_t(j,q) = Pr(FC_t = j, q, \omega_j, y_{1:t}) \tag{3.3}$$

$$P_t^{MAP} = Pr(\psi_j, \omega_j, y_{1:t}) \tag{3.4}$$

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 [14].

A development of the above equation will result in:

$$Pr_t(j,q) = (1 - G(t-j-1))L(j,t,q)Pr(q)P_j^{MAP} \tag{3.5}$$

$$P_t^{MAP} = \overset{max}{j,q}\Big[\frac{g(t-j)}{1-G(t-j-1)}Pr_t(j,q)\Big] \tag{3.6}$$

Finally, the Viterbi path can be recovered by finding the j and q values that maximize (3.6) at time t. We then can repeat the process again in order to find the values which maximize (3.6) 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.

3.1.1.4 Results

In our experiments we simulate agents' behaviors at every timestep. The ad hoc agent runs Algorithm 2. The CNN in 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.

Influence of n_t on accuracy Table 3.1 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 makes sense, since the more timesteps the agent has for using 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 3.1: Change point detection accuracy and the mean squared error (MSE) of change point time estimation for different values of n_t.

Task performance in presence of CP 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 3.2 depicts the number of time-steps the 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 knows the pre-planned agent's type in any timestep, i.e., has perfect information, the number of timesteps needed for successfully completing the task, as can be seen from the figure, is expected to be the lowest possible. Thus, we consider this case to be our 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 the figure. 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,

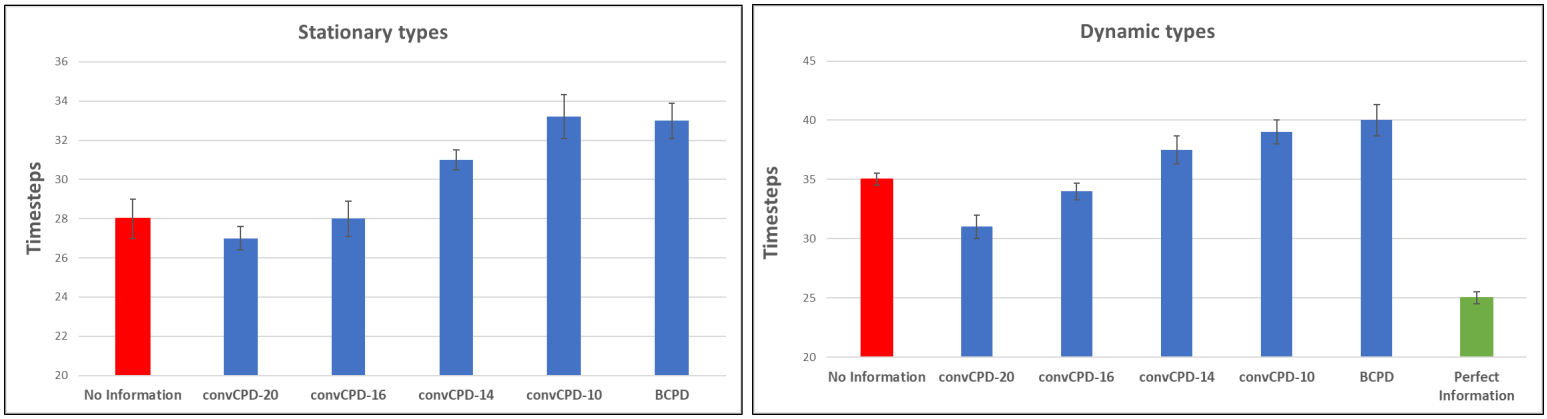

Figure 3.2: The number of timesteps required for completing the task using the different change point detection algorithms for both stationary and dynamic types.

i.e., without any awareness to the fact that agents are changing their types throughout the task.[2] If using the BCPD however, the number of timesteps it takes the team to complete the task is the highest observed. This happens because if for some reason, the last few timesteps assign high probability to a type that is not the true type, and the ad hoc agent recently reset the probabilities, then it would believe the actual type to be other than the right one. As a result, the ad hoc agent will plan according to the wrong type and move to the wrong prey, which will lead to a delay in completing the task.

Task performance in the absence of CP Finally, in many real life situations, agents may not know in advance whether their teammates will change

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

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 3.2 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 achieves the best results. 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 the 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 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.

3.1.2 Setting 2 - Ad hoc Multi-Agent Navigation

This domain is inspired by work from [17], where the authors aim to solve the problem of ad hoc navigation in multi-agent systems. In this setting, each agent has a goal position to navigate to, and does so by employing one of a subset of navigation algorithms. The paper considers two popular navigation algorithms - Optimal Reciprocal Collision Avoidance (ORCA) [37], Social Force Model [20] - and their variants. The combination of goal position and navigation algorithm together define and agent's type in this setting while co-operation is enforced by rewarding the agents to reach their goal positions as soon as possible with the least number of collisions. This problem is very similar to our ad hoc teamwork problem, and is even solved in with the paradigm of differentiating type estimation and planning. For type estimation, the authors use a slightly modified version of the Bayesian inference in Algorithm 1. For planning, the authors use the Hindsight Optimization [16] to pick a velocity at each time-step. Hindsight Optimization is a technique to solve MDP type planning problems by determinizing transition functions first, followed by solving these deterministic MDPs and then approximating the current Q-Value using these solutions.

The authors test their methodology on different arrangements of agents' beginning and target goal positions. The experiments show that the ad hoc agent performs better than naively dodging obstacles or simply reacting to its neighbors. We keep the assumption about ad hoc agent's full observability of other agents.

3.1.2.1 Domain Description

For the sake of our experiments, we pick their *Circle* setting since most other settings provide a very narrow set of indistinguishable types to chose from. In this arrangement, agents start off at equally spaced *interest points* on the circumference of a circle and navigate to goal of choice within the *interest points*. Figure 3.3 illustrates this setting, with 32 *interest points*. For the experiments, I fix the number of *interest points* to 6, with 5 non-adhoc agents and 1 ad hoc agent. The task is said to be completed successfully if each agent reaches a unique *interest point* with no two agents ending up at the same *interest point* at the end of the run. This is possible if the adhoc agent successfully reasons about the types of all other agents, identifies the un-targeted *interest point* and navigates accordingly. If an agent changes its type midway (by extension, the goal is also changed), the ad hoc agent must successfully identify the switch and navigate to the new goal accordingly so it doesn't end up at an *interest point* already occupied by another agent.

The task performance is measured with the help of two metrics apart from simple Success/Failure. The first metric is the total simulation run-time which measures the group's performance as a whole. Since sometimes group delays could be caused by non ad-hoc agents, a second metric is used to judge the performance of the ad hoc agent. This metric is simply the length of the path the adhoc agent takes while navigating to the goal. If the ad hoc agent had perfect information, it would take the shortest route, while any kind of mistakes in inference increase the path length.

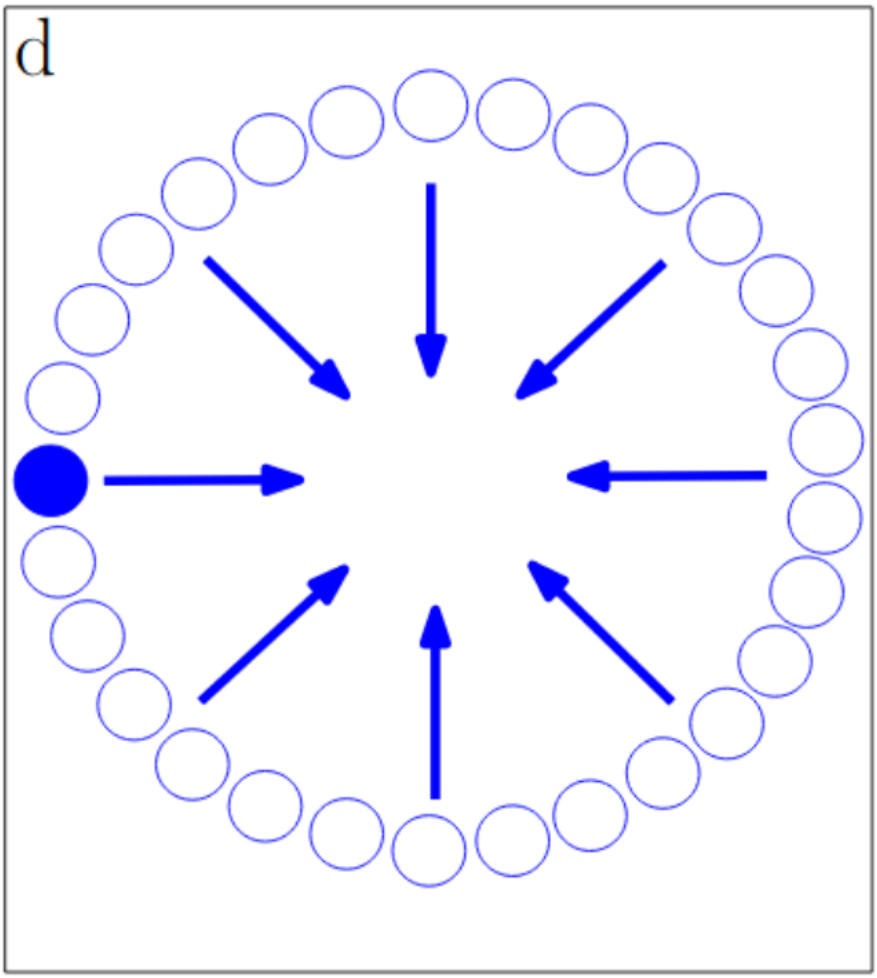

Figure 3.3: The Circle Setting

3.1.2.2 Agent Types

Each non-adhoc agent uses ORCA [37] to navigate to its goal position. ORCA is a local collision-avoidance navigation algorithm, that guarantees a collision-free path independently without explicit communication with other agents (assuming other agents use ORCA too). It does so by formulating the collision avoidance problem in terms of velocity obstacles and solves a linear program to find the velocity closest to the desired velocity avoiding velocity obstacles as much as possible. ORCA works under the assumption that any incoming agent takes half of the responsibility of avoiding pairwise collisions.

Note, however, that ORCA is a local navigation algorithm, which means it is responsible solely for avoiding collisions in the very near future for each time-step. The global navigation plan has to be derived with the help of a separate global planner. The ORCA algorithm requires a preferred velocity as a parameter at every time-step from the global planner. The preferred velocity must be chosen such that the overall path results in reaching the desired goal.

Each agent's type in our modified setting is solely decided by the goal *interest point*, just like in the previous experimental section (3.1). The timing and nature of type switching is similar to Experiment 1 - the type can be switched at any time throughout the task, and any number of times within a task.

3.1.2.3 Ad Hoc Agent

The Ad Hoc agent can observe all of the agents with noisy sensors reading position and velocity.

$$\hat{p}_{a_i} \sim \mathcal{N}(p_{a_i}, \sigma_p^2) \tag{3.7}$$

$$\hat{v}_{a_i} \sim \mathcal{N}(v_{a_i}, \sigma_v^2) \tag{3.8}$$

where p_{a_i} , v_{a_i} are the true position and velocity of agent i (a_i) while $\hat{p}_{a_i}$, $\hat{v}_{a_i}$ are the observed position and velocity of a_i by the ad hoc agent. σ_p^2 and σ_v^2 denote the variance in position and velocity measurements respectively.

The ad hoc agent plans using the same Hindsight Optimization [17] method as in the reference paper.

3.1.2.4 Results

The experimental configuration parameters are provided in table 3.2.

The data-set collection procedure and the network architecture are the same as in Experiment 1 (Section 3.1.1). The only modification is in the way the network is trained. Instead of training the network from scratch using the data collected, we adopt a neural network distillation procedure, to learn from a bigger network. For this purpose, we use the VGG16 image classification network [32], which is a deep convolution neural network to classify natural images into one of 1000 common object classes. The network has 13 convolutional layers and 3 fully connected (FC) layers. The architecture is illustrated

Property	Value
Number of non ad hoc agents	5
Grid Size	100m x 100m
ORCA - Max Speed for the Agent	2 m/sec
ORCA - Time Step Length	0.5 sec
ORCA - Agent Radius	1m
Variance in position obsv. noise p_{a_i}	$2 \times AgentRadius$
Variance in velocity obsv. noise v_{a_i}	$2 \times AgentMaxSpeed$

Table 3.2: Values of agent properties set in the experiments

in 3.4. We fine-tune the network on the change-point detection data-set, and then use the final layer's outputs to train our CPD network 2.3. This way, we make use of the representational ability of the VGG16 network to improve accuracy of our smaller network.

To train the VGG16 network, the likelihood arrays were up-sampled and resized to match the expected input size (224x224x3). A final fully connected layer (FC9) was added on top of FC8 to decrease the number of output class labels from 1000 to 30, since we only have 30 (Equation 2.2) possible type change pairs. Only the last two fully connected layers were fine-tuned. Once the training is completed, the last layer's (FC9) outputs were extracted as targets for the ConvCPD network (2.3) using the exact same training procedure as in the previous experiment.

I fixed the size of the time window (n_t) to be 20 since that performed the best in the previous experiments. Further, lower (n_t)s are not very amenable to up-sampling which is necessary to fine-tune the VGG16 network.

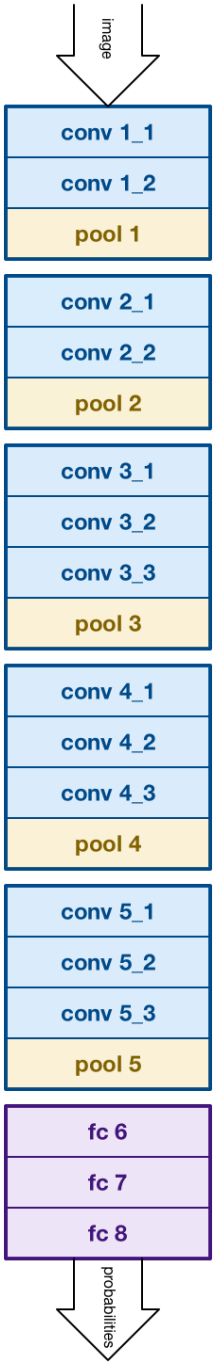

Figure 3.4: Classic VGG16 Architecture

Accuracy Table 3.3 compares the accuracy of ConvCPD vs BCPD in this setting.

	ConvCPD	BayesianCPD
Accuracy	95%	90%
MSE	1.5	3.5

Table 3.3: Comparison of change point detection accuracy and the mean squared error (MSE) of change point time estimation.

As we can see, ConvCPD performs better than Bayesian CPD in terms of accuracy.

Task performance in absence of CP To compare task performance, let us first consider the case of stationary types. Figure 3.5 compares the average time taken to finish the task when agents don't switch types. The run time is the lowest when the ad hoc agent doesn't mistakenly wander around due to false-positive inference of change points.

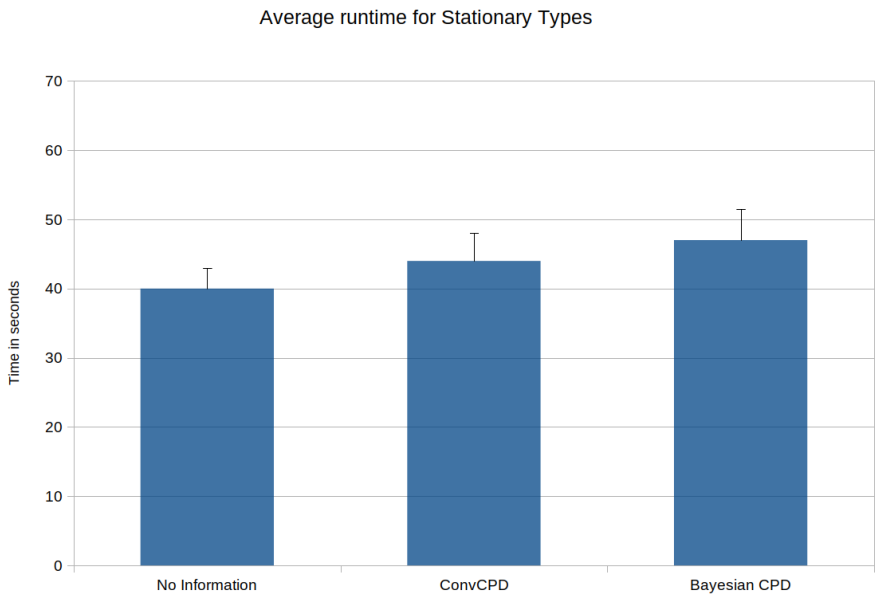

Figure 3.5: Run Time for Stationary Types

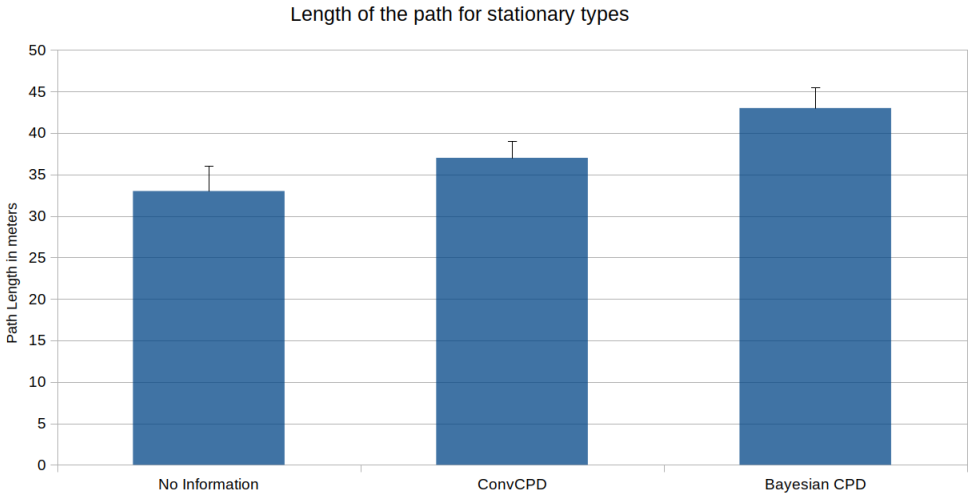

Figure 3.6: Path Length for Stationary Types

In this case, the *No Information* case performs the best, and the ConvCPD and BCPD trail closely behind. ConvCPD performs slightly better than BCPD due to the improved accuracy in measurement of change-points.

The path length shows a similar trend in Figure 3.6. The path length is again lowest in the case of *No Information*, with the CPD algorithms trailing behind. This order is explained similarly to the above result: The ad-hoc agent covers more distance while running a CPD detection algorithm due to the occasional false-positive inference of change-points causing it to change course.

Task performance in presence of CP In the second case, I consider task performance in the presence of change points. I compare the *No Information* case with ConvCPD, BCPD as well as the *Perfect Information* case. As expected, the *Perfect Information* performs best in both the run time and path length (3.7,3.8). This performance ranking is expected since with perfect information, the agent can always take the shortest path thus completing the task in the shortest time. Here too, ConvCPD performs slightly better than BCPD for both metrics.

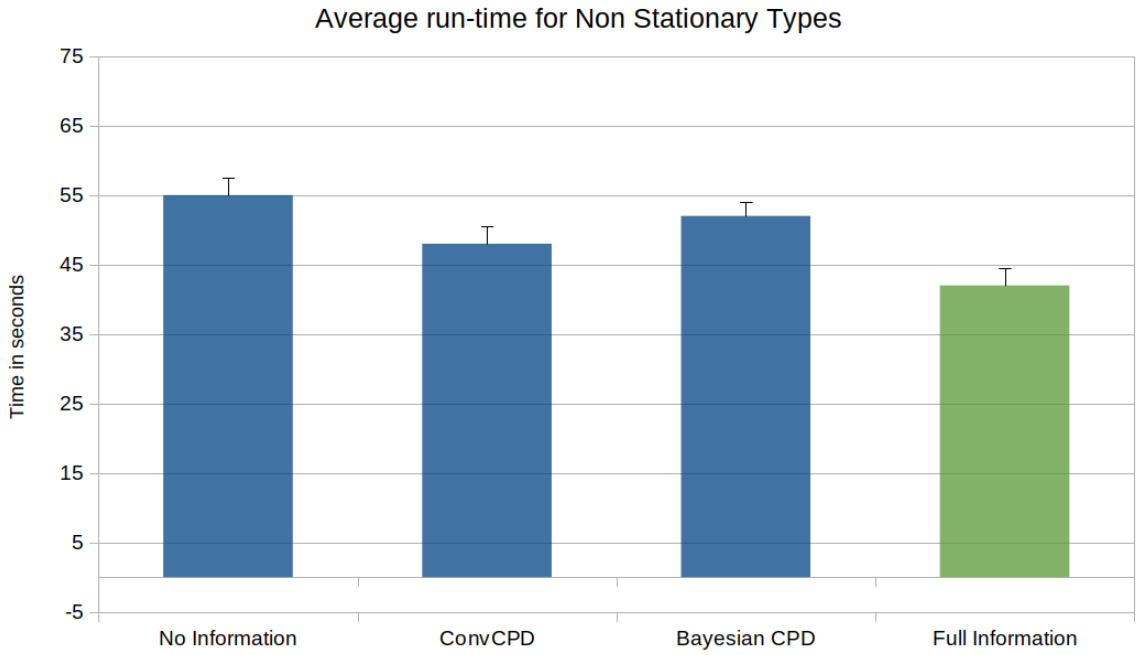

Figure 3.7: Run Time for Non Stationary Types

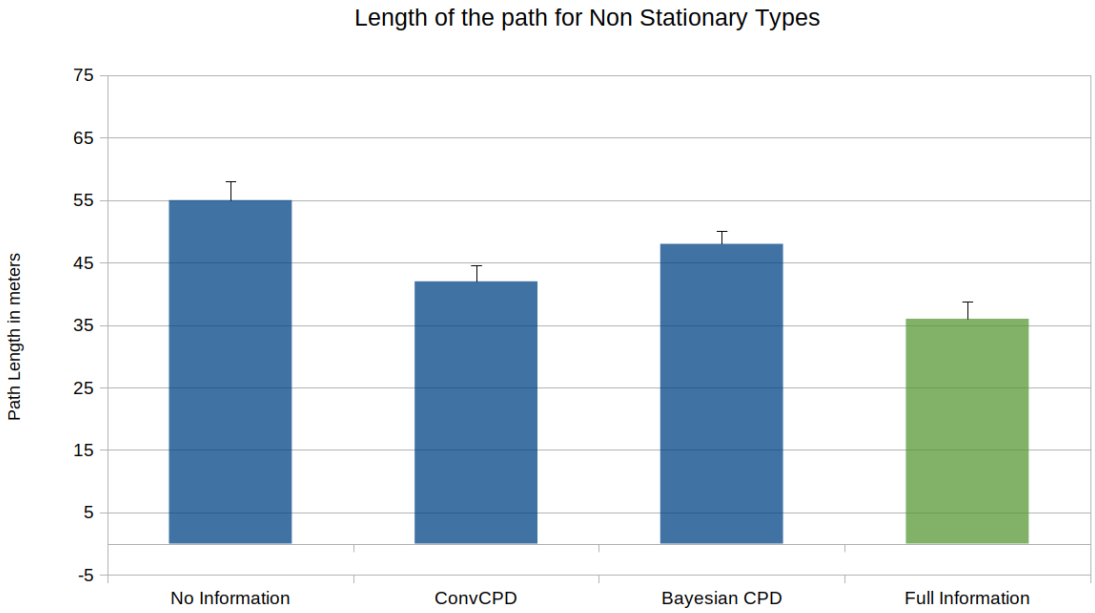

Figure 3.8: Path Length for Non Stationary Types

Comparing Experiment 2 to Experiment 1, it can be noticed that the disparity in performance between BCPD and ConvCPD is reduced for Experiment 2. This decrease in disparity could be explained by the fact that the probability distributions for type definitions in Experiment 2 originate from observational noise which is a simple normal distribution. BCPD is well-suited to work on such models which shows in its performance gains. Despite this advantage, ConvCPD still performs slightly better than BCPD.

3.1.3 Summary of Experiments

The experimental domains introduce ad hoc team work tasks that require active detection of change-points and adapting quickly, as described in our hypothesis. In these settings, the results verify that the proposed ConvCPD algorithm performs better than BCPD algorithm and could also be used in absence of change points. Thus, the initial hypothesis that resetting posteriors on correct detection of changepoints is verified by comparing methods that accurately detect changepoints with the *No Information* case, which has the worst performance.

Chapter 4

Conclusion

4.1 Conclusions

This work considered an extended version of the ad hoc teamwork problem in which agents can change their behavior types through the task. I approached the resulting problem by treating it as a change-point detection problem and 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 on a modified predator-prey domain,as well as a multi agent social navigation setting. 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 work 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 more interesting domains/settings like the Pursuit Domain [2] and Half Field Offense in robot soccer [19]. Further, we also wish to study how existing generic planning algorithms could be improved to help agents handle the changes in agent behaviors.

Bibliography

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

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. 6 2015.

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] David Carmel and Shaul Markovitch. Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental & Theoretical Artificial Intelligence*, 10(3):309–332, 1998.

[11] Eddie Dekel, Drew Fudenberg, and David K Levine. Learning to play bayesian games. *Games and Economic Behavior*, 46(2):282–303, 2004.

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

[13] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model. 12 2016.

[14] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

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

[16] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Anytime navigation with Progressive Hindsight optimization. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 730–735, Sep 2014.

[17] Julio Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria L Gini. Moving in a crowd: Safe and efficient navigation among heterogeneous agents. In *IJCAI*, pages 294–300, 2016.

[18] Kristian J. Hammond. Case-Based Planning: A Framework for Planning from Experience. *Cognitive Science*, 14(3):385–443, 7 1990.

[19] 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.

[20] Dirk Helbing and Peter Molnar. Social Force Model for Pedestrian Dynamics. *arXiv*, May 1998.

[21] 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.

[22] Richard Jones. Rapid Game Development In Python. *Open Source Developoers' Conference*, pages 84–90, 2005.

[23] 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.

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

[25] 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.

[26] 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 Knowl-*

edge Management - CIKM '17, pages 2203–2206, New York, New York, USA, 2017. ACM Press.

[27] Scott Niekum, Sarah Osentoski, Christopher G Atkeson, and Andrew G Barto. Champ: Changepoint detection using approximate model parameters. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 2014.

[28] Stefanos Nikolaidis, Keren Gu, Ramya Ramakrishnan, and Julie Shah. Efficient model learning for human-robot collaborative tasks. arxiv. 2014.

[29] Stefanos Nikolaidis and Julie Shah. Human-Robot Cross-Training: Computational Formulation, Modeling and Evaluation of a Human Team Training Strategy. 2017.

[30] E. S. PAGE. CONTINUOUS INSPECTION SCHEMES. *Biometrika*, 41(1-2):100–115, 6 1954.

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

[32] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, Sep 2014.

[33] 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.

[34] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[35] 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.

[36] 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.

[37] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[38] Xiang Xuan. Bayesian Inference on Change Point Problems. 2004.

[39] Ruiqing Yin, Herv Bredin, and Claude Barras. Speaker Change Detection in Broadcast TV Using Bidirectional Long Short-Term Memory Networks. In *Interspeech 2017*, pages 3827–3831, ISCA, 8 2017. ISCA.