Decentralised Detection of Emergence in Complex Adaptive Systems

EAMONN O'TOOLE, VIVEK NALLUR and SIOBHÁN CLARKE, Trinity College Dublin

This paper describes Decentralised Emergence Detection (DETect), a novel distributed algorithm that enables agents to collaboratively detect emergent events in Complex Adaptive Systems (CAS). Non-deterministic interactions between agents in CAS can give rise to emergent behaviour or properties at the system level. The nature, timing and consequence of emergence is unpredictable and may be harmful to the system or individual agents. DETect relies on the feedback that occurs from the system level (macro) to the agent level (micro) when emergence occurs. This feedback constrains agents at the micro-level, and results in changes occurring in the relationship between an agent and its environment. DETect uses statistical methods to automatically select the properties of the agent and environment to monitor, and tracks the relationship between these properties over time. When a significant change is detected, the algorithm uses distributed consensus to determine if a sufficient number of agents have simultaneously experienced a similar change. On agreement of emergence, DETect raises an event, which its agent or other interested observers can use to act appropriately. The approach is evaluated using a multi-agent case study.

General Terms: Emergence, Complex Adaptive Systems

Additional Key Words and Phrases: Decentralised, Detection, LASSO, cumulative sum, distributed

ACM Reference Format:
Eamonn O'Toole, Vivek Nallur and Siobhan Clarke, 2015. Decentralised Detection of Emergence in Complex Adaptive Systems. *ACM Trans. Autonom. Adapt. Syst.* , , Article ( ), 30 pages.
DOI: 10.1145/3019597

1. INTRODUCTION

Future large-scale systems will both host and depend on an array of Complex Adaptive Systems (CAS), which are systems composed of a large number of independent agents that interact, adapt and learn [Holland 1992]. Agents are software components, systems or people, each with unique, possibly conflicting, goals. CAS are capable of spanning organizations and large geographic areas with no centralised control or monitoring [Northrop et al. 2006]. They form organically through the non-deterministic interactions of their agents and it is impossible to predict the behaviour of these systems with any great degree of accuracy in advance [Mogul 2006]. Emergence is a hallmark of such systems [Ottino 2004], and refers to the appearance of persistent properties and behaviours (emergents) at the system (macro) level. These emergents are caused dynamically from the non-deterministic interactions between entities at the agent (micro) level [Wolf and Holvoet 2005]. However, despite being caused by agent interaction, the emergents are novel with respect to the underlying agents [Wolf and Holvoet 2005]. Typical examples of emergence include traffic-jams, swarm formation of birds and the Internet [Fromm 2005b].

The interactions of agents at the micro-level have causal power over the emergents at the macro-level. The macro-level can also have a causal power over the micro-level, i.e., downward causation, where the emergent behaviour constrains the entities at the micro-level [Bedau 2002]. For example, a car in a traffic-jam contributes to traffic volumes, causing emergent congestion behaviour, and this emergent behaviour impacts the car by limiting its speed and also, possibly, its route. It is beneficial to be aware of emergence in a CAS so that harmful effects can be mitigated (a traffic jam) or beneficial effects leveraged (increased security when flocking). However, the scale, unpredictability and decentralised architecture of CAS means that detecting emergence in these systems is challenging, and compounded by the nature of emergence, which is itself dynamic and unpredictable.

This paper describes Decentralised Emergence Detection (DETect), a novel distributed algorithm that enables constituent agents to collaboratively detect emergent events in CAS. DETect's contribution is that it does not require a centralised controller or system monitor, and instead uses information from an agent's locality to facilitate detection. DETect automatically selects what information to monitor from an agent and its environment, removing the requirement for advance knowledge of expected emergence. The algorithm relies on the downward causation from the macro-level to the micro-level when emergence is present in the system, that naturally constrains the agents [Chalmers 2006; Bedau 2002]. *We hypothesise that this feedback can be detected by individual agents through changes in the statistical relationship between the agent and its environment.* By monitoring this relationship and sharing information when a change is detected, agents can collaboratively act as detectors of emergence.

2. BACKGROUND AND CHALLENGES

The concept of emergence first appeared in philosophy in the time of Socrates and has been studied in a variety of fields since [Di Marzo Serugendo et al. 2006]. It is a fundamental concept in complex systems and a necessary part of what qualifies complex systems as complex [Flake 1998; Mittal 2013]. An extensive body of literature exists that attempts to characterise emergence but, despite this, there is no widely accepted definition of emergence, though a number of characteristics are commonly discussed [Wolf and Holvoet 2005; Dessalles and Phan 2001; Fromm 2005a].

2.1. Emergence in CAS

Underpinning emergence is the notion that the whole is greater than the sum of the parts, or to put it another way, that order can arise from chaos [Kubík 2003]. In CAS, emergence refers to macro-level patterns, structures and properties arising in systems of decentralised interacting agents, where these patterns, structures and properties are not contained in the properties of its parts [Dessalles and Phan 2001]. This definition highlights a number of characteristics of emergence. First, these systems are decentralised with no centralised controller or monitor. Second, it is necessary to talk about these systems at two levels when discussing emergence: the individual agent (micro) and the global system (macro). Third, emergence is only possible in systems composed of autonomous agents who interact non-linearly and it is these interactions that cause the emergent behaviour or property. Finally, the emergent properties at the macro-level cannot be simplified to a composite of the properties at the micro-level.

Fromm describes a taxonomy of different emergence types [Fromm 2005b]. These are: *Type-I* (where the micro causes the macro level phenomenon) *Type-II* (where the macro level feeds back onto the micro level) and *Type-III* (where the micro-level agents learn and adapt to this feedback). Type-III is the most significant type of emergence as it results in the systems as a whole becoming reflexive [de Haan 2006]. This occurs because the macro-level now has a causal effect on the agents on the micro-level, causing the agents to learn and adapt their behaviour in response to the detected emergent behaviour. This adapted behaviour can in turn impact the emergent behaviour closing the circle of causality between layers.

The observer of the emergent behaviour or property is also a key consideration [Mittal 2013]. Muller uses the observer's identity to distinguish between weak and strong emergence [Muller 2004]. In weak emergence, the observer is external to the system looking at its state from a global viewpoint, for example, a human observing a flock of birds flying in formation. Strong emergence occurs when the agents themselves are the observers, identifying a phenomenon at the macro level that represents an evolution in the system they participate in. From the flocking example, individual birds identify the emergence of the flocking behaviour. The agents' ability to act as emergence

detectors requires that their view is sufficiently broad to identify the phenomenon as global rather than local. Such an identification involves a change in behaviour and so a feedback or bi-directional link is created, leading to the system becoming reflexive.

2.2. Detecting Emergence

Emergent behaviour and properties can be unexpected, undesirable and difficult to control, for example traffic congestion [Singh et al. 2013]. It is also possible that emergence can be beneficial, such as the increased security from predators offered by flocking. In either case, it is desirable to detect emergence when it occurs to mitigate harmful effects or exploit beneficial effects. It is possible for both internal and external system observers to recognise emergence when it occurs and therefore act as detectors. However, the characteristics of both CAS and emergence means that this is non-trivial.

Agents with knowledge of their environment and the ability to use past experiences can theoretically be used to detect these phenomena if they are themselves part of the system [Dessalles and Phan 2001]. This is possible if the agents know what to look for and have a sufficiently broad view of the system. However, CAS are composed of many parts, agents and subsystems that interact non-linearly in ways difficult to recognise, manage and predict [Maxwell et al. 2002]. This unpredictability results in unforeseeable connections and subsystems forming organically at runtime, and is compounded by the unpredictability of the emergence that results. This means that it is not possible to deploy specially appointed agents to detect emergence, as it is not always possible to know what to look for in advance.

Additionally, CAS are decentralised, a further obstacle to 'lookout' detector agents. This is especially true in large-scale CAS, such as the Smart Grids, or intelligent traffic management systems. Obtaining a global view of the system state to detect emergents becomes impractical, as that system state is constantly changing. Agents do have access to locally available information which is more timely, but this limited local view is insufficient to make claims about global behaviour and properties. Therefore, agents need to collaborate with one another, quickly aggregating individual local views to form larger and larger compound system views. Finally, emergence is transient in nature, forming at a certain point in the systems evolution before eventually evaporating, e.g, a traffic jam. As a result, any detector should be able to detect these transition phases to enable timely action to be taken in response to the new system state.

In short, an algorithm facilitating emergence detection in CAS should:

— Req #1: Occur at runtime;
— Req #2: Be decentralised across constituent agents;
— Req #3: Rely on locally available information;
— Req #4: Support autonomic selection of what data to monitor;
— Req #5: Support collaboration between agents to achieve a larger system view;
— Req #6: Detect formation and evaporation of emergence.

3. RELATED WORK

Variable-based approaches employ system-wide variables and statistical analysis to determine the existence and extent of emergence in a system. Such approaches allow detection of emergence to occur at runtime, but depend on a centralised architecture and assume access to global properties of the system to generate the required variables. Additionally, the specific variables needed for analysis must be known in advance. An example is Seth's work [Seth 2008], where both linear and non-linear time series analyses based on Granger Causality are used to define G-Emergence. A defined macroscopic property, such as the centre of a flock of birds, is emergent when it is statistically autonomous and caused by the microscopic properties (individual position

of each bird). A centralised framework to apply such a metric is outlined by Birdsey and Szabo [2014] where the system is monitored at intervals as it evolves. A similar approach is presented by Niazi and Hussain with the SECAS framework, which uses a distributed set of sensors to collectively sense the complex behaviour [Niazi and Hussain 2011].

Entropy, a measure of disorder in a system, is used by several variable-based approaches for detecting emergence [Fisch et al. 2010; Procházka and Olševičová 2015]. Emergence is the difference in system entropy from the process start to end, measured by the number of events [Mnif and Muller-Schloer 2006]. A mathematical approach for representing strong emergent properties of a system has been proposed by measuring entropy at different levels of abstraction [Bar-Yam 2004]. Chan [Chan 2011] acknowledges the importance of interactions between agents in creating emergence in a system. Here, detection of emergence is facilitated by monitoring a time-series of the number of interactions and state changes across all agents in the system. Emergence exists when this interaction metric departs from normality. These approaches all require knowledge of the global system state to calculate these measures.

De Wolf et al provide guarantees about system-wide behaviour in decentralised autonomic systems [De Wolf et al. 2005]. Users observe the evolution of macro-level properties when accurate models of micro-level properties are provided. Users must have knowledge of relevant, obtainable system-wide variables in advance. Grossman et al describe "Angle", a hierarchical framework to detect emergence in distributed clustered systems [Grossman et al. 2009]. Emergence occurs at a change point in time-series data. This framework is distributed, though the monitoring and analysis are done by static nodes, which limits its applicability to CAS with open, volatile agents.

Formal approaches use formal language and modelling to define emergence in a system before it is deployed. These techniques are used to both detect emergence at runtime [De Angelis and Di Marzo Serugendo 2015], and predict emergence at design-time. Kubik outlined one example [Kubík 2003], where a formal grammar characterises weak emergence. Emergence is the set of properties produced by agent interactions and cannot be derived by summing individual behaviours using superimposition. This approach is centralised and requires all possible agent behaviour to be known and unchanging. However, the use of superimposition across all agent languages leads to state-space explosion, increasing computational cost.

This problem is addressed in [Teo et al. 2013], where analysis is limited to only those states that are possible and of interest. However, relevant states must be known in advance. Szabo and Teo [Szabo and Teo 2015] also address state-space explosion where the degree of interactions between agents is used to identify the states of interest, similar to that described in Chan [Chan 2011]. However, this analysis occurs retroactively once the system simulation has completed, so discovering emergence cannot be used within the system itself. As a result, their applicability to CAS, where fundamentally unpredictable emergence may arise, is limited. Moshirpour presents a model-based technique that detects emergent behaviour at system design-time [Moshirpour et al. 2012], which can be used to determine the cause of the behaviour, and thus eliminate or control the emergence. Emergent behaviour is characterised by implied scenarios, which are types of behaviour present in the synthesised model of the system but are not explicitly defined in its specification.

Event-based approaches are a hybrid of the previous approaches, incorporating aspects from both statistical analysis and formal approaches. However, these approaches also rely on centralised architectures and design-time knowledge is required, making them unsuited for emergence detection at runtime in open CAS.

Chen et. al [Chen et al. 2008] describe emergent behaviour at different levels of abstraction based on defining event types. Simple and complex events are differentiated,

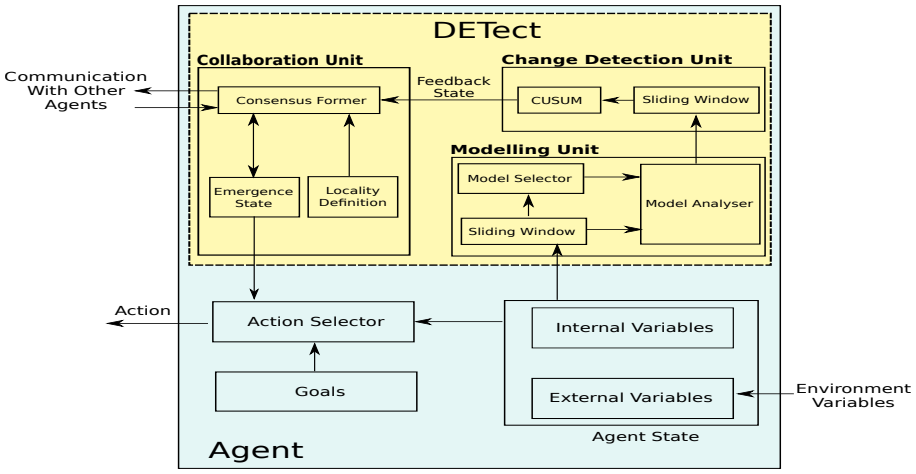

Fig. 1. DETect - Conceptual Architecture

with simple events formalised as agent state transitions when viewed from a particular level of abstraction. Complex events are a configuration of simple events where the configuration includes dimensions such as space or time. Statistical analysis is used to discover correlations between event types in simulations, enabling identification of relationships across abstraction levels of the system. The Mayet Architecture detects emergent unwanted behaviour in games at runtime [Lewis and Whitehead 2011] by monitoring the system and determining if its state conforms to a set of designer specified constraints. Emergent events are detected at runtime, but a centralised architecture is needed where the observer has access to global system state and design time knowledge is required to determine what constitutes a valid system state.

Decentralised detection of emergence at runtime in CAS requires that constituent agents in the system act as detectors. However, individual agents are not well-equipped to fulfil this role as no single agent has a sufficiently broad system view. In addition, emergence is inherently unpredictable and so it is impossible for agents to know in advance what properties of the system should be observed.

4. THE DETECT APPROACH

Decentralised Emergence Detection (DETect) is a novel distributed algorithm that enables agents to collaboratively detect emergent events when they occur. DETect is built on the concept that the presence of emergence in a system results in feedback from the macro-layer to the micro-layer, through downward causation, and that this feedback constrains the agents in the system [Chalmers 2006; Bedau 2002]. In our previous work we demonstrated that this feedback results in changes in the statistical relationship between individual agents and their local environment, including other agents, when emergence is present in the system compared to when there is no emergence [O'Toole et al. 2014]. The second insight is that as emergence is global in nature, agents involved should simultaneously experience a changed relationship as the emergent behaviour forms and evaporates in the system.

Combining these properties of emergence inspires the conceptual architecture of DETect and a modular design is used to achieve three key competences to facilitate emergence detection. This architecture is illustrated in Figure 1, with each competence encapsulated in one of three distinct units; the modelling unit, the change detection unit and the collaboration unit. The *Modelling Unit* provides agents with a general method

of modelling their statistical relationship with their local environment. It achieves this using properties of the agent (internal variables), such as speed or direction of travel and properties of the agent's environment (external variables), such as the number of agents near-by or their average speed. The *Modelling Unit* selects a subset of these variables autonomously at runtime to compose the model. Once selected, the statistical properties of the model are periodically measured as the system executes, with recent observations used to establish an expected baseline for the agent's current relationship with its environment. Next, the *Change Detection Unit* provides agents with an on-line means of detecting a sudden and statistically significant change in the relationship, that may indicate that the agent is experiencing feedback from emergence. However, it is not possible for any single agent to independently conclude the presence of emergence due to the limited scope of their system view. Therefore, the *Collaboration Unit* uses a distributed consensus algorithm to determine the proportion of agents simultaneously experiencing a similar change with the existence of an emergent event concluded if the proportion is sufficiently high. Once this agreement is reached, a detection event is raised. The remainder of this section describes each of these three components in detail.

4.1. Modelling Unit

Design: The first step to achieve decentralised emergence detection is to allow each agent to model its relationship with its environment. This relationship can be monitored for changes that *may* signify feedback from the macro-level. Agents may use multiple internal variables to describe themselves and multiple external variables to describe their environment, so there are possibly a huge number of possible relationships between these two variable sets. It would be computationally expensive for agents to monitor each of these individual relationships separately at runtime. A simplified statistical model is required that allows multiple relationships to be represented at once, as well as a means of selecting only those relationships that provide relevant information to be included.

$$y_i = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon_i \tag{1}$$

In DETect, the *Modelling Unit* takes the variables that describe the agent (internal variables) and the variables that describe the agent's environment (external variables) as input. We assume each agent knows its own internal variables and has knowledge about the external variables it cares about and can pass these to the *Modelling Unit*. The *Modelling Unit* uses Multiple-Linear Regression (MLR), [Draper and Smith 1981] to model the relationship between these sets of variables. MLR (equation 1) provides a statistical model of the relationship between a response variable, Y, and two or more explanatory variables, X, by fitting a linear equation to the observed data. The coefficients β are the expected change in the response variable for every 1 unit change in the respective explanatory variable. This model describes how the mean of Y responds when the explanatory variables are changed. Additionally, MLR outputs a p-value for each explanatory variable, indicating how significant changes in that variable are likely to be on Y (the smaller the p-value, the more significant the relationship).

The accuracy and validity of MLR can suffer if the explanatory variables, X, contain multi-collinearity. This occurs when linear relationships exist between individual explanatory variables, so changes to one affect the other. Multi-collinearity does not impact the reliability or predictive power of the overall model, but can affect the reliability of calculations of the effect each explanatory variable has on the response variable, which is what we are concerned with. As an example of this, take a MLR model in a flocking agent, where Y is the agent's speed and the explanatory variables

contain variable *X1*, the number of other agents in the agent's field of vision and *X2*, the number of total agents in a defined radius of the agent. It is intuitive that an increase in *X1* will also likely mean an increase in *X2*, however the agent requires a means of discovering this relationship autonomously before taking appropriate action.

$$\beta^{lasso} = arg\min_{\beta} \sum_{i=1}^{n} (y_i - (\beta^T xi))^2 + \lambda \left\|\beta\right\|_1 \tag{2}$$

Although MLR provides a simple way of monitoring the relationship between many variables, it does not provide a means of choosing what variables should be in the model. Some variables do not have any relationship to one another, for example, the direction of travel of a flocking agent and the agent's age. In this case, the agent could potentially ignore that relationship as it does not offer value. However, without any semantic understanding of this in advance, the agent needs to discover this at runtime.

There are a number of model selection approaches, such as P-value based, Akaike Information Criterion [Akaike 1973], the Bayesian information criterion [Albert and Chib 1997] or Principal Component Analysis (PCA) [Wold et al. 1987]. DETect uses Least Absolute Shrinkage and Selection Operation (LASSO) [Tibshirani 1996], which provides the best trade-off between accuracy, efficiency and complexity. LASSO, equation 2, is a shrinkage and selection method for use with linear regression [Tibshirani 1996] to prevent over-fitting. It returns the set of coefficients, β^{lasso}, that minimises the sum of squared errors while a regularisation penalty, λ, is applied to the absolute value of each of the coefficients, β. This penalty results in all coefficients being shrunk toward zero and some coefficients will be exactly zero meaning they have no affect on the model and can be discarded. As a result, LASSO provides automatic variable selection while also dealing with multicollinearity in the explanatory variables by removing variables that do not add information to the model.

Model Selection: The *Model Selector* component selects models as detailed in Algorithm 1. Model selection begins by initialising a *model map* (line 1), that will be filled with internal-external variable pairs, representing the variable relationships that will form the regression model. Pre-processing steps are applied to the data, which remove variables that are static (contain all the same values) or are just random noise. First, some zero-mean noise is added to each individual observation (line 2) to ensure that no entirely static variable is fed into LASSO. Each individual internal and external variable is checked to see if it is just random noise (line 3). This is done by converting the observations for each variable to a distribution and using Pearson's Chi-square test for randomness [Pearson 1900]. If a variable's variation is deemed to be entirely random, it is excluded from the model as it does not contain any information. Following this, the number of internal and external variables is updated (lines 4-5) before the final pre-processing step centres and scales each variable's data, by subtracting its mean from each observation and dividing by its standard deviation (line 6).

Next, the remaining variables are fed to LASSO to choose the model. LASSO is run in two stages, first for each internal variable against all external variables (lines 8-18), and then for each external variable against all internal variables (lines 19-27). A matrix, $LassoCoefficientsMatrix$, is initialised (line 7) with zeros, where each row corresponds to an internal variable and each column to its relationship with an external variable. The observations for each internal variable and for all external variables are taken (line 9). An array is initialised to store the output of LASSO (line 10). LASSO works by returning the coefficients, β, for all external variables as fitted by the LASSO algorithm (line 11). The coefficients are checked and if a coefficient is non-zero, the corresponding entry in $LassoCoefficientsMatrix$ is updated by adding 1 (lines 14-16).

ALGORITHM 1: Model Selection in DETect

Input: Internal Variables Sliding Windows SWI_i and External Variables Sliding Windows SWE_i of agent A_i. The number of external variables L. The number of internal variables K.
Output: $modelMap$ containing all internal-external variable pairs used in the agents model

```
1  modelMap = new Map(InternalVariable,ExternalVariable)
2  injectNoise(SWI_i, SWE_i)
3  removeRandomVariables(SWI_i, SWE_i)
4  L = count(SWE_i)
5  K = count(SWI_i)
6  centerAndScale (SWI_i, SWE_i)
7  LassoCoefficientsMatrix = 2DArray[K * L]
8  for SWI_i1 to SWI_iK do
9  |  Ym = SWI_il;
10 |  Initialise LassoCoefficient = new Array[L];
11 |  LassoCoefficient = RunLasso(Yk,SWE_i);
12 |  for LassoCoefficient[l] to LassoCoefficient[L] do
13 |  |  if LassoCoefficient[l] != 0 then
14 |  |  |  LassoCoefficientsMatrix[i,l] = LassoCoefficientsMatrix[i,l] + 1
15 |  |  end
16 |  end
17 end
18 for SWE_i1 to SWE_iL do
19 |  Yl = SWE_il;
20 |  LassoCoefficient = RunLasso(Yl,SWI_i);
21 |  for LassoCoefficient[1] to LassoCoefficient[K] do
22 |  |  if LassoCoefficient[k] != 0 then
23 |  |  |  LassoCoefficientsMatrix[k,l] = LassoCoefficientsMatrix[k,l] + 1
24 |  |  end
25 |  end
26 end
27 for each cell in LassoCoefficientsMatrix do
28 |  if cell == 2 then
29 |  |  Add InternalVariable[Row-Number], ExternalVariable [ColumnNumber] to modelMap
30 |  end
31 end
32 if size(modelMap) > 0 then
33 |  return(TRUE)
34 end
35 else
36 |  return(FALSE)
37 end
```

The regularisation term, λ, is selected independently for each Lasso run using cross-validation, meaning it does not have to be set in advance.

In the second stage, this process is repeated for each external variable against all internal variables, with the $LassoCoefficientsMatrix$ once again updated should LASSO return a non-zero coefficient for the corresponding internal-external variable pair (lines 23-25). Finally the coefficients matrix is checked to see what cells contain a non-zero coefficient for the internal-external variable pair for both LASSO runs (lines 28-32). Variable pairs that have these relationships are added to the model map (line 30). Finally, if the model contains at least 1 internal-external variable pair the model selection is deemed successful and a boolean $True$ is returned (line 33-37). It may be the case that no relationships are selected when this algorithm is run. If that is the case, model selection is re-run every time the variable sliding windows are filled until at least one internal-external variable relationship is selected. As a result, an agent's relationship model with its environment in DETect is represented as one or more MLR models, each of which is independently monitored for change as the system runs. This

model is periodically analysed using MLR (Algorithm 5 described in Appendix B). The duration between consecutive regression analyses is referred to as the *Regression Window* and various sizes of this window are tested as part of the case study described in Section 5.1.

4.2. Change Detection Unit

An agent's relationship to the environment can now be monitored over time to detect changes. The agent creates a range of expected metrics for the relationship against which periodic evaluations can be compared, and significant departures from the expected range quickly diagnosed. Expected values are constantly updated to reflect the transient nature of emergence. For example, flocking behaviour in birds both forms and evaporates quickly. Detecting when both occur is necessary to achieve emergence detection. Therefore, before flocking forms, the characteristics of the system without flocking should provide the standard against which an agent judges their current relationship. Once flocking has occurred, the agent compares against the characteristics of the system with flocking, detecting when the emergent behaviour evaporates.

DETect uses a sliding window, the *CUSUM Window*, to achieve this effect. Each internal-external variable pair that forms part of the model has its own *CUSUM Window*. Each window is composed of recent observations of the relationship, taken from the N most recent evaluations (p-values) and is updated after every time-step ($M = 1$). The *CUSUM Window* forms the baseline against which each newly generated p-value is compared to determine if the most recent observation indicates the mean or standard deviation of the p-values are changing. Changes at both emergence formation and evaporation can be detected, assuming the width of the window, N, is less than the duration of the emergent behaviour. The *CUSUM Window* length N is tested in our evaluation with a range of values.

$$S_{n+1} = max(0, S_n + x_n - K) \tag{3}$$

Next, the agent defines what constitutes a significant change. A number of existing change point detection algorithms could identify when the probability distribution of a time series of values changes. The Cumulative Sum (CUSUM) algorithm [Page 1954] is computationally inexpensive, involving sequentially summing time series samples, x, as outlined in equation 3. S_0 is initialised to 0 and K refers to a weighting applied to each sample. When S exceeds a defined threshold, h, a change point in the series has been found. Our choice of values for both K and h is discussed later. Note, the term *CUSUM Window* is introduced by DETect, sliding windows have previously been used in conjunction with the CUSUM algorithm [Montes De Oca et al. 2010; Nikovski and Jain 2009].

The *Change Detector* unit combines sliding windows with CUSUM for detecting changes in the agent/environment relationship (Algorithm 2). It takes as input the model containing the names of the selected internal variables and their associated external variables. It also uses the parameter N for the *CUSUM Window* and K and h for the CUSUM algorithm. Setting the values for both the weighting parameter, K and the CUSUM threshold, h, is important to the performance and sensitivity of the CUSUM. It is typical to set a value of K to half the number of standard deviations from the mean deemed significant. Assuming a normal distribution in the data, 95% of values can be expected to fall within 2 standard deviations of the mean. Using this 5% significance level, $K = 1$ was selected, and used throughout DETect's evaluation. The threshold was set to $h = 4$ after preliminary experimentation.

Lines $1-9$ of Algorithm 2 set up the *CUSUM Window* and cumulative sums, *SMean* and *Sdev* for each internal-external variable pair in the *model map*. *SMean* (line 6)

ALGORITHM 2: Relationship Monitoring and Change Detection

Input: The model being observed, $modelMap$, containing the names of the internal-external variable pairs. New p-values for each internal-external variable pair in the $newP_ijs$ in $modelMap$. The size of the CUSUM sliding window N_{cusum}. h the CUSUM threshold. K CUSUM weighting parameter
Output: A feedback detection event, $feedBackFound$

```
1  modelInternalVars = Internal Variables in modelMap;
2  for each modelInternalVars in MIV_a do
3      modelEnternalVars = modelMap.get(MIV_a);
4      for each modelEnternalVars in MEV_b do
5          initialise empty CusumWindow_ab
6          Smean_ab = 0
7          SDev_ab = 0
8      end
9  end
10 /* Main CUSUM Part */
11 for each Time steps do
12     if newP_ijs arrived then
13         feedBackFound = FALSE
14         for each CusumWindow_ab do
15             newP_ab = get P-value from newP_ijs
16             if size(CusumWindow_ab) == N_cusum then
17                 expectedMean = mean(CusumWindow_ab)
18                 expectedSd = sd(CusumWindow_ab)
19                 [SMean_ab, SDev_ab] = CUSUM(expectedMean,expectedSd,SMean_ab,SDev_ab,K)
20                 if absolute(SMean_ab) > h or absolute(SMean_ab) > h then
21                     feedBackFound = TRUE
22                 end
23                 Delete CusumWindow_ab[0]
24             end
25             CusumWindow_ab.add(newP_ab)
26         end
27         send feedBackFound to Collaboration Unit
28     end
29 end
```

is used to look for deviations from the expected mean of the p-value and *SDev* (line 7) is used to look for deviations from the expected standard deviation as the system evolves. The main CUSUM section begins at line 10. A check is made at every time step to determine if a new set of p-values have been generated by the modelling unit (line 12). If so, the feedback detection event flag is set to false before the analysis takes place (line 13). Next, each *CUSUM Window* monitored is processed, with the specific p-value associated with that variable pair extracted from the received set of p-values (line 15). If the *CUSUM Window* is full, the expected mean and standard deviation are calculated and the CUSUM analysis is executed (lines $17-19$). This analysis returns the updated CUSUM values for both *SMean* and *SDev* and the absolute value of these is compared to the CUSUM threshold h, to determine if a significant change has occurred (line $20-22$). Next, the oldest value in the *CUSUM Window* is deleted (line 23), and the most recent observation of the p-value is added (line 25). Finally, the state of *feedBackFound* is sent to the *Collaboration Unit*, which indicates if a statistical change has occurred in the agent's relationship with its environment (line 27).

4.3. Collaboration Unit

When an agent detects a change, this indicates that some emergence *may* be present in the system. However, an individual agent cannot reason about the existence of emergence due to the limited scope of their view. Therefore, agents need to collaborate to build a sufficiently large view of the system, before consensus about the existence of

an emergent event can be achieved. This goal requires a decentralised communication protocol that allows agents to identify other agents, and share change information. As the existence of emergence is often transient, the time taken in achieving consensus is critically important. This is compounded by the possibility that each agent may detect feedback at slightly different times depending on their unique experience, meaning that some resilience is required. Finally, the dynamic nature of CAS with a changing topology means that who to exchange information with must be considered. Gossip-based communication protocols have widely been used across a variety of networks types, proving efficient in both large scale networks [Jelasity et al. 2005] and mobile ad-hoc networks [Datta et al. 2004]. Consensus formation can be facilitated in such networks [Carli et al. 2010][Lavaei and Murray 2012] without a centralised controller, and so is an ideal candidate to act as the communication protocol between agents.

DETect's *Collaboration Unit* is responsible for inter-agent communication to share information about feedback detection, so agents can learn about the system state beyond their immediate locality. Algorithm 3 outlines a gossiping consensus algorithm, based on distributed averaging, that allows agents to reach consensus on the presence of sufficient feedback detection in the system. When DETect raises a feedback detection event, the variable Lv is set to 1 (lines $2-6$). DETect also maintains a second variable, Ev, which represents the proportion of other agents believed to have detected feedback. EV is updated by randomly selecting one of the agent's neighbours and updating both Ev values to their average (lines $9-13$). After this is done, Ev is scaled towards the agent's own feedback variable Lv, so that without reinforcement from other agents, DETect will believe its own experience (lines $18-20$). DETect concludes the existence of an emergent event when Ev exceeds a certain threshold (lines $24-26$). The value of this threshold is explored as part of the evaluation described in Section 6.

Compromises between speed and accuracy are made in consensus formation between agents (lines $7-13$). An upper and lower limit is used on the neighbourhood size of the agent, so DETect does not gossip unless the agent has a minimum number of neighbours. The nature of p-values means that some random changes may occur in the monitored relationships and cause individual agents to falsely detect change. If an agent is in a sparsely populated area and has few one-hop neighbours, the consensus averaging for this group will be artificially high compared to a densely populated area. However, agent interactions are a requirement for emergence to exist [Wolf and Holvoet 2005], so by not trying to detect emergence when an agent has few neighbours and thus, few interactions, false-positive emergence detections can be reduced. DETect remembers that feedback was detected for a period of time, allowing multiple agents who do not detect feedback at exactly the same time to still form consensus over time. Finally, an agent's neighbourhood is composed of the other agents that it interacts and communicates with. DETect itself is agnostic as to how this communication range is determined and, therefore, neighbourhood could potentially be either spatial or non-spatial depending on model characteristics.

4.4. Discussion

Types of Emergence: The goal of DETect is to detect emergence at runtime without requiring an external or global observer to do so. In this sense, it can be said that DETect attempts to detect *strong* emergence, as defined by Muller [Muller 2004]. As a result, the emergence of interest is that which can be sensed by the agents through downward causation, with the expectation that by the presence of emergence, agents can take steps to mitigate or leverage its effects. DETect is therefore positioned in the gap that exists between Fromm's Type-II (mechanistic) and Type-III (reflective) emergence, where feedback exists from the macro-level to the micro-level [Fromm 2005b].

ALGORITHM 3: DETect Distributed Consensus Forming Step

Input: A feedback detection event, $feedBackFound$. The set N_{it} containing the one-hop neighbours of agent A_i at time t. DETect's previous estimate of feedback consensus in the system Ev. The minimum and maximum neighbourhood sizes permitted, $neighMin$ and $neighMax$. The consensus threshold value that triggers an emergence detection event $consensusTheshold$. How quickly to scale consensus belief to feedback value, $scaler$

Output: An emergence detection event $EmergeDetected$

```
1  EmergeDetected = FALSE;
2  if feedBackFound == TRUE then
3  |  Lv = 1;
4  else
5  |  Lv = 0;
6  end
7  if size(N_it) ≥ neighMin) then
8  |  /* If there are too many neighbours, select the neighMax closest */
9  |  if size(N_it) ≤ neighMax) then
10 |  |  gossipCanidates = N_it;
11 |  else
12 |  |  gossipCanidates = nearest(neighMax, N_it);
13 |  end
14 |  partner = Random(gossipCanidates);
15 |  partnerEv = Ev of partner;
16 |  myEv = (partnerEv + Ev) / 2;
17 |  Ask partner update Ev to myEv;
18 |  EV = myEV;
19 |  diff = EV - Lv;
20 |  EV = EV - diff * scaler);
21 else
22 |  Ev = 0;
23 end
24 if Ev ≥ consensusTheshold then
25 |  EmergeDetected = TRUE;
26 end
```

Both De Haan and Fromm state that systems capable of Type-III emergence are true complex adaptive systems [de Haan 2006].

Domain Experts: It should also be noted that although DETect is designed to automatically select the variables that form its model, this does not mean that input from a domain expert may not aid the model selection process. DETect receives the internal and external variables that are made available to it by the agent. However, it is possible that a domain expert may identify non-useful variables and exclude these from DETect's visibility. In such a scenario, the speed at which the *Modelling Unit* processes the LASSO analysis will be increased, however the validity of the model will remain the same due to LASSO eliminating all relationships that do not contain information.

Scope: Finally, DETect is only concerned with detecting emergence. Taking the MAPE loop [Kephart and Chess 2003] as an example of an adaptation pattern, DETect is located in the monitoring phase, with output from DETect potentially triggering adaptations later in the adaptation workflow. As a result, deciding what to do once emergence is detected is outside the scope of DETect.

5. FORMATIVE STUDY

To evaluate the feasibility of DETect in CAS scenarios, we use three multi-agent models of different scale implemented in Netlogo [Wilensky 1999]. These are a traffic simulation, pedestrian movement [Procházka and Olševičová 2015] and boid flocking [Reynolds 1987], with each model capable of exhibiting emergence. All three models are similar in the sense that emergence is spatial in each system. This was neces-

Table I. DETect feedback detection evaluation factors

Factor	Description	Levels (Pedestrian & Traffic)	Levels (Flocking)
Regression Window	Size of the sliding observation window used for Regression Analysis	LOW - 20 HIGH - 40	LOW - 20 HIGH - 40
CUSUM Window	How large is the sliding window used by the CUSUM algorithm	LOW : 80 MEDIUM : 100 HIGH : 120	LOW : 10 MEDIUM : 20 HIGH : 30

sitated by the extremely limited availability of diverse open source simulation models that exhibit emergence, where the presence of emergence is easily identifiable and, therefore, uncontroversial. We nonetheless believe that our evaluation in the context of such spatial systems is rigorous with multiple models used including flocking, the standard emergence model used across literature [Birdsey and Szabo 2014; Seth 2008; Niazi and Hussain 2011; Chan 2011; Szabo and Teo 2015]. Additionally, the pedestrian model [Procházka and Olševičová 2015] is one where the behaviour and emergence were not written and defined by us. This is discussed further in Section 6.7 where we outline the future work motivated by this research.

In each simulation model, all agents are autonomous and each has its own unique implementation of the DETect algorithm. Due to space restrictions, the traffic case study is described in full, with references made to the pedestrian and flocking models when the results obtained differ from the traffic case study. Additionally, the case study presented focuses on both the Change Detection unit (feedback detection) and the Collaboration Unit (consensus formation) which are the primary components involved in emergence detection, the primary purpose of the study. An evaluation of the model selection algorithm is included in Appendix C.

Finally, a number of parameters used by DETect are presented in the study below, with a limited range of values explored. Finding optimal values for each of these parameters is not the intention of this study which instead focuses on evaluating DETects feasibility as an emergence detector. Therefore, sensible values for each parameter, that facilitate this goal, are sufficient for the purpose of this study. The full list of values are presented in Table I and Table II, and are representative of values for their respective domains. These acceptable ranges were identified through iterative experimentation prior to the study with initial values chosen based on the goal of each affected component, using some knowledge of the domain. In addition, taking examples from Table I, we learned through experimentation that the *Cusum Window* was highly sensitive for Flocking and less so for Pedestrian & Traffic. The values chosen reflect the change in sensitivity, induced by the domain.

5.1. Experimental Set-up

Feedback Detection: The first stage of the case study focusses on feedback detection, using a multi-level factor experiment design. The factors and their levels are described in Table I. The *Regression Window* size is set to one of two levels, and the *CUSUM Window* size set to one of three levels. The *CUSUM Window* sizes are consistent for both the pedestrian and traffic model, however different sizes are used for the flocking model to achieve useful performance. This is discussed in more detail in Section 6. Each combination of factor level was replicated 10 times for each simulation model, with both factorial analysis of variance (ANOVA) and a two-tailed t-test performed to identify the important factors and the statistical significance of the performance.

Emergence Detection: DETect's emergence detection events should coincide with emergent events occurring in the system. Emergent events in, for example, the traffic model are the formation and evaporation of congestion. For the evaluation, emergence

Table II. DETect consensus formation evaluation factors

Factor	Description	Levels
Minimum Neighbourhood Size	What is the minimum number of neighbours needed before DETect will begin gossiping	LOW : 8 HIGH: 16
Maximum Neighbourhood Size	What is the maximum number of neighbours DETect will consider as potential gossiping partners	LOW: 20 HIGH: 30
Feedback Detection Memory	How long will DETect remember a recent feedback detection event	LOW: 5 HIGH: 10
Consensus Threshold	How large is Ev required to be before an emergence detection event is generated	LOW: 0.25 HIGH: 0.40

Fig. 2. Screenshot of traffic simulation environment showing street network (white lines), car agents (yellow), post-code borders (blue lines) and depots (red dots)

has to both form and evaporate during simulations runs and these transition periods must be objectively recognisable. To facilitate this, each simulation is periodically switched between emergence and non-emergence phases allowing emergent behaviour to both form and disperse. This switching allows change periods to be identified at the threshold of each phase where DETect should both detect feedback and subsequently build consensus on the presence of emergence in the system. Feedback detection and consensus forming are examined separately during this study, so important factors for each task can be evaluated separately. The number of feedback detection events and emergence detection events that occur during these identified *change periods* are compared with the frequency of such events outside these periods.

The performance of emergence detection is evaluated using a multi-level factor experiment design. The factors and levels used are outlined in Table II, with each factor set to one of two levels. Each combination of factor level was replicated 5 times, giving a total of 80 replications. The results of the second stage are used to provide suitable levels for both the *Regression Window* and *CUSUM Window*. Factorial ANOVA and a two-tailed t-test is performed to identify the important factors and to evaluate the number of emergence detection events generated by agents during *change periods* compared to non-change periods. Results are considered significant when the t-test returns values of $p < 0.05$.

5.2. Traffic Simulation

The traffic case study uses OpenStreetMaps to model the Manhattan road network [BBBike.org 2014]. Agents are taxis implemented to traverse a model of the street network with GraphHopper [Graphhopper 2014]. GraphHopper is an open source routing library, based on OpenStreetMap data, used to model realistic routing. A screen shot of the model is in Figure 2 with the agents visible as yellow on the map. The blue boxes represent zip-code areas, each containing a taxi depot where the agents begin the simulation. Agents are initially evenly distributed across the map and traverse the map with the goal of travelling to all depots during the simulation.

The simulation uses 2500 taxi agents, each completing its journey while avoiding collisions with other agents, and stopping for traffic lights at intersections. Agents travel from depot to depot with their first depot selected randomly by each agent. Periodically, all agents are directed to the same depot as their next destination, forcing all agents to converge to the same area. This convergence leads to the formation of traffic congestion, which is the emergent behaviour we wish to detect.

Each agent monitors 3 internal and 5 external variables, (Table III), giving a total of 15 internal-external variable relationships to choose from. One internal (Age) and one

Table III. Traffic Model Internal and External Variables

Name	Description
Internal Variables	
Heading	The current heading in degrees of the agent
Speed	The current speed of the agent
Age	A static variable randomly selected at agent initialisation
External Variables	
Neighbours Heading	The mean heading agents in the agent's field of vision
Neighbours Speed	The mean speed agents in the agent's field of vision
Spacing	The distance to the closest other agent
Number of Neighbours	The number of agents in the agent's field of vision
Temperature	A randomly varying environment variable

external variable (temperature) are static or vary randomly, and are not used by the agent when taking an action. These variables are not potentially useful and therefore they are inserted as a test of the *Modelling Unit* during model selection.

Flow-rate, a weighted average of the number of agents passing through a street section and their average speed, is used to objectively measure congestion on each street section. As congestion levels rise and more cars join a traffic-jam, the number of congested streets rises. This number is monitored globally to indicate when traffic congestion is growing or shrinking. If the number of streets with congestion rises to a high level and remains constant, gridlock has occurred and a traffic easing measure is employed. This global policy is applied at traffic lights on congested streets where waiting agents are prompted to take an alternative route. Agents can then move again, easing congestion levels, and enabling evaluation of emergence dispersion.

6. EVALUATION

The number of events (feedback or emergence detection) generated across all agents is recorded every 50 time steps during each simulation run. The objective measure of emergence is also monitored at the same frequency. A low-pass butterworth filter [Parks and Burrus 1987] is applied to the objective measure time series following each simulation, with first order difference used to identify periods of change. This allows us to identify when emergence formed and evaporated and therefore identify when DETect should detect both feedback and emergence. The average number of events generated every 50 time steps during formation, evaporation and no-change periods are then measured. Each simulation has an initialisation period for model selection, followed by a seeding period where agents compile sufficient samples to fill the *CUSUM Window*. DETect can then look for changes in the agent's environment that may signify feedback from emergence. The length of this period is a factor of the length of both the *Regression Window* and *CUSUM Window* and so differs across runs. As a result, this period has been excluded from our evaluation of each simulation run.

6.1. Evaluation Objectives

The evaluation criteria presented in this section are structured so that the performance of DETect in model selection, feedback detection and emergence detection can be individually evaluated, with the combination of parameters that provide the best performance for feedback and emergence detection in each simulation model explored.

Feedback Detection: To evaluate the performance of feedback detection in DETect, we are concerned with determining its effectiveness and the combination of factors that result in the best performance. Feedback detection is effective if, for some combination of factors, DETect satisfies the following requirements:

— Obj #1: The number of feedback events generated by DETect across all agents is statistically significantly higher during periods of no emergence.

— Obj #2: The number of feedback events generated by DETect across all agents is statistically significantly higher during periods of no emergence.

Emergence Detection: Emergence detection is evaluated by observing the number of emergence detection events generated by the *Collaboration Unit* across all agents throughout the simulation. As this involves all components, the overall effectiveness of the DETect algorithm is evaluated. In particular, we measure how effectively DETect performs when detecting emergence, exploring the combination of factors for consensus formation that best facilitates this. DETect effectively detects emergence if the following requirements are satisfied for some combination of factors:

— Obj #3: The number of emergence detection events generated by DETect across all agents is statistically significantly higher during periods of emergence formation compared to periods of no emergence.
— Obj #4: The number of emergence detection events generated by DETect across all agents is statistically significantly higher during periods of emergence evaporation compared to periods of no emergence.
— Obj #5: Emergence detection events are generated in a timely manner so an interested party can take appropriate action.

6.2. Feedback Detection

The average number of feedback detection events generated during simulations of both the traffic model and flocking model for each factor combination is shown in Figure 3. Events are counted every 50 time steps with the average number of events generated during non-change (red), emergence formation (green) and evaporation illustrated. Similar results were observed for both the traffic and the pedestrian models with a significantly higher number of events generated during both formation and evaporation periods compared to non-change periods. The size of the *Regression Window* significantly affected the sensitivity of the algorithm, while changes in the *CUSUM Window* size had a smaller impact. In the traffic model, fewer than 20 feedback detection events were generated during any period when the *Regression Window* size was set to the lower level. Conversely, at the higher level, the average exceeded 100 events for all periods, and at the higher level of the CUSUM window, exceeded 200 events for both emergence formation and evaporation. As the model is composed of 2500 agents, an average of less than 20 events during transition periods is too low and we therefore deem the low *Regression Window* to be too insensitive.

The algorithms performed acceptably at the higher regression window size with a significantly higher number of events generated during formation and evaporation, compared to non-change periods. The difference was not as stark in the Pedestrian model, where a large number of false positives are generated, but it is still sufficiently large. For all models, the combination of high *Regression Window* and high *CUSUM Window* performs best, where the average number of events generated during formation periods was over twice that of non-change periods. Evaporation periods displayed a similar high number of average event regularity at this factor level. As a result, the High level for both the *Regression Window* and *CUSUM Window* is used during the consensus formation study. The selection of suitable values for each of these parameters is discussed further in Section 6.4.

In the Flocking model, no combination of factors achieves a higher number of feedback detection events during non-change periods compared to both formation and evaporation periods. As with the other two models, the *Regression Window* size was a significant factor in the performance. The sensitivity of the algorithm at the high level of *Regression Window* size was significantly higher for each period type across all three levels of the *CUSUM Window* size. For example, the average number of feedback

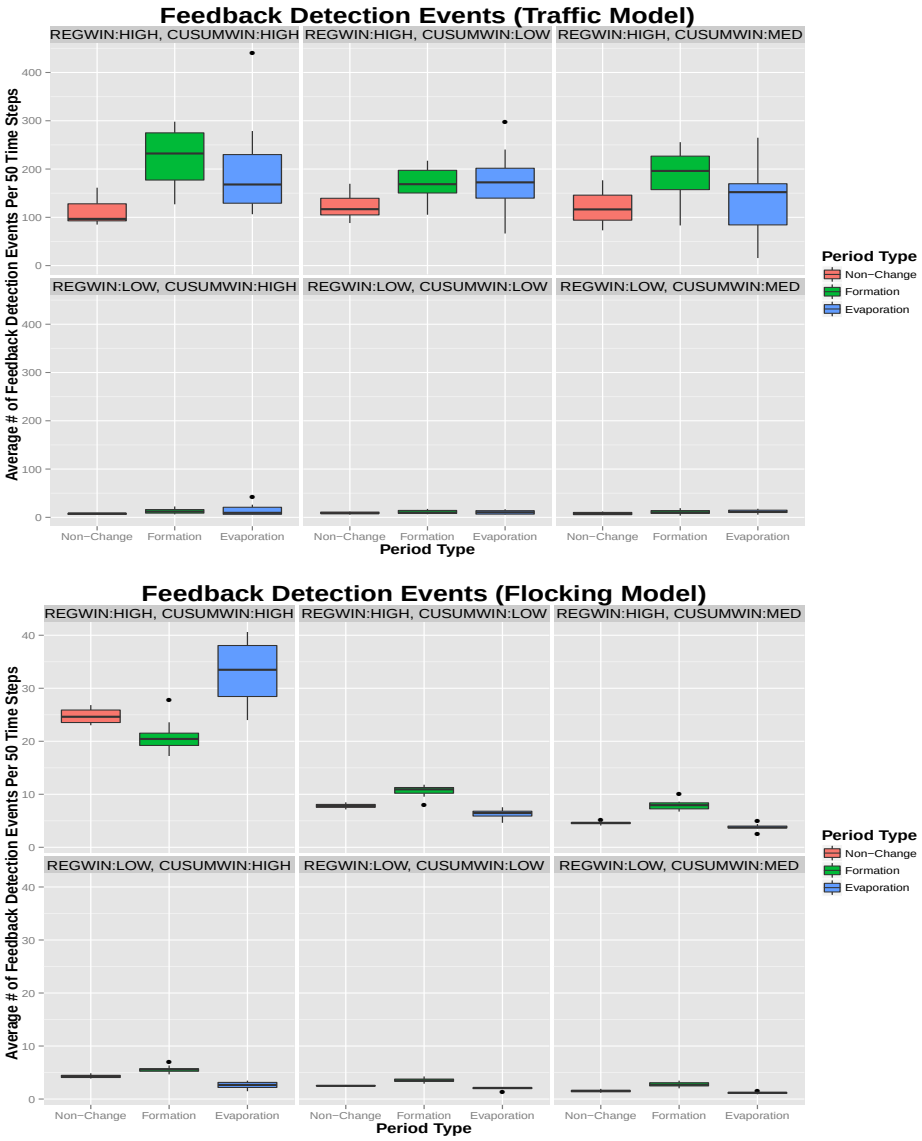

Fig. 3. Feedback detection results for the Traffic (top) and Flocking Model (bottom) for each combination of the CUSUM Window (CUSUMWIN) and Regression Window (REGWIN). The average number of detection events are divided into non-change (red), emergence formation (green) and evaporation (blue) periods.

detection events during formation periods for the medium CUSUM level increases from 3.52 with the low *Regression Window* size, to 10.62 with the larger window. Variation in the *CUSUM Window* size also demonstrated an increasing sensitivity as the window size was increased, but the magnitude was not as significant as with the *Regression Window*. An exception to this occurred when both factors were at the higher level with the feedback detection demonstrating over-sensitive behaviour. This combination exhibited more feedback detection events occurring during evaporation periods than non-change periods, but the mean number of feedback events during non-change

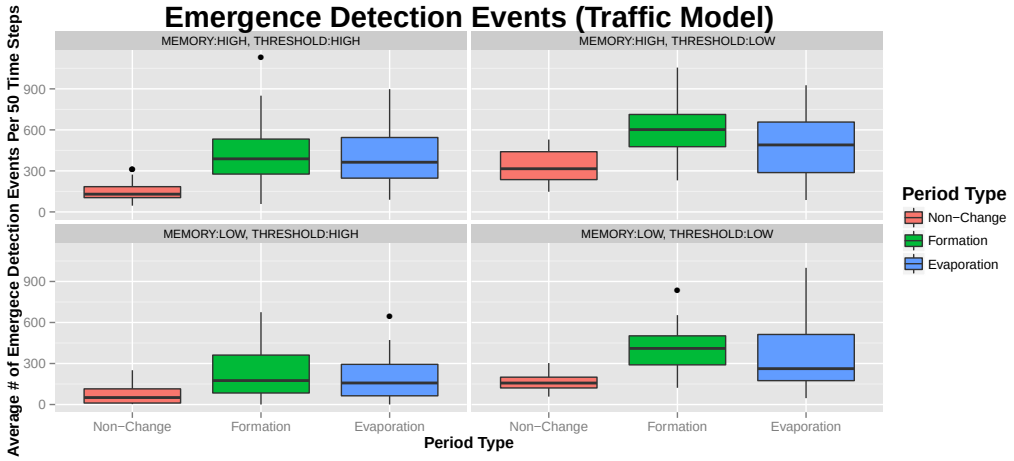

Fig. 4. Consensus formation and emergence detection results for the Traffic Model for each combination of consensus threshold and feedback detection memory length factors

periods was 24.73, meaning 16.5% of agents were experiencing a false positive changing relationship at any moment. This proportion of agents is too high and considering also the poor performance during formation periods, we excluded this combination of factors.

In all other instances formation periods experienced higher detection events compared to non-change periods, but evaporation periods did not. So, we had to compromise for the flocking model and select a factor combination that achieves a higher average number of feedback detection events during periods of formation. Simultaneously, we need to generate close to the same number of events during evaporation periods as non-change periods. As a result, the combination of high regression window size and medium CUSUM window size are used in the next stage.

6.3. Emergence Detection

The *Collaboration Unit* involves 4 factors, Consensus Threshold, Feedback Detection Memory length, Neighbourhood Size Minimum and Neighbourhood Size Maximum, each with two levels. *Consensus threshold* and *Memory length* have the largest impact on performance, and so we focus on those here. As with feedback detection, the results in both the traffic and pedestrian models were similar, with a statistically significantly higher number of emergence detection events generated during formation and evaporation compared to non-change periods, for each combination of factors. In the flocking model, the legacy of feedback detection during evaporation periods impacted results. Although satisfactory performance was achieved during emergence formation periods, emergence detection events were not statistically significantly higher during evaporation periods compared to non-change periods.

The results for emergence detection in the Traffic model for each combination of consensus threshold and memory length are illustrated in Figure 4. *Consensus threshold* had a large impact on the number of detection events, with the number more than doubling during non-change periods at the lower level. This lower level has a similar impact during both formation and evaporation periods at the lower level of memory length. However, when the *Memory Length* is set to the higher level the increased number of false positive during non-change periods does not see a comparative increase in detection rates during formation and evaporation. A High level for both

Fig. 5. Timeliness of emergence detection events in relation to the start and end of each transition period (dashed lines) for both the Traffic and Pedestrian model

Consensus Threshold and *Memory Length* achieved the best results. DETect generated a statistically significantly higher number of emergence detection events during emergence formation, 424.19, and evaporation 407.53, compared to non-change periods, 152.99, yielding the largest margin between the period types.

Figure 5 shows the timeliness of emergence detection events for both the Traffic and Pedestrian models at the best factor combination, as the average number of events detected relative to the start of formation and evaporation transition periods. The vertical dashed line at time step 0 is the start of the transition period (when emergence formation or evaporation begins) and the second vertical line represents the average

Table IV. Summary of results againsts evaluation objectives

Obj.	Description	Flocking Model	Pedestrian Model	Traffic Model
#1	The number of feedback events generated by DETect across all agents is statistically significantly higher during periods of emergence formation compared to periods of no emergence.	Satisfied	Satisfied	Satisfied
#2	The number of feedback events generated by DETect across all agents is statistically significantly higher during periods of emergence evaporation compared to periods of no emergence.	Not Satisfied	Satisfied	Satisfied
#3	The number of emergence detection events generated by DETect across all agents is statistically significantly higher during periods of emergence formation compared to periods of no emergence.	Satisfied	Satisfied	Satisfied
#4	The number of emergence detection events generated by DETect across all agents is statistically significantly higher during periods of emergence evaporation compared periods of no emergence.	Not Satisfied	Satisfied	Satisfied
#5	Emergence detection events should be generated in a timely manner, to allow an interested party receiving the events to take appropriate action.	Partially Satisfied	Satisfied	Partially Satisfied

length of such periods across all runs. The average number of both feedback and emergence detection events generated every interval (50 time steps) are plotted in relation to the start of each transition period.

The timeliness of emergence detection in the traffic model does not match that achieved in the Pedestrian model, where almost all detection events occurred during transition periods. In Traffic, the average formation period lasts longer, 76 time intervals, resulting in a gradual increase in the number of both feedback and emergence detection events. This results in a maximum of 504 simultaneous emergence detection events, 20% of all agents, occurring 82% into the average formation period. The rate that events are generated remains high and results in a significant number occurring after the transition period has ended. As a result, these events are counted as having occurred during a non-change period. By comparison, the average evaporation period is shorter, lasting for 55 time intervals. A maximum of 450 simultaneous emergence detection events, 18%, occur 45% into these periods on average. This allows a larger proportion of the transition period to react to the changing emergent state however, as with the formation periods, the number of events remains high throughout the period with some occurring after the transition phase is deemed to have ended. The relative high number of emergence detection events that are generated at the start of this plot (time interval -160 to -100) are the result of those events that were prompted by the formation periods but did not occur until after the formation period ended.

6.4. Discussion

From these results, we conclude (as summarized in Table IV) that DETect is, in general, an effective algorithm for detecting emergence formation across each model.

Feedback Detection: Objectives #1:2 concern feedback detection and are satisfied in the Traffic and Pedestrian models, with a combination of parameters identified that achieved higher detection rates during transition periods. For both models, DETect's feedback detection performed better with a combination of a higher *Regression Window* and higher *CUSUM Window*. These factors represent the memory of each agent, and how much time should be used to judge recent observations against. In the Flocking

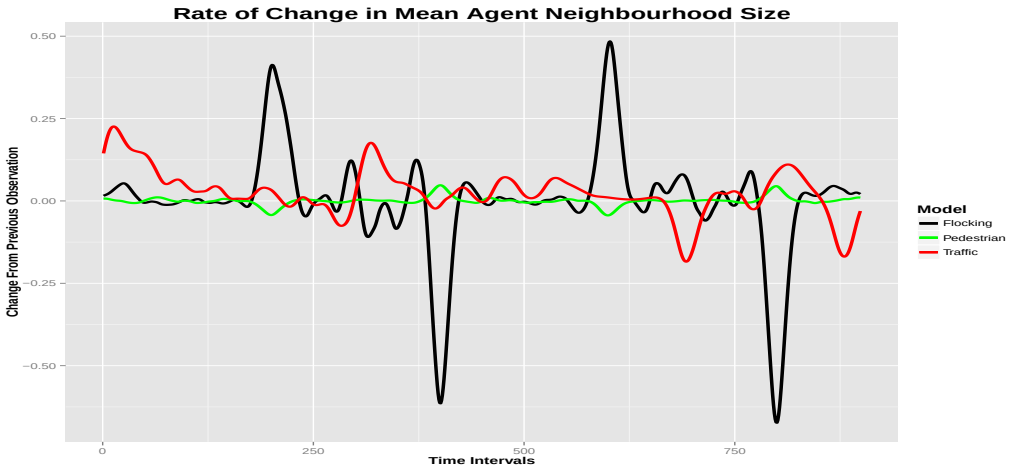

Fig. 6. Rate of change in the neighbourhood size of agents during runs in each model

model, feedback detection was effective during formation periods, satisfying objective #1, but not during evaporation periods, failing to satisfy objective #2.

DETect's limitation during emergence evaporation in the Flocking model means additional work is required to improve performance here. This is in addition to the current need for a significantly smaller series of *CUSUM Window* sizes when using the Flocking model, 10-30, compared to the other two, where consistent levels were applied across all factors, 80-120. We attempted to find a range of CUSUM window sizes that would perform across all models. However, the sensitivity of feedback detection was significantly impacted when the CUSUM window size was increased in the flocking model, with a reduction in window size creating the same effect in the Pedestrian and Traffic model. No common range could be found and our study focussed on the ranges that resulted in useful performance for each model. We believe the reason for this discrepancy is due to the nature of the emergent phenomena in each model. For example, movement of boid agents in the flocking model appears to be more dynamic compared to the traffic and pedestrian models, where collisions are forbidden, resulting in a smaller memory of the past being more beneficial.

We characterise this dynamicity, with a comparison of the rate of change in the mean neighbourhood size of agents throughout a simulation of each model, as illustrated in Figure 6. Neighbourhood size is defined as the number of one-hop neighbours each agent has. The positive spikes correspond to periods of emergence formation and the negative spikes to periods of evaporation. Flocking demonstrates significantly larger change rates during these periods compared to the other models and a more dynamic environment when emergence is present compared to periods without emergence. Significant further experimentation is required to fully understand this variation across a large number of different systems. Our future work will focus on this to provide heuristics for deciding *CUSUM* and *Regression* window sizes.

Emergence Detection: Objectives #3:5 concern emergence detection and the overall performance of the DETect algorithm. The factor combinations that provided the best performance differed across each of the models. For the Pedestrian and Traffic models, a common combination existed that provided significantly higher detection events during periods of emergence and evaporation compared to non-change periods. So, objectives #3:4 are satisfied for both models. In flocking, consensus formation per-

formed well during periods of emergence formation satisfying evaluation objective #3, but were not sufficiently high during periods of emergence to satisfy evaluation objective #2. This limitation impacts the performance of consensus formation. So, objective #4 was not satisfied for the Flocking model.

The timing of feedback and emergence detection events across all models coincided with the start of formation and evaporation periods. In the Pedestrian model, these periods saw a significant spike in generated events with over 60% of agents simultaneously generating detection events during transition periods. This increased event rate could indicate to an adaptation manager that an emergent event is occurring, allowing appropriate steps to be taken in response, satisfying objective #5. Similar performance was achieved during formation periods in flocking but as evaporation periods were not effectively detected, objective #5 is only partially satisfied in the flocking model.

In comparison, the rate of detection events in the Traffic model increased gradually, with a significant number occurring after the end of the transition period. The effect of this "lateness" is that although these detection events are prompted by the transition period, they are classified in our study as false-positives, as they occurred during non-change periods. As a result, DETect's accuracy in the Traffic model is perhaps somewhat pessimistic. Objective #5 is partially satisfied for the Traffic model, with additional work required to improve DETect's sensitivity during these transition periods, so that detection is sufficiently timely to allow appropriate adaptation to be undertaken before the emergent system state becomes established.

6.5. Scalability of DETect

In larger systems with hundreds of thousands of agents or agents with thousands of variables, scalability of DETect will prove to be important. Although this is not explicitly evaluated in this paper, an analysis of the algorithm indicates that DETect can scale gracefully as the number of agents in the system increases. The reason for this is that the DETect component that is sensitive to the number of agents, the *Collaboration Unit*, uses Algorithm 3, with a complexity of O(n). DETect does not require that consensus is formed across all agents in the system. Instead each agent periodically approximates the number of proportion of agents that are experiencing feedback from emergence. This process involves selecting one of its one-hop neighbours to gossip with and exchanging information.

In contrast, Algorithms 1 and 2 relate to the complexity of agents in terms of the number of internal and external variables that are tracked. For model selection, each internal variable is compared against all external variables before the process is repeated for external variables against internal variables. As a result, Algorithm 1 has a complexity of O(n^2). DETect's *Change Detection Unit* tracks the number of relationships between internal and external variable pairs that are selected by model selection. In a worst case, model selection may not eliminate any relationships meaning that the *Change Detection Unit* must monitor M external variables for N internal variables. We therefore conclude that Algorithm 2 also has a worst case complexity of O(n^2). The model selection algorithm is therefore fundamental to the scalability of DETect as the number of internal and external variables increase. Appendix 7 presents an evaluation of DETect's model selection algorithm demonstrating its ability to significantly reduce the number of relationships that are monitored across all 3 models.

6.6. DETect versus existing approaches

Table V summarises the contributions made by DETect in the context of the requirements for emergence detection in CAS, outlined in Section 2.2, compared to existing emergence detection methods. This does not mean that existing approaches are without merit, however it reflects the benefits offered by DETect which approaches emer-

Table V. DETect contribution compared with existing techniques

	Occur at Runtime	Decentralised across agents	Rely on local information	Autonomic Selection Of Data	Collaboration between agents	Detect Formation & Evaporation
Variable-Based						
Seth 2008	✓		✓			
Niazi and Hussain 2011	✓					✓
Chan 2011	✓					✓
Birdsey & Szabo 2014	✓					
Procházka and Olševičová 2015	✓					
Formal						
Moshipour et. al 2012						
Teo et. al 2013						
De Angelis & Di Marzo Serugendo 2015	✓	✓	✓		✓	
Szabo & Teo 2015						
Event Based						
Chen et. al 2008						
Lewis & Whitehead 2008	✓					
DETect	✓	✓	✓	✓	✓	✓

gence detection from the agent level up, in contrast to the almost exclusively top-down approach of existing techniques.

The primary contribution is that DETect is a mechanism for decentralised detection of emergence by the constituent agents of the system. This property reflects the decentralised nature of the systems that generate emergence and is in contrast to existing detection techniques that depend on centralised or hierarchical architectures to varying degrees. To the best of our knowledge, DETect's decentralised feedback-based approach is unique in the field of emergence detection.

A second contribution is to reduce the need for design time knowledge of the specific type of emergence that is expected in the system. This is in contrast to existing detection approaches with variable-based approaches requiring advance knowledge of the system-wide variables that describe emergence [Seth 2008; Niazi and Hussain 2011], and formal language and event-based approaches requiring the expected emergent behaviour and properties to be described [Teo et al. 2013; De Angelis and Di Marzo Serugendo 2015]. As a result, the requirement for specific design time knowledge of the emergent behaviour is removed. However, design time input is not completely eliminated as DETect uses a number of parameters throughout its work flow that affect performance and sensitivity. This is discussed further in Section 6.7 where the future work motivated by DETect is described.

Finally, DETect addresses the transient nature of emergence, which forms and evaporates as the system evolves over time. This property of emergence and the timeliness of detection are not considered by existing approaches which evaluate a static snapshot of the system's evolution to determine if it contains emergence. Possible exceptions to this are Niazi and Hussain [Niazi and Hussain 2011] and Chan [Chan 2011] whose approaches, although not explicitly designed to address this issue, could be adapted to achieve this affect. However, both of these approaches, depend on a centralised architecture and need system-wide properties, which limits their scalability and timeliness. In DETect, the need to gather system-wide properties is removed. This facilitates detection of both the formation and evaporation of emergence in the system, improving the timeliness of detection of the new emergent system state.

6.7. Future Work

The evaluation described in this section demonstrates both the feasibility of DETect and the advantages it offers over existing techniques. However, additional research may yield further improvements in a number of areas. First, all three models used involved agents that moved spatially, so similar variables are used to describe both the agents and their environments in each model and the emergence is spatial in nature in all models. Our claims regarding the effectiveness of DETect are therefore limited to such spatial models and there may exist other types of models with different emergence characteristics. Determining the general applicability of our approach requires that future work examine DETect across a range of different systems which include systems that are open, ultra-large in scale, contain heterogeneous agents and different forms of emergence.

On a related note, DETect is composed of three functional units each with a set of parameters e.g., window sizes, thresholds etc., that need to be set appropriately. The evaluation presented in this paper identified an acceptable level for a number of these parameters in the context of the simulation models included in the case study. We believe that these values are suitable starting points for users wishing to deploy DETect in similar domains. However, automatically setting these parameters for an arbitrary domain or finding optimal values for each remains an open research question. This is particularly important in cases where there is little detailed knowledge of either the target domain or potential emergent behaviours that could arise.

The agents included in our experimentation adapt their behaviour relative to their environment. Evolving new functional behaviour in response to emergence would be ideal, but that is not the focus of this paper. Although agents themselves may or may not be adaptive, DETect fits into the monitoring stage of a MAPE loop, so planning and executing comprehensive behavioural adaptations is beyond the scope of this research. Nevertheless, we intend to use DETect in other work on adaptive systems in the research team.. In particular, we wish to establish the effect such adaptations may have on the validity of the model selected by DETect and if a re-initialisation is required post-adaptation.

Finally, DETect is designed to operate in CAS, detecting the formation and evaporation of emergence at run time. The simulation models used in our case study use discrete time, with each agent's clock assumed to be synchronised. Agents periodically execute tasks such as observing the environment, detecting change using CUSUM, finding a partner and gossiping etc., concurrently during a single discrete time unit. In practise this assumption may limit DETect's effectiveness if deployed to a real-time system where agents have different computational power and DETect's components takes longer than a single unit of discrete time (whatever this may be) to complete these tasks. Of particular significance is the importance of agents simultaneously detecting feedback from emergence that is the core of DETect's approach. The parameter, *Feedback Detection Memory*, currently facilitates this by allowing agents to remember feedback detection for a certain period of time. However, determining whether this is sufficient for real-time systems or whether more sophisticated solutions such as parallelization are required motivates additional research in this area.

7. SUMMARY

This paper addresses emergence detection in Complex Adaptive Systems. We identified a set of requirements necessary to achieve emergence detection and presented the DETect algorithm, a novel distributed algorithm that enables constituent agents in CAS to collaboratively detect emergent events. DETect relies on the feedback that occurs naturally from the system level (macro) to the component level (micro) when emergent

behaviour or properties appears, facilitating detection using only information local to each agent. DETect is evaluated using three multi-agent simulation models, establishing its suitability for emergence detection in CAS. Our future work will concentrate on improving the generalisability of DETect and evaluating its performance in diverse simulation environments where emergence is not generated by agent movement.

REFERENCES

H Akaike. 1973. Information theory and an extension of the maximum likelihood principle. In *Second international symposium on information theory*, B. N. Petrov and F. Csaki (Eds.). Budapest: Academiai Kiado, 267 – 281.

Jim Albert and Siddhartha Chib. 1997. Bayesian Tests and Model Diagnostics in Conditionally Independent Hierarchical Models. *J. Amer. Statist. Assoc.* 92, 439 (Sept. 1997), 916–925.

Yaneer Bar-Yam. 2004. A mathematical theory of strong emergence using multiscale variety. *Complexity* 9, 6 (July 2004), 15–24.

BBBike.org. 2014. OSM extracts for New York. (2014). http://download.bbbike.org/osm/bbbike/NewYork/

Mark Bedau. 2002. Downward causation and the autonomy of weak emergence. *Principia: an international journal of epistemology* 6, 1 (2002), 5–50.

Lachlan Birdsey and Claudia Szabo. 2014. An architecture for identifying emergent behavior in multi-agent systems. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1455–1456.

Ruggero Carli, Fabio Fagnani, Paolo Frasca, and Sandro Zampieri. 2010. Gossip consensus algorithms via quantized communication. *Automatica* 46, 1 (Jan. 2010), 70–80.

David J Chalmers. 2006. Strong and Weak Emergence. *The reemergence of emergence: The Emergentist hypothesis from science to religion* (2006), 244–256.

Wai Kin Victor Chan. 2011. Interaction metric of emergent behaviors in agent-based simulation. *Proceedings of the 2011 Winter Simulation Conference (WSC)* (Dec. 2011), 357–368.

Chih-Chun Chen, Sylvia B Nagl, and Christopher D Clack. 2008. A method for validating and discovering associations between multi-level emergent behaviours in agent-based simulations. In *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 1–10.

Anwitaman Datta, Silvia Quarteroni, and Karl Aberer. 2004. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. In *Semantics of a Networked World. Semantics for Grid Databases*. Springer, 126–143.

Francesco Luca De Angelis and Giovanna Di Marzo Serugendo. 2015. A logic language for run time assessment of spatial properties in self-organizing systems. In *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE, 86–91.

J. de Haan. 2006. How emergence arises. *Ecological Complexity* 3, 4 (Dec. 2006), 293–301.

T. De Wolf, G. Samaey, T. Holvoet, and D. Roose. 2005. Decentralised Autonomic Computing: Analysing Self-Organising Emergent Behaviour using Advanced Numerical Methods. In *Second International Conference on Autonomic Computing (ICAC'05)*. IEEE, 52–63.

Jean-Louis Dessalles and Denis Phan. 2001. Emergence in multi-agent systems : cognitive hierarchy , detection , and complexity reduction part I : methodological issues. In *Artificial Economics*. Springer, 147—-159.

G Di Marzo Serugendo, M-PG Irit, and Anthony Karageorgos. 2006. Self-organisation and emergence in MAS: An overview. *Informatica* 30, 1 (2006).

Norman Richard Draper and Harry Smith. 1981. Applied regression analysis 2nd ed. (1981).

Dominik Fisch, Martin Janicke, Bernhard Sick, and Christian Muller-Schloer. 2010. Quantitative Emergence – A Refined Approach Based on Divergence Measures. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 94–103.

Gary William Flake. 1998. *The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation*. MIT press.

Jochen Fromm. 2005a. Ten Questions about Emergence. (Sept. 2005), 13.

Jochen Fromm. 2005b. Types and forms of emergence. *arXiv preprint nlin/0506028* (2005).

Graphhopper. 2014. GraphHopper. (2014). https://graphhopper.com

R Grossman, Michael Sabala, Matt Handley, and Lee Wilkinson. 2009. Discovering Emergent Behavior From Network Packet Data : Lessons from the Angle Project. *Next Generation of Data Mining* (2009), 243—-260.

John H Holland. 1992. Complex adaptive systems. *Daedalus* (1992), 17–30.

Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems* 23, 3 (Aug. 2005), 219–252.

J.O. Kephart and D.M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (Jan. 2003), 41–50.

Ales Kubík. 2003. Toward a formalization of emergence. *Artificial life* 9, 1 (Jan. 2003), 41–65.

J. Lavaei and R. M. Murray. 2012. Quantized Consensus by Means of Gossip Algorithm. *IEEE Trans. Automat. Control* 57, 1 (Jan. 2012), 19–32.

Chris Lewis and Jim Whitehead. 2011. Repairing Games at Runtime or, How We Learned to Stop Worrying and Love Emergence. *IEEE Software* 28, 5 (Sept. 2011), 53–59.

Timothy T Maxwell, Atila Ertas, and MM Tanik. 2002. Harnessing complexity in design. *Journal of Integrated Design and Process Science* 6, 3 (2002), 63–74.

Saurabh Mittal. 2013. Emergence in stigmergic and complex adaptive systems: A formal discrete event systems perspective. *Cognitive Systems Research* 21 (2013), 22–39.

M. Mnif and C. Muller-Schloer. 2006. Quantitative Emergence. In *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*. IEEE, 78–84.

Jeffrey C. Mogul. 2006. Emergent (mis)behavior vs. complex software systems. *ACM SIGOPS Operating Systems Review* 40, 4 (Oct. 2006), 293.

Veronica Montes De Oca, Daniel R. Jeske, Qi Zhang, Carlos Rendon, and Mazda Marvasti. 2010. A cusum change-point detection algorithm for non-stationary sequences with application to data network surveillance. *Journal of Systems and Software* 83, 7 (jul 2010), 1288–1297. DOI:http://dx.doi.org/10.1016/j.jss.2010.02.006

Mohammad Moshirpour, Abdolmajid Mousavi, and Behrouz H. Far. 2012. Detecting Emergent Behavior in Distributed Systems Using Scenario-Based Specifications. *International Journal of Software Engineering and Knowledge Engineering* 22, 06 (Sept. 2012), 729–746.

Jean-Pierre Muller. 2004. Emergence of collective behaviour and problem solving. In *Engineering Societies in the Agents World IV*. Springer, 1—-20.

Vivek Nallur, Julien Monteil, Tyler Sammons, Melanie Bouroche, and Siobhan Clarke. 2015. Increasing Information in Socio-Technical MAS Considered Contentious.. In *Ninth IEEE Self-Adaptive and Self Organizing Workshops (SASOW)*.

Muaz A. Niazi and Amir Hussain. 2011. Sensing Emergence in Complex Systems. *IEEE Sensors Journal* 11, 10 (Oct. 2011), 2479–2480.

Daniel Nikovski and Ankur Jain. 2009. Fast adaptive algorithms for abrupt change detection. *Machine Learning* 79, 3 (jul 2009), 283–306. DOI:http://dx.doi.org/10.1007/s10994-009-5122-x

Linda Northrop, Peter Feiler, Richard P Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, Douglas Schmidt, Kevin Sullivan, and Kurt Wallnau. 2006. *Ultra-Large-Scale Systems - The Software Challenge of the Future*. Technical Report. Software Engineering Institute, Carnegie Mellon. 134 pages.

Eamonn O'Toole, Vivek Nallur, and Siobhan Clarke. 2014. Towards Decentralised Detection of Emergence in Complex Adaptive Systems. In *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 60–69.

J M Ottino. 2004. Engineering complex systems. *Nature* 427, 6973 (Jan. 2004), 399.

ES Page. 1954. Continuous inspection schemes. *Biometrika* (1954), 100–115.

Thomas W Parks and C Sidney Burrus. 1987. *Digital filter design*. Wiley-Interscience.

Karl Pearson. 1900. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50, 302 (1900), 157–175.

Jan Procházka and Kamila Olševičová. 2015. Monitoring Lane Formation of Pedestrians: Emergence and Entropy. In *Intelligent Information and Database Systems*. Springer, 221–228.

Craig W Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, Vol. 21. ACM, 25–34.

Anil K Seth. 2008. Measuring emergence via nonlinear Granger causality.. In *ALIFE*, Vol. 2008. 545–552.

Vivek Singh, Gagan Singh, and Suparna Pande. 2013. Emergence, Self-Organization and Collective Intelligence–Modeling the Dynamics of Complex Collectives in Social and Organizational Settings. In *Computer Modelling and Simulation (UKSim), 2013 UKSim 15th International Conference on*. IEEE, 182–189.

Claudia Szabo and Yong Meng Teo. 2015. Formalization of Weak Emergence in Multiagent Systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 26, 1 (2015), 6.

Yong Meng Teo, Ba Linh Luong, and Claudia Szabo. 2013. Formalization of emergence in multi-agent systems. *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation - SIGSIM-PADS '13* (2013), 231.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

Uri Wilensky. 1999. NetLogo (and NetLogo user manual). *Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl. northwestern. edu/netlogo* (1999).

Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1 (1987), 37–52.

Tom De Wolf and Tom Holvoet. 2005. Emergence Versus Self-organisation : Different Concepts but Promising When Combined. *Engineering self-organising systems* (2005), 77–91.

APPENDIX

A. INTRODUCTION

This is the appendix for O'Toole et. al 2015. The main text of the paper describes DETect, a novel distributed algorithm for emergence detection in Complex Adaptive Systems. This appendix provides detail on two additional algorithms that are part of DETect's work-flow but are not integral to the goal of detecting emergence. These are described in Section B. In addition, this Appendix also presents an evaluation of DETect's model selection algorithm in Section C.

B. ALGORITHMS

B.1. Initialisation and Variable Observation

The initialisation phase and variable observation process in DETect is handled using a *Sliding Window*. An observation window of length N is used, that is updated when full by deleting the M oldest values. DETect uses a number of sliding windows throughout its work flow with the values of M and N changing depending on the window.

The initialisation and variable observation process on each agent is detailed in Algorithm 4. To improve readability, all Internal and External Variables of the agent are grouped into a single set, as the process applied to both is the same. DETect first initialises a pair of arrays for all variables used by the agent (lines 2-5). The $AvgArray$, is used to average a group of consecutive observations for each variable to improve data normality. Averaging is done as a preprocessing step for MLR, as normality is assumed in linear regression techniques. The second array is the sliding observation window $SlidingObsWindow$. The maximum length of $AvgArray$ and $SlidingObsWindow$ are specified by the parameters AWS and N respectively.

A new observation for each variable is received at each time step and is added to the relevant $AvgArray$ (line 7-9). The $AvgArrays$ for all variables have a common size as they are updated synchronously. Once the $AvgArrays$ is full (line 10), the mean value is calculated and added to the respective variable $SlidingObsWindow$ and the $AvgArray$ is cleared (lines 11-15). Next, the $SlidingObsWindow$ are checked to determine if they are full (line 16). If so, the windows for all variables are forwarded to either the $analyseModel$ (*cf.,* Algorithm 5) or $selectModel$ (*cf.,* Algorithm 1) functions. The $selectModel$ algorithm will return a boolean value indicating whether the model has been selected successfully (line 21). The sliding windows are then moved forward by deleting the earliest values, corresponding to the size of the jump the window should make, indicated by parameter M (lines 24-26).

ALGORITHM 4: DETect Initialisation and Variable Monitoring on Agent A_i

Input: All variables V_{i1} to V_{iv} of agent A_i. Averaging window size AWS. Sliding observation window size N. How far the Sliding Window should move once full M.
Output: The sliding observation window of variable V_i is filled

```
1  ModelSelected = FALSE
2  for V_i1 to V_iv do
3  |  Initialise empty AvgArray_iv;
4  |  Initialise empty SlidingObsWindow_iv;
5  end
6  for time steps 1 to t do
7  |  for V_i1 to V_ip do
8  |  |  Add Observation_ipt to AvgArray_ip;
9  |  end
10 |  if size(AvgArray_i1) == AWS then
11 |  |  for V_i1 to V_iv do
12 |  |  |  Add mean(AvgArray_iv) to SlidingObsWindow_iv;
13 |  |  |  Initialise empty AvgArray_iv
14 |  |  end
15 |  |  if size(SlidingObsWindow_i1) == N then
16 |  |  |  if ModelSelected then
17 |  |  |  |  analyseModel(SlidingObsWindow_i1 to SlidingObsWindow_iv)
18 |  |  |  end
19 |  |  |  else
20 |  |  |  |  ModelSelected = selectModel(SlidingObsWindow_i1 to SlidingObsWindow_iv)
21 |  |  |  end
22 |  |  |  for V_i1 to V_iv do
23 |  |  |  |  Delete SlidingObsWindow_iv[1 : M];
24 |  |  |  end
25 |  |  end
26 |  end
27 end
```

B.2. Model Analysis

After the model of the relationship between the agent and its environment has been selected, the sliding observation window becomes the $RegressionWindow$ and is reduced to the size of this parameter. The results of tests of various sizes of the $RegressionWindow$ are described in Section 6. The reason for this reduction in window size is to reflect the changed emphasis of DETect post-model selection. Selecting the model requires that a sufficiently broad sample of values for all variables are observed so that each internal-external variable relationship can be evaluated. Using a large sample decreases the likelihood that momentary random fluctuations will adversely affect the relationships are selected. However, once the model is selected DETect is concerned with detecting changes in the relationship quickly and this requires that a shorter time period is used for evaluation.

Once the model is selected, all subsequent instances when the Regression windows is full results in the selected model being analysed using MLR, as outlined in Algorithm 5. The *Model Analyser* component takes the set of internal variables chosen by the *Model Selector* and, in turn, fits a MLR model where Y is each internal variable, and the explanatory variables X are its chosen external variables (lines $5-12$). The output of this MLR is a set of p-values for each internal-external variable pair, with the p-values representing the significance of the relationship between the variables at that time (line 11). The p-values are forwarded to the *Change Detector Unit*, where they are monitored over time for change to determine if feedback is present in the environment. By collecting p-values over time, a model of the significance of a relationship can be formed, making it possible to compare against future measurements. For example, if

ALGORITHM 5: Relationship Modelling

Input: $modelMap$ containing all internal-external variable pairs used in the agents model. Internal Variables Regression Window SWI_i and External Variables Regression Window SWE_i of agent A_i.
Output: P-value time-series, $newP_ijs$ for all variable pairs contained in $modelMap$

```
modelInternalVars = Internal Variables in modelMap;
newPijs = empty matrix array;
for each modelInternalVars do
    Y = SWIij of modelInternalVarsj;
    X = empty array;
    for each Eis of modelInternalVarsj in modelMap do
        X = X + SWEis;
    end
    newPijs = MLR(Y,X);
end
```

the historical p-values for a relationship are consistently high, a sudden drop in the p-values in recent analysis will indicate that something has happened to change the significance level of the relationship.

C. MODEL SELECTION EVALUATION

To evaluate model selection, we included some additional internal and external variables monitored by agents in each of the three models. These additional variables provide no useful information to the agent, but increase the number of potential relationships that can be chosen. The performance of the *Modelling Unit* is assessed for both reducing the number of relationships to be monitored and selecting useful ones. The internal-external variable pair relationships selected by DETect for each agent's MLR model across all runs are recorded. Examples of both *useful* and *non-useful* variables for the traffic case study are outlined in the next section.

Model Selection: To evaluate model selection in DETect, we observe the variable pairs selected across all agents. Recall that some of the internal and external variables impact the agent's decision making and are useful, while some are static or vary randomly and are not useful. While useless variables are undesired, their inclusion in a model does not automatically undermine the model's fidelity, as the statistical relationship between the internal and external variables may not change over time. However, a model composed exclusively of these variables should be avoided, considering unnecessary computation cost. Therefore, model selection can be said to be effective if the following performance requirements are satisfied:

— Obj #1: For all agents, DETect selects a subset of all possible variable pairs.
— Obj #2: The number of useful variables selected is higher than non-useful and DETect should never select only non-useful variables for any agent

C.0.1. Model Selection. The MLR models selected across all agents for 60 simulations were analysed, as outlined in Table VI. In general, model selection performed well with a significant number of relationship pairs excluded in all models. The most efficient performance is in the Pedestrian model where DETect selected 3.14 variable pairs per agent from a possible 25. In both the Flocking and Traffic models, DETect never selected models composed only of non-useful internal or external variables. A small number of agents, 406 out of $22,920$, or 1.77% selected only models composed exclusively of non-useful internal variables, while almost 6% selected models composed exclusively of non-useful external variables. Although useful variables were almost always selected, DETect did include some non-useful variables of a high proportion of agents in each of the models. In Flocking, only 1.23% of agents had models composed entirely of useful variables, with Pedestrian and Traffic exhibiting slightly higher fig-

Table VI. Model Selection Results For All Models

	Flocking	**Pedestrian**	**Traffic**
No. of Agents	9,000	22,920	151,276
Total No. Variable Pairs	24	25	15
No. of Variable Pairs Selected	7.62	3.14	8.05
Agents with only Non-useful Variables	0.00%	0.00%	0.00%
Agents with only Non-useful Internal variables	0.00%	1.77%	0.00%
Agents with only Non-useful External variables	0.00%	5.91%	0.00%
Agents with only Useful variables	1.23%	11.83%	6.21%
Agents with only Useful internal variables	24.63%	82.52%	93.60%
Agents with only Useful external variables	2.84%	14.80%	6.53%

ures, 11.83% and 6.21% respectively. This was caused primarily by a failure to exclude all non-useful external variables with DETect achieving this in less than 15% of all agents in each simulation model. Nonetheless, DETect's feedback and emergence detection mechanisms were robust to the inclusion of these variables in the models, as illustrated below.

C.0.2. Discussion. Objective #1 is satisfied for each of the three models with a significant number of internal-external variable pairs excluded from the final MLR. Additionally, in both the flocking and traffic models, the selected model is never composed entirely of either non-useful internal variables or non-useful external variables, satisfying Objective #2. In the pedestrian model, for a small number of agents, 1.77% and 5.5%, DETect does select models with exclusively non-useful internal or external variables respectively. Moreover, for all models, the number of agents with non-useful variables included was high. However, this should be viewed in light of DETect's overall performance at both feedback detection and emergence detection, where their inclusion did not adversely affect the performance, especially in the Pedestrian and Traffic models. This demonstrates that DETect's feedback detection algorithm is robust to such inaccuracies. It is also possible that including some mis-information in the system model may help to improve performance [Nallur et al. 2015], however this requires some additional experimentation to fully determine.