Toward the Self-Organisation of Emergency Response Teams Based on Morphogenetic Network Growth

Nicolas Toussaint[1*], Emma Norling[2] and René Doursat[1]

[1]Centre for Advanced Computational Science, Manchester Metropolitan University, Manchester, UK
[2]Department of Computer Science, University of Sheffield, Sheffield, UK
[*]corresponding author: nicolas.toussaint@stu.mmu.ac.uk

Abstract

Focusing on the challenge of fostering the self-assembly of socio-technical networks, we present the application of Morphogenetic Engineering principles in this domain to a 2D spatial case study involving a team of first responders. Our model and simulation illustrate how members of a rescue team could be guided via hand-held devices toward better coordination and positioning at appropriate locations, based on peer-to-peer communication and local landmarks in the environment (such as incidents or exits), without the need for a centralised control centre. Using Raspberry Pi devices, we illustrate this scenario in various situations that require quick decision-making to control and manage. Our work suggests the possibility of novel forms of bottom-up self-organisation among groups of users and machines, in contrast to top-down imposed hierarchies and policies.

Introduction

Crises are inevitable. They range from large-scale wildfires to injured individuals in a crowd. The ability to respond and manage a crisis properly is crucial to limit casualties, prevent further damage, and help minimise subsequent risks. The specificities of each emergency scenario, which often includes misleading or incomplete information, make the coordination of rescue services particularly difficult. Compounding the problem, crisis situations can change rapidly as new events also occur unexpectedly (secondary hazards). Emergency responders must react and adapt to a dynamically shifting situation in real time by modifying their plans accordingly. For example, if someone breathes undetected toxic gas during an operation and is incapacitated, other agents must react by protecting themselves, evacuating everyone else from the new risk area, and take care of the injured. Detrimental interference among agents' actions may also arise from simultaneous task assignments leaving each other unfinished, or when a new urgency makes agents deviate from a global objective. Despite all these pitfalls, it is crucial to be able to constantly reevaluate the dynamic situation and make appropriate decisions toward reducing impact on human life and infrastructure.

So far, many research works focused on the amount of data and information that can be collected and processed in real time to improve the assessment of emergency situations and decision-making accuracy. The recent development of new technologies has contributed to making this endeavour increasingly affordable. New methods range from pervasive sensors found in everyday devices to fast communication and sophisticated AI algorithms able to aggregate multiple streams of data and recognise patterns. In an emergency context, the Internet of Things (IoT) is thought to have a positive impact (Yang et al., 2013), whether during preliminary evaluation of an incident, critical to a plan-based approach (Chen et al., 2008), or during the course of an event to track and monitor people's actions, locations, and considerable other information (Scheurer et al., 2017; O'Flynn et al., 2018; Ciabattoni et al., 2017).

For instance, 3D mapping of a disaster inferred from data gathered by bystanders (Sadhu et al., 2016) can help design an appropriate strategy. Indoor activity can be reconstructed from correlated sounds and inertial data captured with wearable devices, or outdoor drones. Such technologies could detect a first responder lying on the ground, coupled with perturbations in their vital signs, which would prompt their peers to treat this as a new incident and/or casualty. Active landmarks (mobile sensors) can also be placed by the agents (Palmieri et al., 2016) at the start of an operation to obtain a more precise picture of the unfolding incident. Information can also be extracted from social media feeds (Conrado et al., 2016). Smart systems have been designed to gather data from a variety of sources (Yuan and Detlor, 2005) and analyse the information they contain (Bartoli et al., 2015), paving the way toward an emergency response information system (ERIS) that can handle big data.

Information-gathering hardware and software systems also have shortcomings. Problems include difficulty to use, the need for special training and maintenance of key technological components, a high price tag, and lack of reliability in critical circumstances. Moreover, the flood of information sent to a main hub, the "Emergency Operations Centre" (EOC), from an ERIS can become rapidly overwhelming. This information is used centrally to design an appropriate strategy and decide which actions or movements need to be

undertaken to achieve an effective response. Task assignments are then decided and communicated back to the corresponding field agents. However, both decision-support systems and decision makers can be overburdened by the huge number of input and output features. The complexity of innovative information technologies often occupies too much of the EOC agents' cognitive space (Weidinger et al., 2018).

In this paper, we argue that self-organisation properties can be exploited and put to work to alleviate these difficulties. Endowing field agents with more autonomy and local power can greatly support an EOC toward achieving efficient coordination. In fact, reversing the terms, the EOC would only assist the bottom-up emergence on the ground, or be required when treating radically disruptive global events. Self-organisation properties allow the system to negotiate new environmental and organisational perturbations without falling into an "error state" (Serugendo, 2009), in which carefully crafted rules assigned to each agent have been rendered obsolete. Disturbances can involve disconnection of agents from the ERIS network, misunderstandings between agents, and so on. Typically, the responsibility to detect and identify such events would fall to the EOC (Chen et al., 2008), which would have to modify agents' instructions to adapt to the dynamically evolving situation. The measures taken should then steer the system back into a "rational state" in which coordination is considered efficient again. By contrast, self-organised systems are composed of many independent entities (here, field agents) interacting with each other and their local environment, and creating a collective behaviour that may also fulfil a goal such as the emergence of a functional structure. All entities follow the same set of instructions with possible variants. The outcome essentially relies on the multitude of local choices made by the agents via their interactions, instead of centrally assigned instructions unable to take into account sudden problems or failures.

There are only few examples of self-organised emergency response systems. One of them features adaptive collaboration without global awareness in a scenario of search and rescue, where dynamically changing factors determine the likelihood for help requests to be sent or accepted (Frasheri et al., 2018). In this work, however, the situation does not require the formation of a coherent modular network structure. Collective structure formation can be helped by models based on *gradient* propagation (Doursat, 2008; De Wolf and Holvoet, 2006), i.e. the peer-to-peer exchange and mutual update of positional information values. This concept inspired by developmental biology has been invoked to propose self-organising robotic swarms (Doursat and Sánchez, 2014) and socio-technical networks (Ulieru and Doursat, 2011), in which complex and persistent structures emerge from connectivity choices and "hop counters" propagated among nodes. A developmental approach based on self-assembly mechanisms controlled by the same ruleset or "genome" in every agent combines the advantages of robustness and adaptability of the growing structure.

While this article does not intend to offer an alternative to existing centralised systems, it aims to show how they could be complemented by a layer of decentralisation. It also illustrates the feasibility of applying an abstract model of network self-assembly to a more concrete spatial and distributed context via idealised emergency scenarios. Finally, it makes a step toward more practical use cases that employ Raspberry Pis (RPis) to represent devices carried by field agents. The rest of the text is organised as follows. First, we introduce the abstract model and its customisation to a 2D environment; then, we describe the experimental software/hardware setup of our application; finally, we discuss the results obtained so far and sketch out future work.

Model

In this section, we briefly introduce the abstract model of programmable network self-organisation (Ulieru and Doursat, 2011) on which the present work is based. Then, we present a customisation of this model to 2D space and its adaptation to a distributed external environment.

Abstract Model Overview

The core of the model is a feedback loop going through three steps corresponding to the three main components of network self-assembly: *links*, *ports* and *gradients*. All nodes of the network possess a predefined set of input/output pairs of ports, denoted by (X, X'), (Y, Y'), and so on. Each port is in one of the following states: 'closed' (not accepting connections), 'open' (accepting) or 'busy' (open but not accepting). A link can connect two nodes i and j via open and matching output/input ports only, e.g. $X'_i \leftrightarrow X_j$. Optional "port tagging" can further restrict connections during runtime by creating specialised port subtypes and mutual compatibility rules, e.g. X'^a_i cannot connect to X^b_j. Nodes exchange integer gradient values over the links, denoted by (x, x') and so on, to convey positional information. These values are hop counters that increase by 1 every time they cross over to another node. Each node calculates the minimum of all received values within a given port type (and other correction rules apply). A small program, or *ruleset*, is embedded in each node, taking in input local gradient and environmental values and opening/closing ports in output. The ratio of both values tunes the propensity of the structure to react to the environment. This common ruleset constitutes the "genome" of the network, as it controls the patterns of connectivity in each node based on information received from the neighbours—hence the "shape" of the network.

In sum: (a) gradient values are exchanged and updated (G step), possibly several times; then, (b) ports may be opened/closed depending on these values (P step); and (c) new nodes may be added/removed and linked/detached where ports are available/busy (L step); then these steps are

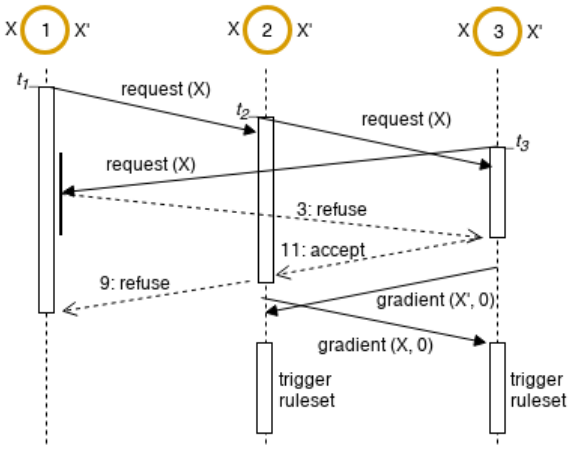

Figure 1: A request-sending scenario involving three nodes. Each answer is tagged with the corresponding line number from Algorithm 1. Here all nodes possess the same open ports X and X', and exchange requests to find matching ports to connect to. A request is sent to the opposite port of a recipient node, which may wait for an answer to its own request before replying in turn. In this case, node 1 declines a request as $t_3 > t_1$, triggering a cascade of other accept/decline answers. Eventually, a link is created between nodes 2 and 3, which then start exchanging gradient values.

Algorithm 1 Request Receiving

```
1:  procedure IS_VALID(timestamp, value)
2:      if (timestamp > local_timestamp or
          not has_compatible_port(value) or
          has_waiting_request) then
3:          return false
4:      else
5:          has_waiting_request ← true
6:          state ← wait_main_thread()
7:          if has_changed(state) then
8:              has_waiting_request ← false
9:              return false
10:         else
11:             return true
```

repeated. Complex deterministic structures can emerge from the change in time of all node states, guided by their common ruleset. The graph builds itself from the bottom up following the sequence of all alternatives that nodes face and the actions they take based on local information only.

Distribution and Concurrency

To prevent interference and race conditions among nodes during these steps, which would put the network into an inconsistent state, we also distinguish between different timescales. In this framework, gradient propagation G must perform faster than program execution P inside each node, itself faster than link creation L. The implication is that several rounds of G routines can be processed before one round of P is done; and similarly several P's before one L. Clearly, if it were not the case then concurrent changes in two different parts of the network could become incompatible, i.e. happen too fast for each one to take into account the other. While this could be remedied by running the algorithm on a central host based on a sequential node scheduler, it will not be robust in a truly asynchronous scenario where each node runs on its own device.

Once this temporal hierarchy is established, no concurrency issues remain within the first two steps, and further synchronisation is not required. Gradient updates and propagation G are atomic operations independent from the previous state, so that the specific order in which values are sent and received is not critical. Port operations P only change the nodes' internal states locally and do not immediately affect peer nodes (that is, before L is triggered). Link creation L, however, can still harbour race conditions since several nodes can be simultaneously sending and receiving requests for new connections and each outcome can influence subsequent behaviour. This may cause problems such as when a node accepts an incoming request at the same time that its outgoing request to create a link is accepted on the same port, yet the port can only hold one link; or when two competing ports receive links at the same time, whereas one should have caused the other to be closed and vice-versa.

To create new links, nodes regularly broadcast requests containing a list of their open ports to their peers. Recipients compare this list to their own available ports to identify matching pairs. If at least one match is found, the node sends back a positive response with the port identified and both nodes acknowledge the new link. To prevent concurrent link requests, we designed a priority mechanism (Algorithm 1) which assigns an order to these requests. It enables nodes to process them sequentially and individually, delaying responses for other link requests until completion of the current one. The sequential order is determined by two criteria (ordered by importance): 1) the origin of the requests ('sent' requests are ranked higher than 'received' requests); and 2) the time stamp (the older the request, the higher the ranking). Prioritising sent requests ensures that the ports advertised in the requests are kept open until the reception of the response. In the case where a node can accept a received request after sending its requests to other nodes, leading to port closures, recipients would then have to check that matching ports are still available before creating a link, generating additional communication. This criteria however only gives a local order, which can be contradictory across different points of view. For example, if node i sends a request to node j and vice-versa, and both are prioritising a different request, this will create a deadlock. The second criterion based on time stamps prevents this problem by creating a global priority order. Furthermore, to reduce unnecessary waiting, agents only set aside the first compatible request received while immediately declining the remaining requests.

Embedded in Space

While the previous model was designed at an abstract level for all kinds of graph topologies, we propose here to apply it to 2D Euclidean space. In this context, new common features are added such as the *positions* of nodes and objects, and an equation to *move* nodes in space, derived from distance-based interactions and forces among agents and environmental landmarks. Nodes represent agents in space and self-organise into complex shapes guided by their common genetic ruleset, with possible variations depending on the environmental cues they encounter. In the examples presented here, these external influences have been placed in such a way that the simple rules followed by the agents (e.g. connecting to peers, accepting requests, moving in certain directions) create "interesting" structures. In the remainder of the text, we use the term 'agent' indifferently to refer to a 'node' or its associated 'field agent'.

Interactions: Interactions between agents in 2D are the same as in the abstract method. Distances are not taken into account during communication as nodes are actually relying on a pervasive broadcasting system with ID-tagged messages to emulate point-to-point connections. However, this assumption could be weakened to constraint the channels of communication to a certain region of space based on a maximum radius of signal reception, which would impose limits on the possible combinations of nodes.

Agents are now also able to perceive high-level features (such as casualties, fire exits, etc.) in their local environment via *sensors*. Sensors are activated whenever specifically associated features are present in the detection range, in which case appropriate forces are applied on the agent. Depending on the "mode" of the sensors, these forces are either attractive, aiming to keep the agent within the proximity of a feature of interest, otherwise they are repulsive. Agents can also adjust external influences, following their ruleset, by changing a sensor mode as needed. For example, at first an incident can attract agents to encircle it; but when the area becomes overcrowded, rules change the sensor mode of unneeded agents and, as a result, scatter them over a large area. This strategy increases the region covered by perceptual agents, which improves the system's overall response to possible secondary hazards. An agent may also dispose of its sensors, i.e. desensitise itself to certain types of landmark and specialise its behaviour.

Movements: Agents move to create *spatial* chains and graphs that follow the corresponding abstract *topological* chains and graphs created by the G-P-L connectivity routines, and based on an optimal length for each link. For example, if there is a chain of three agents connected through their (X, X') ports, they should move in 2D space in such a way that the middle node positions itself between the other two (details below). In addition, nodes can also be attracted toward, or pushed away from, particular landmarks that they sense in the environment. Free agents disconnected from their peers can move in arbitrary ways. Combined together, neighbourhood and landmark interactions give rise to forces that reshape the network's spatial structure in accordance with both internal connectivity and external environment. However, converging to a global spatial layout that satisfies all these constraints can be impossible when they are contradictory, resulting in a frustrated state. In this case, a suboptimal equilibrium characterised by a nonzero but small global force can still be achieved. In sum, application of spatial rules is intertwined with the connectivity routines to ultimately drive the graph growth.

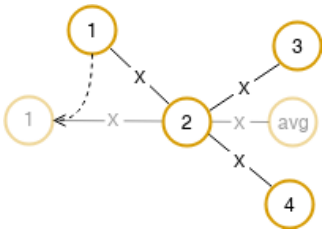

Figure 2: Example of local connectivity configuration. Node 1 only has one connection (on the X' port), while it requests the coordinates of node 2 and the centre of mass of nodes 3 and 4 in order to calculate its new position (see text). In this case, it ends up being on the other side of node 2 with respect to the other nodes.

The core of the motion algorithm is as follows: at each internal step, an agent i asks each one of its connected peers k to send (a) its position (denoted by $\overrightarrow{U}_k$ in Eq. 1) and (b) the average of the positions of k's "output" peers (from the viewpoint of i as an "input") via the same type of link (denoted by $\overrightarrow{U}'^{(i)}_k$). In the example of Figure 2, node 1 sends a positional request to node 2 via the X port; then node 2 asks its linked neighbours (here 3 and 4) their positions via the X' port. The total displacement vector of i is calculated from this information, possibly combined with the coordinates of certain landmarks $\overrightarrow{U}_l$ detected by the sensors:

$$\Delta\overrightarrow{U}_i = \frac{1}{n}\left(\sum_{k=1}^{n}\overrightarrow{V}_k + \alpha\overrightarrow{W}_k\right) + \frac{1}{m}\sum_{l=1}^{m}\beta\overrightarrow{W}_l \qquad (1)$$

where node $\overrightarrow{U}_i$ has n neighbours and m landmarks in its detection range; $\overrightarrow{V}_k$ and $\overrightarrow{W}_k$, described below, represent two types of displacements generated by "forces" among agents; $\overrightarrow{W}_l$ reflects the same second type of force applied to agents by landmarks; and the α and β factors (resp. 4 and 1 here), adjust the balance of these three constraints.

$$\overrightarrow{V}_k = \begin{cases} (\overrightarrow{U}_k - \overrightarrow{U}'^{(i)}_k) + (\overrightarrow{U}_k - \overrightarrow{U}_i) \text{ if } \overrightarrow{U}'^{(i)}_k \neq \overrightarrow{0} \\ \overrightarrow{0} \text{ otherwise} \end{cases} \qquad (2)$$

$\overrightarrow{V}_k$ drives node i toward a position symmetrically disposed from $\overrightarrow{U}'^{(i)}_k$ with respect to U_k (if U_i indeed has second-degree neighbours beyond k, otherwise $\overrightarrow{V}_k$ has no effect).

$$\overrightarrow{W}_k = \left(||\overrightarrow{U}_k - \overrightarrow{U}_i|| - d_0\right) \frac{\overrightarrow{U}_k - \overrightarrow{U}_i}{||\overrightarrow{U}_k - \overrightarrow{U}_i||} \quad (3)$$

$\overrightarrow{W}_k$ (resp. $\overrightarrow{W}_l$) is a vector pushing node i at an optimal distance d_0 from node k (resp. landmark l). This distance depends on the type of the element, whether it is a node or a landmark, and whether the latter is attractive or repulsive. In the case of repulsive landmarks, the distance is set to a high value such that the node is driven out of the landmark's detection range and influence.

Finally, the resulting displacement vector $\Delta\overrightarrow{U}_i$ is constrained by the motion capacity of the node represented by a maximum distance $d_{\max}$, and must be renormalised if it exceeds this distance:

$$\Delta\overrightarrow{U}_i \leftarrow \begin{cases} d_{\max}\frac{\Delta\overrightarrow{U}_i}{||\Delta\overrightarrow{U}_i||} \text{ if } ||\Delta\overrightarrow{U}_i|| > d_{\max} \\ \Delta\overrightarrow{U}_i \text{ otherwise} \end{cases} \quad (4)$$

Gradient loops: One potential problem with this model that must be avoided is the development of loops exclusively composed of one port type, as this would lead to an infinitely increasing gradient propagation. In the abstract model, where a single graph grows from the incremental addition of nodes, one at a time, blocking strategies can prevent this problem by closing specific ports at appropriate instants. For example, chains can be constrained to expand in only one direction (e.g. either X or X'), thus preventing both ends from connecting together and forming a loop. In the asynchronous 2D case, however, nodes are already all present as physical entities at the start, and several nodes may potentially trigger concurrent graph growths simultaneously. Therefore, embryonic structures at equivalent stages of development should be able to cooperate and merge into a larger graph to avoid unnecessary duplicates (e.g. several arcs of circular node chains forming around the same landmark). However, since port closing does not discriminate between loop development and graph merging, another mechanism must be implemented into the model to specifically prevent single-port loops. To this goal, we augment the gradient's information with the identifier of its source node. As a result, each extremity of a chain also possesses the node ID of the other end. The link creation routine L is also amended as follows: when a node broadcasts to its peers connection requests containing a list of its open ports, it also verifies for each peer that it is, precisely, *not* the source of the opposite gradient, otherwise that peer must be ignored.

Experimental Setup

In this section, we present a software and hardware implementation of the model described above with a case study to illustrate its behaviour. First, a reduced scenario including a small number of physical agents, here eight RPis, shows the applicability of the model to a distributed environment and how it can lead to a real-world application. Then, a more comprehensive setup using simulated agents demonstrates complex self-organisation. In both scenarios, we focus here on the main features of the interactions between agents and for now ignore realistic details such as obstacles.

Model Implementation

Throughout the remainder of the paper, a *software agent* (SA) refers to an autonomous program that can run individually on a field agent's device or among other SAs on a central computer. In both cases, SAs embody the nodes of our model, whereas *environmental agents* (EAs) represent external landmarks that can affect the settings or evolution of a simulation and its real-world deployment.

Environmental agents and EOC: In a full-fledged distributed system, each EA would only contain a partial description of the local physical environment surrounding a landmark. For the needs of our implementation, however, where generic RPis stand in for field devices but lack actual sensing/actuating capabilities, we revert some of the decentralisation principles and concentrate the EAs into a single EOC running on a laptop. This EOC also contains the common ruleset specified at the start and broadcast to all SAs. As both the list of landmarks and the agent's rules can change during the simulation, the latter is broadcast again while the former is kept centrally.

After each move, an SA sends to the EOC a list of its active sensors. The EOC responds with the new state of each sensor, associated with the landmarks' positions if any has been detected. Another role of the EOC is to display a global view of the current simulation to be observed and monitored by the modeller. To this purpose, software agents send a notification after every action (such as gradient value updates, link creations, movements, etc.), which are then aggregated by the EOC into a global picture.

In summary, while landmarks should not normally be provided by the EOC but detected directly by the field agents, the ability to continually broadcast ruleset updates and provide a visualisation panel remain useful features in a concrete scenario. The states and positions of field agents can be used alongside other sources of information by the EOC to improve the situational awareness of possible central decision-makers in real time. The broadcasting functionality enables the EOC to react to critical changes by overriding outdated rulesets.

Software agents: SAs are given a code that determines how they interact, process the information received and make decisions. While the architecture and program of the model's core G-P-L routines is immutable, the modeller can

Algorithm 2 Agent Ruleset

```
1: open X, close X′, Y, Y′
2: open C, close C′
3: turn_on S_c, S_x                          ▷ casualty and exit sensors
4: circle_size ← 8
5: if s_c = 0 then                           ▷ if no casualty detected
6:     open X
7:     set_attractor S_c
8: if s_c > 0 and x < circle_size then
9:     open X, X′
10:    close C, C′                           ▷ prioritise encircling
11:    if x > 0 then
12:        close Y
13:    if x = (circle_size −1) then
14:        open Y′, C′, X′-tagged-by-s_c
15:    if x′ = (circle_size −1) then
16:        open Y, X-tagged-by-s_c
17: if x ≥ circle_size and x′ = 0 then
18:    close X, X′, Y, Y′
19:    set_repellent S_c
20: if c = 0 then
21:    turn_off S_x
22: if c > 0 then
23:    open C′, turn_on S_x
24: if (s_x > 0 and c > 0) or (c = 0 and x = 0 and c′ > 0)
    then
25:    close C′                              ▷ adaptive chain length
```

focus on the "genomic" ruleset embedded inside P, which is specific to each use case. Unlike many agent-based modelling solutions, composed of a set of agents in a shared container and randomly activated by a scheduler, each individual SA is run in a process of its own. It also possesses its own graphical interface to display practical information to the field agent (more details in Results). For this experiment, we used versatile RPis to host the SAs. For real-world use by field agents, these devices could be later augmented with sensing capabilities or replaced by other digital assistants such as mobile phones. Here, sensor information related to the field agents themselves (such as GPS coordinates or orientation in space) is simulated locally.

Use Case Scenario

In our test scenario, a crowd has gathered in some venue to watch a community event. The venue is a static grid of dimensions 700×500 with an exit at position $(350, 10)$. Initially, field agents are stationed arbitrarily in space. They have a detection range of $d_r = 200$ and at each internal step they can cover a maximum distance of $d_{\max} = 20$. Their goal is to detect any incident and contain it by forming a circle to make space for the arrival of first aid. After completing the isolation process, the remaining field agents form a chain between the incident and the exit to facilitate the evacuation of the casualties. Here an incident happens at location $(400, 450)$, requiring eight agents to encircle it. Once the structure has stabilised, another incident is added at $(250, 150)$ that agents must address.

In this context, the ruleset (Algorithm 2) was designed so that the agents' connectivity behaviour ultimately solves the above challenge (Fig. 3). To this goal, they are endowed with three pairs of ports: (X, X'), (Y, Y') and (C, C'), alongside two high-level sensors: S_c, activated when a casualty is inside the agent's detection range, and S_x, triggered by an exit. Both X and C ports are opened first so that idle agents can contribute to a circle (connected via X) or a chain (via C). When an agent is part of a growing circle, both X and X' are open, enabling concurrent circles to merge together and new idle agents to be added in no particular order. When the X-chain reaches a sufficient size, appropriate nodes connect via Y to complete the circle (lines 13-16). Furthermore, the (X, X') ports are tagged with the value of S_c, i.e. the casualty's ID, restricting possible connections to peers near the same casualty. If too many agents are around the casualty, excess agents disconnect themselves from the structure and move away (lines 17-19). After the circle containment process is complete, and if there are other agents remaining, the C' port is opened to trigger the growth of a chain in the direction of an exit (Fig. 4).

Results

First, we implemented this scenario in eight RPis (Fig. 3a,b) and a central laptop (Fig. 3a,c) to observe the model's behaviour in a more concrete context. Since the small number of agents was just enough to encircle the first casualty, the scenario was stopped at this point (no exit chain was formed). Each RPi independently executed one SA, while the computer held the EOC and list of EAs (here, one casualty and one exit). Figure 3b shows screenshots of the subjective views from three of the eight devices. This visualisation interface is composed of the local environment and subgraph perceived by the corresponding field agent (e.g. landmarks and connected peers) along with personal information, such as its connectivity state. The top bar also displays the direction and distance to the next calculated position.

Through this first trial, we wanted to evaluate the collective ability of agents to consistently develop a robust shape over multiple runs, solely based on their local actions. SAs were not assigned predetermined locations neither in 2D nor in the graph; instead, they were given the same program (including Algorithm 2 at the core of the P routine) and were driven via these rules and their interactions toward forming a specific and stable (albeit temporary) network topology. Moreover, although the order of link requests, sensor activations and, ultimately, graph nodes was not deterministic and produced significant variations at each new simulation, the final overall circle structure was reliably the same.

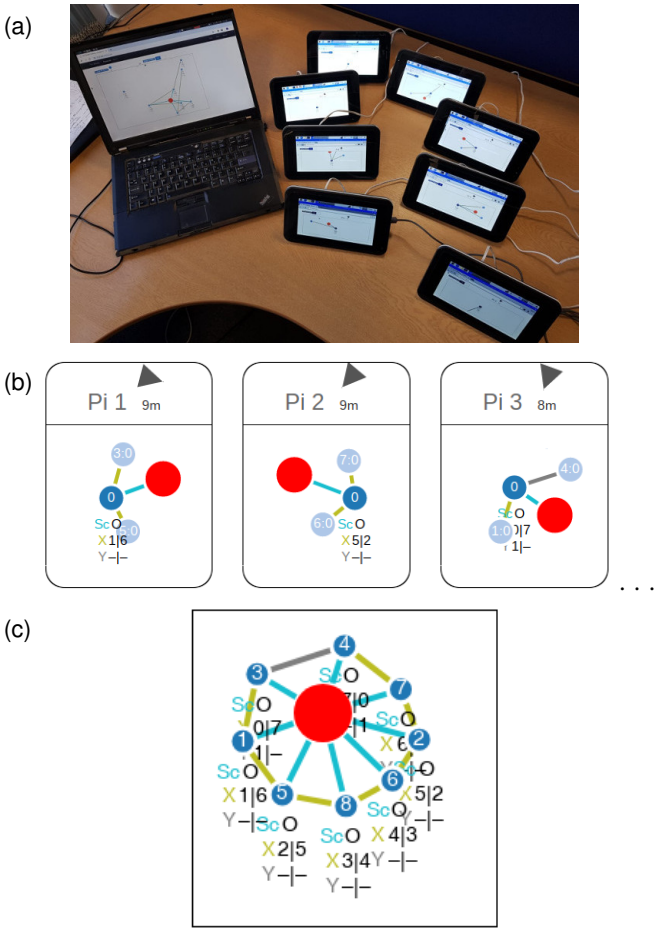

Figure 3: First experiment with eight RPis. Agents (in dark blue) discover an incident (in red) and encircle it. Links represent detection via sensors (in cyan) or interactions with other agents (in green). (a) Physical setup: eight RPi's communicate with each other and a laptop. (b) Three of the agents' subjective views, showing the local connectivity and landmark, as displayed on each device. Directions to reach the next calculated position are featured in the top bar. (c) Global view on the central computer. Both types of views also indicate the state of ports and gradient values.

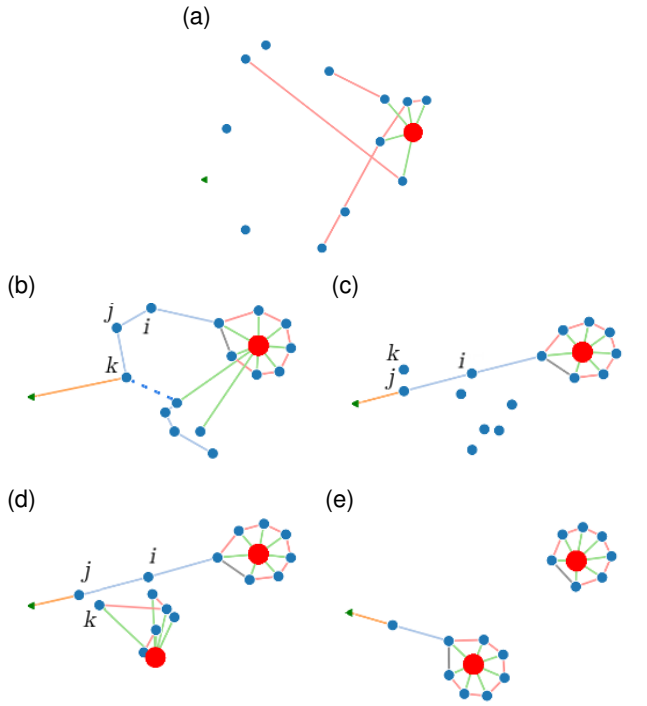

Figure 4: Timeline of a simulation with 15 agents on one machine. (a) Agents are randomly scattered in 2D space (not all visible here), then an incident happens (in red): the nearest agents detect it (green links) and start sending requests to peers to contain it (red links). (b) The incident is encircled, then a meandering chain (blue links) has formed past the exit (green triangle) and breaks up (dashed link). (c) Part of the chain straightens while other nodes detach from it. (d) A second event requisitions the idle agents. (e) End of the simulation: the two incidents are contained.

In a second experiment, 15 SAs were simulated on a single machine (Fig. 4). While this setup is clearly not equivalent to the constraints of running on multiple asynchronous devices, it nevertheless resulted in the same type of circle and chain structures. As for the precedent evaluation, the resulting global structures were consistent throughout the runs. Figures 4c,e are representative of these graphs, while intermediate steps differ according to the specific agents' interactions and locations. Based on this, we could make a few interesting observations. First, the ability to collectively react and grow appropriate graphs is triggered by the detection of landmarks. Rules are designed to prevent matching pairs of ports from connecting when agents are roaming freely in the initial state (lines 1, 2). This situation is unlocked whenever a casualty sensor is activated (lines 8-16). Here, when an incident occurs (red disc), five field agents detect it (green links) and open their X' port (Fig. 4a). This port acts as a recruiter attracting more field agents toward the point of interest. Once sufficiently near, agents behave similarly and, if enough are present, the circle is closed (Fig. 4b). Here, two extra agents are still within detection range of the event (longer green links) but are moving away.

Then, the relative influence of internal gradient values with respect to external events (sensor values) is reflected in the ruleset. Here the circle formation results purely from the former, whereas its particular location is guided by the latter. The additional chain structure also stems from the port logic, whereas its exact length varies with the distance between landmarks. After encirclement, the other agents form a long 7-node chain (Fig. 4b) until node k comes within detection range of the exit (green arrow). This causes k to disconnect from its successor (dashed link) and break up the chain. The loose end of the chain disassembles, while forces among the remaining consecutive nodes make the chain straighten (Fig. 4c). This causes j to draw near to the exit and connect directly to it, which in turn releases k and allows it to participate in other events (Fig. 4d,e).

Finally, priorities among specific parts of the structure can be hard-coded in the rules. For example, a node belonging to a circle could close its C port (line 10) to avoid being enrolled in a chain. On the contrary, a node belonging to a chain keeps its X port open so that it stays available for a circle. This type of rule asymmetry results in promoting circles before chains to represent the fact that containment is considered more important than evacuation in this scenario. In fact, after forming the first chain, node i was poached and recruited into a new circle around the second casualty, while j remained chained to the exit and i (Fig. 4e).

Conclusion and Future Work

In this paper we showed how agents could self-organise and form consistent structures according to their environment without top-down directions from a central control. The agents adapted their behaviour as a result of local or distant events, without the need to assess their own usefulness. For instance, if a global sight of the whole circle is blocked by smoke or because it is too big, appropriate assignments and positions are automatically computed by the model.

We presented an example where a first responder team deployed in a simplified representation of some venue provides protection to possible casualties and facilitates emergency evacuation. A reduced scenario with a small number of RPis was performed to verify the model's behaviour in a context closer to reality. We also simulated an extended version to highlight interesting properties and the ability of the model to accommodate a second casualty without the need to overwrite the rules. Furthermore, as coordination is handled by the port and gradient logic, the field agents are better able to focus on concrete value-added actions and less preoccupied about coordination.

For future work, several directions can be envisioned. The model could be integrated into a crowd simulator to analyse its behaviour in a more complex environment. The use of real floor plans instead of a square lattice to represent the environment would also help reduce the gap between reality and simulation. Field agent roles could be diversified (e.g. police, paramedic) to further regulate interactions based on expertise, while different types of sensors (e.g. toxic gas detector) could be carried only by certain agents. Link requests could also be limited to the nearest neighbours, instead of broadcast to all, to reduce the need for field agents to move in order to accommodate the forces resulting from the structure.

Acknowledgements

NT's thesis work, supervised by RD, is supported by the School of Computing, Mathematics and Digital Technology at Manchester Metropolitan University. NT wishes to thank PhD fellow Zoe Bartlett for her help reviewing and correcting the English, and co-supervisors EN and Bruce Edmonds for continued advice throughout his work.

References

Bartoli, G. et al. (2015). A novel emergency management platform for smart public safety. *Int. J. Commun. Syst.*, 28(5):928–943.

Chen, R. et al. (2008). Coordination in emergency response management. *Commun. ACM*, 51(5):66.

Ciabattoni, L. et al. (2017). Real time indoor localization integrating a model based pedestrian dead reckoning on smartphone and BLE beacons. *J. Amb. Intel. Hum. Comp.*, pp1–12.

Conrado, S. P. et al. (2016). Managing social media uncertainty to support the decision making process during emergencies. *J. Decis. Sys.*, 25(sup1):171–181.

De Wolf, T. and Holvoet, T. (2006). Design patterns for decentralised coordination in self-organising emergent systems. In *Int. Workshop Engineering Self-Org. Applications*, pp. 28–49. Springer.

Doursat, R. (2008). Programmable architectures that are complex and self-organized-from morphogenesis to engineering. In *ALife 2008*, pp181–188.

Doursat, R. and Sánchez, C. (2014). Growing fine-grained multicellular robots. *Soft Robotics*, 1(2):110–121.

Frasheri, M. et al. (2018). Adaptive autonomy in a search and rescue scenario. In *SASO 2018*, pp150–155. IEEE.

O'Flynn, B. et al. (2018). First responders occupancy, activity and vital signs monitoring. *Int. J. Adv. Net. Services*.

Palmieri, F. et al. (2016). A cloud-based architecture for emergency management and first responders localization in smart city environments. *Comput. Electr. Eng.*, 56:810–830.

Sadhu, V. et al. (2016). Argus: Smartphone-enabled human cooperation via multi-agent reinforcement learning for disaster situational awareness. In *2016 IEEE Int. Conf. Auton. Comp.*, pp251–256. IEEE.

Scheurer, S. et al. (2017). Sensor and feature selection for an emergency first responders activity recognition system. In *2017 IEEE SENSORS*, pp1–3. IEEE.

Serugendo, G. D. M. (2009). Robustness and dependability of self-organizing systems-a safety engineering perspective. In *Symp. Self-Stab. Sys.*, pp254–268.

Ulieru, M. and Doursat, R. (2011). Emergent engineering: a radical paradigm shift. *Int. J. Autono. Adap. Commun. Sys.*, 4(1):39.

Weidinger, J. et al. (2018). Is the frontier shifting into the right direction? a qualitative analysis of acceptance factors for novel firefighter information technologies. *Inform. Syst. Front.*, 20(4):669–692.

Yang, L. et al. (2013). How the internet of things technology enhances emergency response operations. *Technol. Forecast. Soc.*, 80(9):1854–1867.

Yuan, Y. and Detlor, B. (2005). Intelligent mobile crisis response systems. *Communications of the ACM*, 48(2):95–98.